Csci 4131
Internet Programming: HTTP

Lecture 13, October 17th

Fall 18

Dr. Dan Challou

Logistics

- HW 4 – building a small HTTP server using Pyton is due next Friday, **October 26th, at 2pm**
- You have to understand the HTTP protocol – which is covered in the following readings. I**f you haven't done the readings, you need to.**
  - http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html
  - https://tools.ietf.org/html/rfc2616
  - http://www.w3c.org/Protocols/

Last Time

- Overview of HTTP Protocol
- Review URL's
- Where HTTP Fits in in Network Protocols (Application Layer, built on TCP-IP)
- Reviewed Echo Server – program – a server that operates at the presentation level (socket)

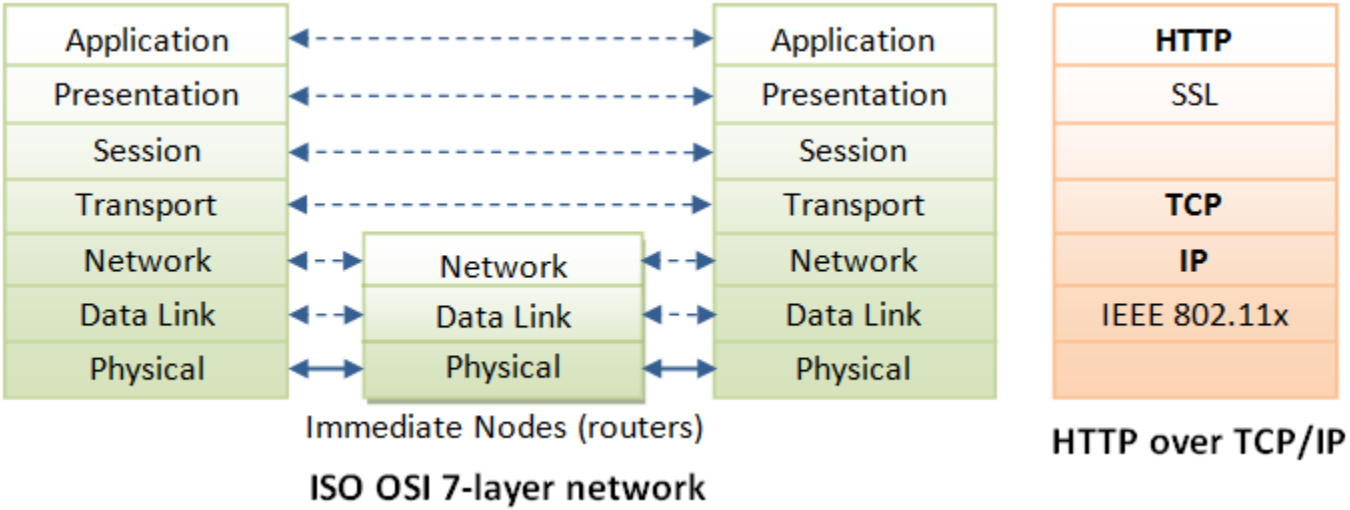
Questions?

Today

- HTTP – “Up close and Personal”
- Build a very limited HTTP server capable of responding to a HEAD request (HW 4 revisited)

Last Time: HTTP over TCP/IP

- HTTP is a client-server application-level protocol.
- It typically runs over a TCP/IP connection, as illustrated below.

