Csci 4131
Internet Programming:
HTTP
TCIP

Lecture 5, February 26th

Spring 19

Dr. Dan Challou

Logistics

- HW 4 – building a small HTTP server using Python is out: Due date is Friday, **March 8th** at **3:00 PM**
- You'll need to understand the protocol – which both the readings (on the schedule and in the assignment) and lecture will cover
- Readings on the class schedule (from last week and this week):
  - http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html
  - http://www.w3c.org/Protocols/

Other Necessary Preparations for enabling your understanding of HTTP: Getting Setup for Python 3.x – which you will use to do HW4

- https://www.python.org – to download python to your machine so you can develop and run HW 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. This site can also be searched for documentation on the libraries we are using in the examples….

- https://docs.python.org/3.2 - documentation on python 3.2

- https://docs.python.org/3.2/library/socket.html - documentation on the socket library – which will help you understand EchoClient and EchoServer python programs (which I'll post on moodle for your review and refactoring for HW Assignment 4)

Learning or Refreshing your Python

- https://docs.python.org/3/tutorial/
- https://www.learnpython.org/
- https://www.lynda.com/Python-tutorials
  (need your x500 and umn password to login to Lynda – lots of Python Tutorials there)

Solutions to HW Problems

- Are available for your review after due date
- Go to any office hour to review them
  - No Pictures
  - No Video
  - No Electronic Copies
  - You can copy with Pen/Pencil and Paper

HW 3 Grades will be posted soon

- Bonus points earned for any assignment are posted in a separate column (next to the assignment grade) on Moodle
- Note – We use Moodle to report your grades on each assignment – it is not used to calculate your grade for the class, that is done using the weights and scale specified in the Syllabus

Upcoming: Node.js

Introduction to Node.js (Building a Webserver in JavaScript)

https://www.w3schools.com/nodejs/

https://codeburst.io/the-only-nodejs-introduction-youll-ever-need-d969a47ef219

Last Time

- JavaScript Closures
- A detailed look at DOM events and interaction with JavaScript
- Started discussion on another Key aspect/technology of the WWW – The HTTP Protocol:
  - Revisited URLs (and URIs and URNs)

Questions?

Review Exercises From Last Lecture

Today

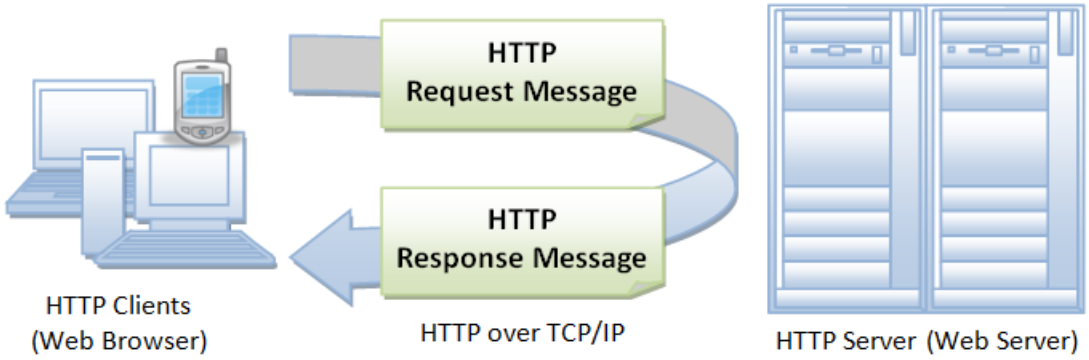
- HTTP Messages
- TCP-IP
- HW 4 Demo
- A Layer 6 Client and Server (You can build your Limited HTTP Server on the Layer 6 Server) Demo and Review
- HTTP Request and Response Messages in Detail