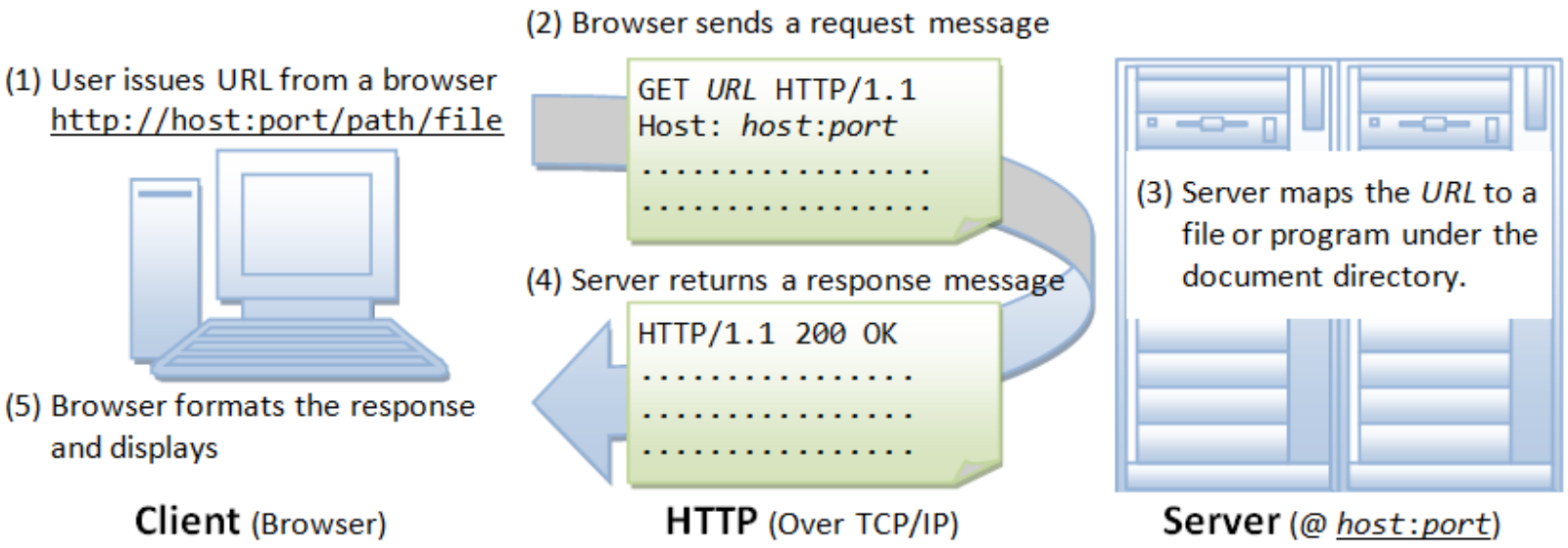


- The EchoClient and EchoServer Programs, posted on the class Moodle site and reviewed last class) operate at the **Presentation** level
- Today, We'll refactor the EchoServer program to respond to an HTTP HEAD request and post that to Moodle for your information as well

Echo Client / Echo Server Revisited

- Layer 6

How the HTTP Procol Works



Details, Details

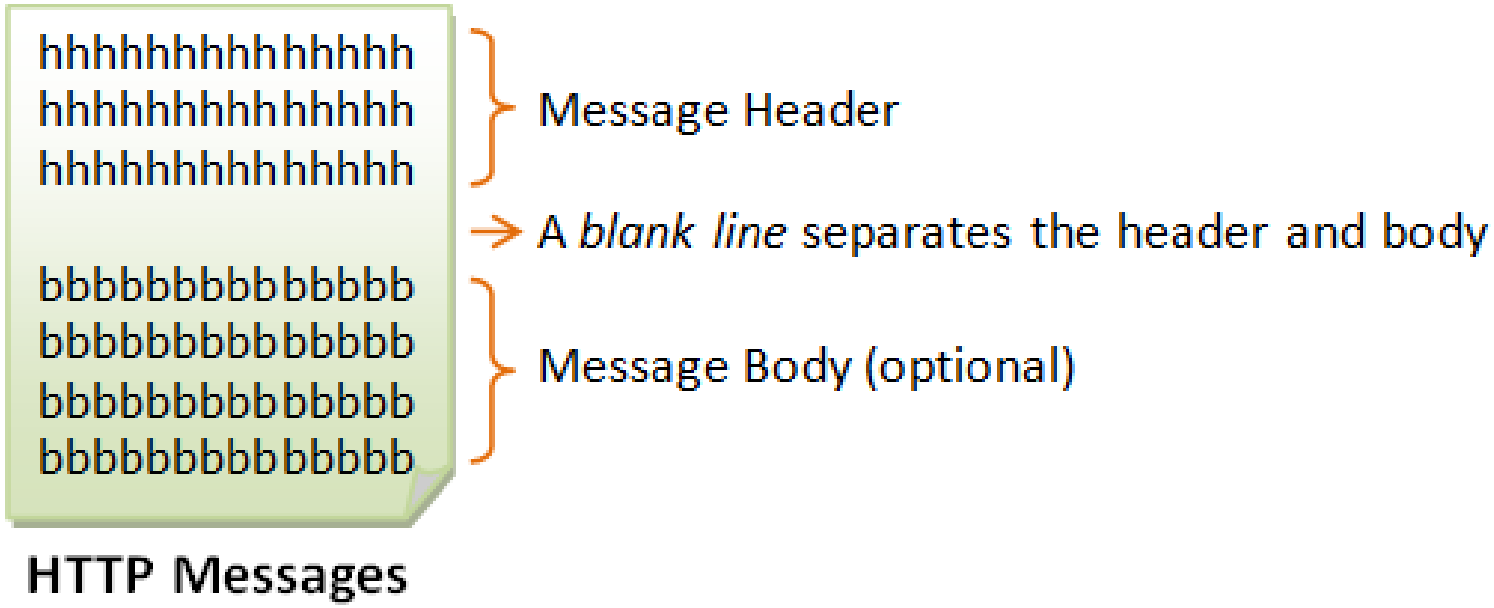
Details on the HTTP protocol.

Do at least the first reading for a fairly detailed overview

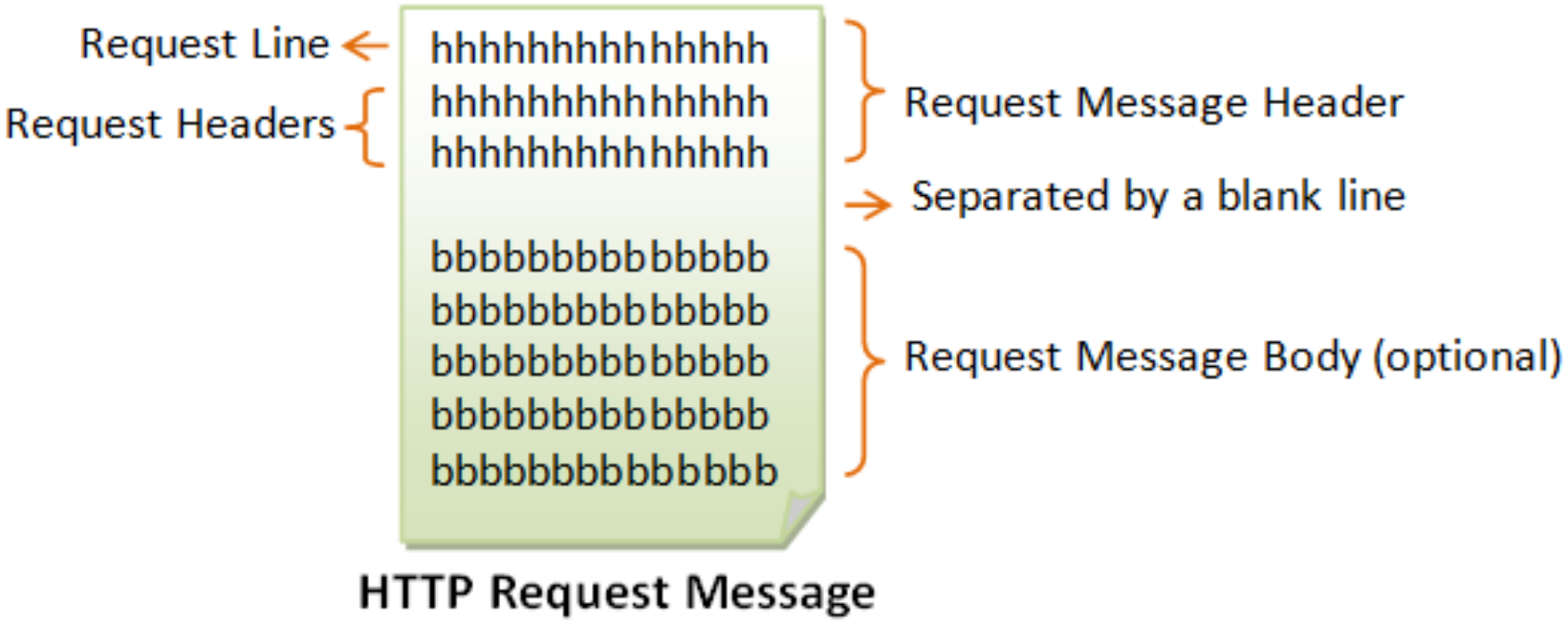
See RFC2161 for in-depth discussion of details of the contents of HTTP request and response Messages

HTTP Request and Response Messages

An HTTP message consists of a *message header* and an optional *message body*, separated by a *blank line*, as illustrated:



Format of HTTP Request Message



HTTP Request – The Request Line (First Line in the Request)

The first line is the REQUEST LINE, and it contains three items:

1. Name of the requested operation.
2. Request-URI (specifying the resource).
3. HTTP version.

In HTTP, message communication is built upon MIME (Multipurpose Internet Message Extension) format.

Request Line in Request Message

- The first line of the header is called the *request line*, followed by optional *request headers*.
- The request line has the following syntax:

  request-method-name request-URI HTTP-version

  - *request-method-name*: HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS. The client can use one of these methods to send a request to the server.
  - *request-URI*: specifies the resource requested.
  - *HTTP-version*: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.

- Examples of request line are:

  GET /test.html HTTP/1.1

  HEAD /query.html HTTP/1.0

  POST /index.html HTTP/1.1

HTTP Request Methods

- GET: A client can use the GET request to get a web resource from the server.
- HEAD: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
- POST: Used to post data up to the web server.
- PUT: Ask the server to store the data.
- DELETE: Ask the server to delete the data.
- TRACE: Ask the server to return a diagnostic trace of the actions it takes.
- OPTIONS: Ask the server to return the list of request methods it supports.
- CONNECT: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.
- Other extension methods

Which Methods will you be implementing in your Server (for your HW4)?

What is the most commonly used HTTP Request Method?

Request Headers

- Request headers are in the form of name:value pairs. Multiple values, separated by commas, can be specified.

request-header-name: request-header-value1, request-header-value2, ...

- Examples of request headers are:

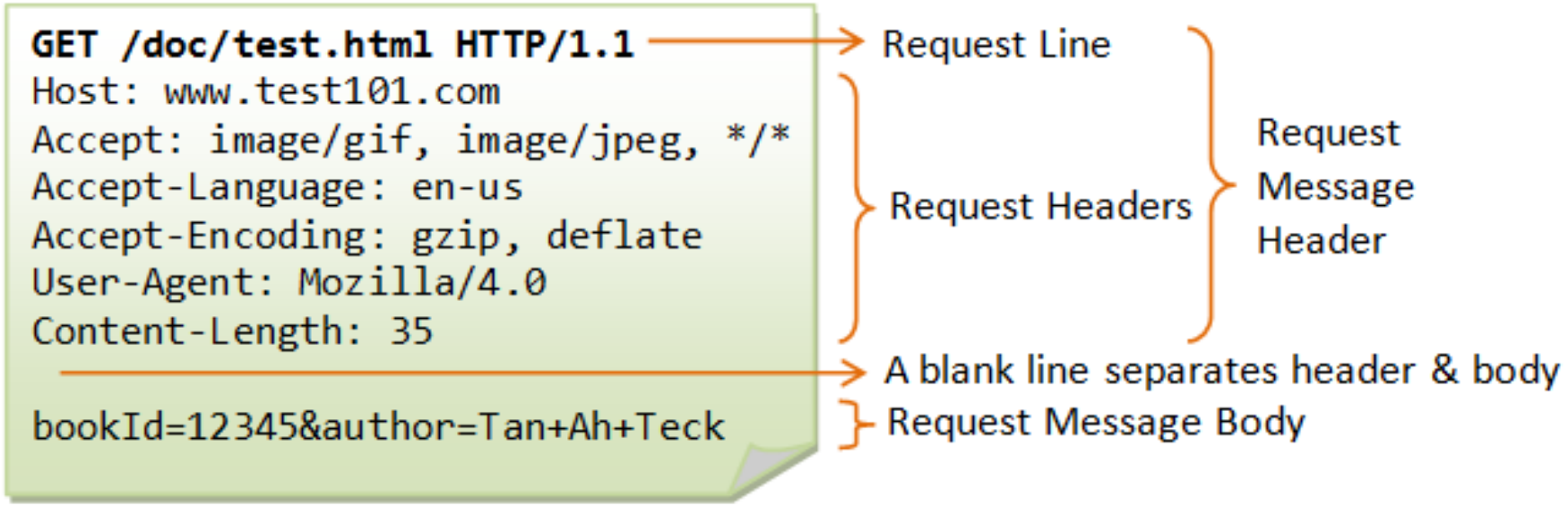
  Host: www.xyz.com

  Connection: Keep-Alive

  Accept: image/gif, image/jpeg, */*

  Accept-Language: us-en, fr, cn

HTTP Request Message Example



Question

- Example(s) of programs that construct and send HTTP requests to HTTP Severs (Like Apache, Microsoft, and the one you are building for HW 4)?

Programs that can send requests to severs (you can use to test YOUR server)

> telnet

telnet> help

... telnet help menu ...

telnet> open 127.0.0.1 8000 *(127.0.0.1 = localhost)*

Connecting To 127.0.0.1...

GET /index.html HTTP/1.0

(Hit enter twice to send the terminating blank line ...)

... HTTP response message ...

Telnet is a character based protocol – no typos when entering a command

OR Can use a Network Program

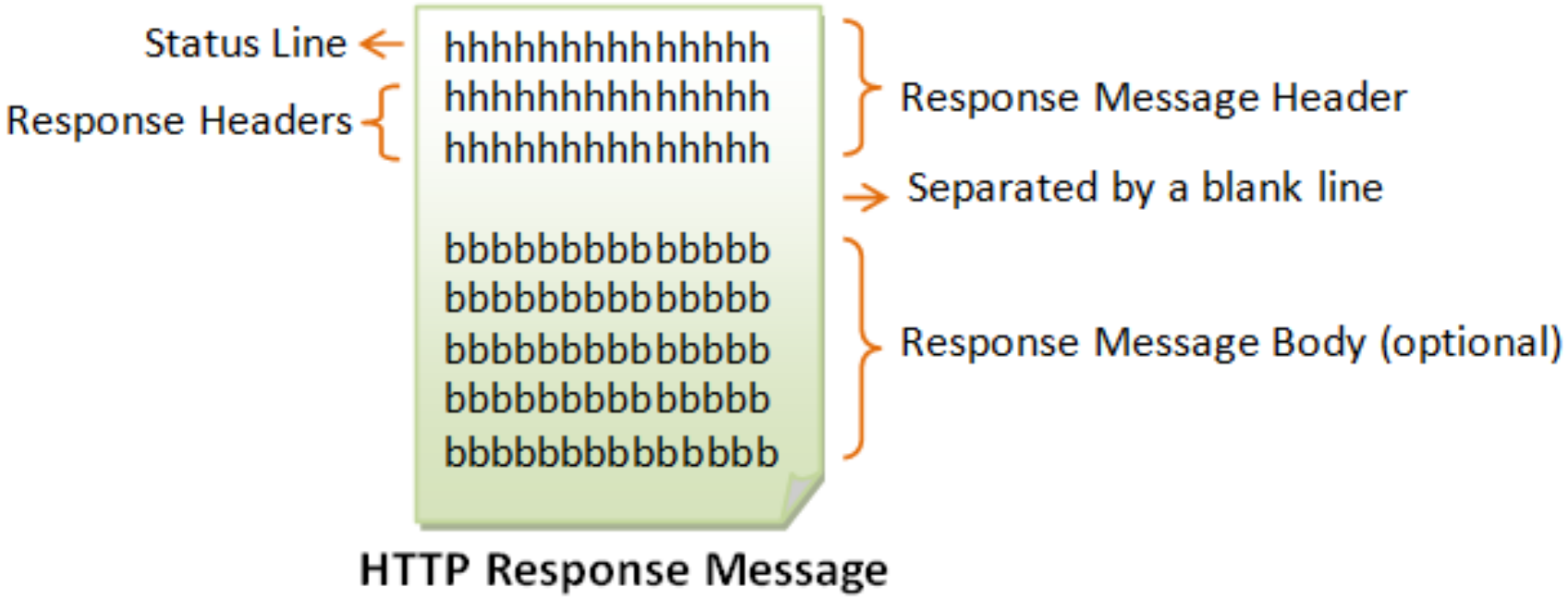
```
import java.net.*;
import java.io.*;

public class HttpClient {
  public static void main(String[] args) throws IOException {
    // The host and port to be connected.
    String host = "127.0.0.1";
    int port = 8000;
    // Create a TCP socket and connect to the host:port.
    Socket socket = new Socket(host, port);
    // Create the input and output streams for the network socket.
    BufferedReader in
      = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
    PrintWriter out
      = new PrintWriter(socket.getOutputStream(), true);
    // Send request to the HTTP server.
    out.println("GET /index.html HTTP/1.0");
    out.println();   // blank line separating header & body
    out.flush();
    // Read the response and display on console.
    String line;
    // readLine() returns null if server close the network socket.
    while((line = in.readLine()) != null) {
      System.out.println(line);
    }
    // Close the I/O streams.
    in.close();
    out.close();
  }
}
```

Or, can use

- Unix/Linux **cURL** command, which supports HTTP scripting
- Or Unix / Linux **wget** command
- Or can use Chrome POSTMAN application – need to add to your browser on the CSE Labs machines, but chrome apps are going away soon
- Or you can use POSTMAN application on your own machines.
- BUT, you must test your stuff on CSE Labs machines to ensure it works on those machines, so we can grade your HW

Or, any Web Browser!

HTTP Response Message (From the Server)



Status Line in Response Message (First Line in the Response Message)

- The first line is called the *status line*, followed by optional response header(s).

- The status line has the following syntax:

  HTTP-version status-code reason-phrase

  - *HTTP-version*: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
  - *status-code*: a 3-digit number generated by the server to reflect the outcome of the request.
  - *reason-phrase*: gives a short explanation to the status code.

- Common status code and reason phrase are "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".

- Examples of status line are:

  HTTP/1.1 200 OK
  HTTP/1.0 404 Not Found
  HTTP/1.1 403 Forbidden

Response Status Codes

- The first line of the response message (i.e., the status line) contains the response status code, which is generated by the server to indicate the outcome of the request.
- The status code is a 3-digit number:

  1xx (Informational): Request received, server is continuing the process.

  2xx (Success): The request was successfully received, understood, accepted and serviced.

  3xx (Redirection): Further action must be taken in order to complete the request.

  4xx (Client Error): The request contains bad syntax or cannot be understood.

  5xx (Server Error): The server failed to fulfill an apparently valid request.

Response Status Codes – Error Codes

- 100 Continue: The server received the request and in the process of giving the response.
- 200 OK: The request is fulfilled.
- 301 Move Permanently: The resource requested for has been permanently moved to a new location. The URL of the new location is given in the response header called Location. The client should issue a new request to the new location. Application should update all references to this new location.
- 302 Found & Redirect (or Move Temporarily): Same as 301, but the new location is temporarily in nature. The client should issue a new request, but applications need not update the references.
- 304 Not Modified: In response to the If-Modified-Since conditional GET request, the server notifies that the resource requested has not been modified.
- 400 Bad Request: Server could not interpret or understand the request, probably syntax error in the request message.
- 401 Authentication Required: The requested resource is protected, and require client's credential (username/password). The client should re-submit the request with his credential (username/password).
- 403 Forbidden: Server refuses to supply the resource, regardless of identity of client.
- 404 Not Found: The requested resource cannot be found in the server.
- 405 Method Not Allowed: The request method used, e.g., POST, PUT, DELETE, is a valid method. However, the server does not allow that method for the resource requested.
- 408 Request Timeout:
- 414 Request URI too Large:
- 500 Internal Server Error: Server is confused, often caused by an error in the server-side program responding to the request.
- 501 Method Not Implemented: The request method used is invalid (could be caused by a typing error, e.g., "GET" misspell as "Get").
- 502 Bad Gateway: Proxy or Gateway indicates that it receives a bad response from the upstream server.
- 503 Service Unavailable: Server cannot response due to overloading or maintenance. The client can try again later.
- 504 Gateway Timeout: Proxy or Gateway indicates that it receives a timeout from an upstream server.

Notes – for real Web Servers (not the HW 4 Server)

- The request method name "GET" is case sensitive, and must be in uppercase.
- If the request method name was incorrectly spelled, the server would return an error message "501 Method Not Implemented".
- If the request method name is not allowed, the server will return an error message "405 Method Not Allowed". E.g., DELETE is a valid method name, but may not be allowed (or implemented) by the server.
- If the *request-URI* does not exist, the server will return an error message "404 Not Found". You have to issue a proper *request-URI*, beginning from the document root "/". Otherwise, the server would return an error message "400 Bad Request".
- If the *HTTP-version* is missing or incorrect, the server will return an error message "400 Bad Request".
- In HTTP/1.0, by default, the server closes the TCP connection after the response is delivered. If you use telnet to connect to the server, the message "Connection to host lost" appears immediately after the response body is received. You could use an optional request header "Connection: Keep-Alive" to request for a persistent (or keep-alive) connection, so that another request can be sent through the same TCP connection to achieve better network efficiency. On the other hand, HTTP/1.1 uses keep-alive connection as default.

Response Headers

- The response headers are in the form name:value pairs

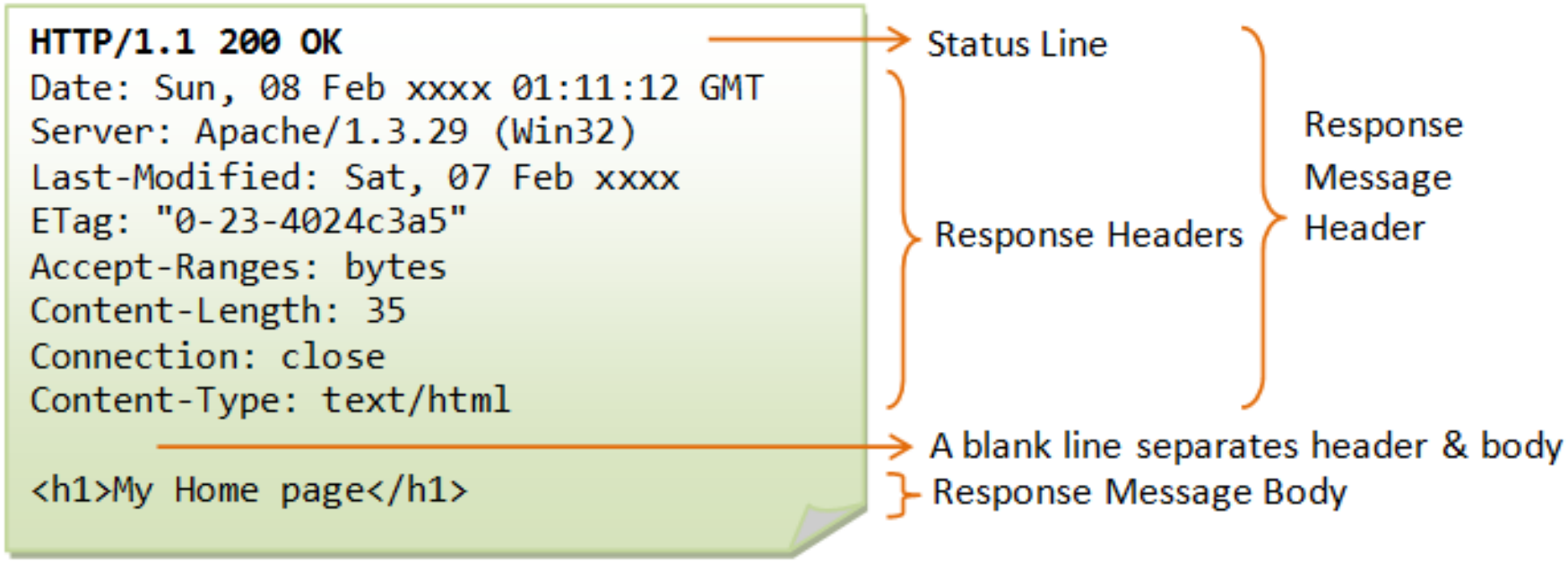
  response-header-name: response-header-value1, response-header-value2, …

- Examples of response headers are:

  Content-Type: text/html
  Content-Length: 35
  Connection: Keep-Alive
  Keep-Alive: timeout=15, max=100

- The response message body contains the resource data requested.

Example Response Message (in response to a GET request)



Question

- Examples of Programs that build HTTP response messages?

MicroSoft, Apache top the list

- https://news.netcraft.com/archives/2017/09/11/september-2017-web-server-survey.html

- Your HW4 will also be on the list…

HW 4, Revisited

- Build a limited-functionality HTTP server by layering the functionality to deal with the HTTP protocol on top of the presentation layer server
- Your server will have to receive request messages (GET, HEAD, POST)
  - Figure out what resource to find and return to the requesting client
  - Find the Resource
  - Return the resource (Compose a Response Message)
- OR
  - Return an error! (In a Response Message)

HW Assignment 4, Revisited

- Build a simple HTTP server in python
  - Example:
  - Refactoring EchoServer.py to respond to a HEAD request

Helpful Headers You can use

- When composing response messages, you can use the following PYTHON constants

CRLF = '\r\n'

1. **OK** = 'HTTP/1.1 200 OK{}{}{}'.format(CRLF,CRLF,CRLF)

2. **NOT_FOUND** = 'HTTP/1.1 404 NOT FOUND{}Connection: close{}{}'.format(CRLF,CRLF,CRLF)

3. **FORBIDDEN** = 'HTTP/1.1 403 FORBIDDEN{}Connection: close{}{}'.format(CRLF,CRLF,CRLF)

4. **METHOD_NOT_ALLOWED** = 'HTTP/1.1 405 METHOD NOT ALLOWED{}Allow: GET, HEAD, POST {}Connection: close{}{}'.format(CRLF, CRLF, CRLF, CRLF)

4. **MOVED_PERMANENTLY** = 'HTTP/1.1 301 MOVED PERMANENTLY{}Location: https://www.cs.umn.edu/{}Connection: close{}{}'.format(CRLF,CRLF,CRLF,CRLF)

You can figure out the others – e.g. 406…

Python Resources

- https://www.python.org – to download python to your machine so you can develop and run Assignment 4 – note, our target machines are ubuntu, and our examples have been developed and tested on the cse labs machines. That is where we will be testing, and grading your programs. They have to work correctly on the CSE LAB machines – you will only receive credit based on how your program runs there.

Next Time

- Moving on to XML, JSON, Ajax