Recall: HTTP – how info moves over the web

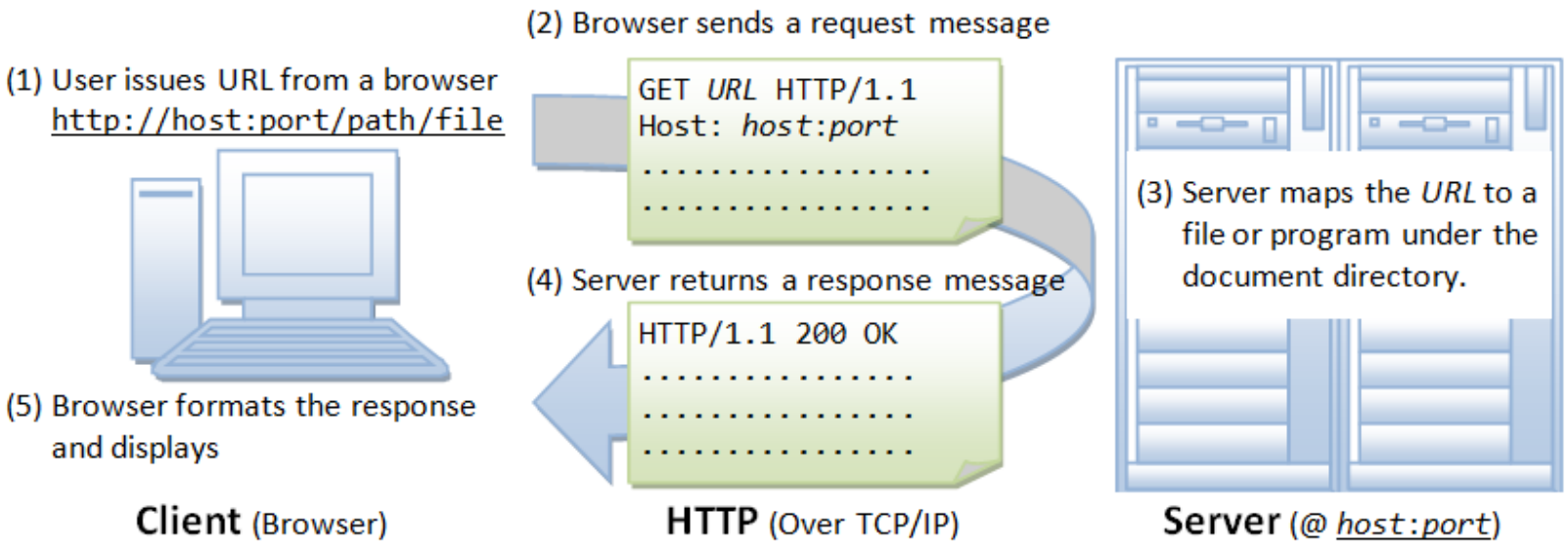
- HTTP (Hypertext Transfer Protocol) is perhaps the most popular application protocol used in the Internet (or The WEB).

- HTTP is an *asymmetric request-response client-server* protocol as illustrated.
  - An HTTP client sends a request message to an HTTP server.
  - The server, in turn, returns a response message.

- HTTP is a *pull protocol*: the client *pulls* information from the server (instead of server *pushing* information to the client).



How the HTTP Procol Works



HTTP Protocol

HyperText Transfer Protocol is a simple protocol for request/reply based interactions between a client and a server.

This serves as the foundation of the World Wide Web.

- Request messages contain a very small number methods that can be invoked on the server.
  - It is extensible so new methods can be added.
  - Parameter data can be included in a request.
- Response messages composed and sent by an HTTP server to a client indicate the status of the request processing and possibly some response data.

HTTP Protocol Continued

- When the previous request message reaches the server, the server can take one of these three actions:

1. The server interprets the request received, maps the request into a *file* under the server's documents directory, and returns the file requested to the client.

2. The server interprets the request received, maps the request into a *program* kept in the server, executes the program, and returns the output of the program to the client.

   (What examples have we seen of this???)

3. The request cannot be satisfied, the server returns an error message.

Example
Assume the following :

- On the host:
  - www.test101.com
- In the file:
  - /docs/index.html

```
<!DOCTYPE html>
<html>
  <body>
     <h1>It works!</h1>
  </body>
</html>
```

You type the following into your Browser's address window (and hit the enter key):

- http://www.test101.com/docs/index.html

Details of the HTTP Protocol – HTTP Request Message

- When you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server.

- For example, the browser translates the URL:
  http://www.test101.com/docs/index.html
  into the following request message:

GET /docs/index.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)

HTTP Protocol, Continued

- The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header).
- Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf".
- In its idling state, an HTTP server simply listens for IP address(es) and port(s) specified in the configuration for incoming request.
- When a request arrives, the server analyzes the message header, applies rules specified in the configuration, and takes the appropriate action.
- The server composes a response message that returns the resource (get), Posts the information to the Server, carries out another HTTP Method, or returns an error
- The webmaster's main control over the action of web server is via its configuration.

In our example, the resource is located

- And the resource has the proper permissions,
- So a “Success” response message is returned, along with the data in the resource

An HTTP response message composed in response to the original request:

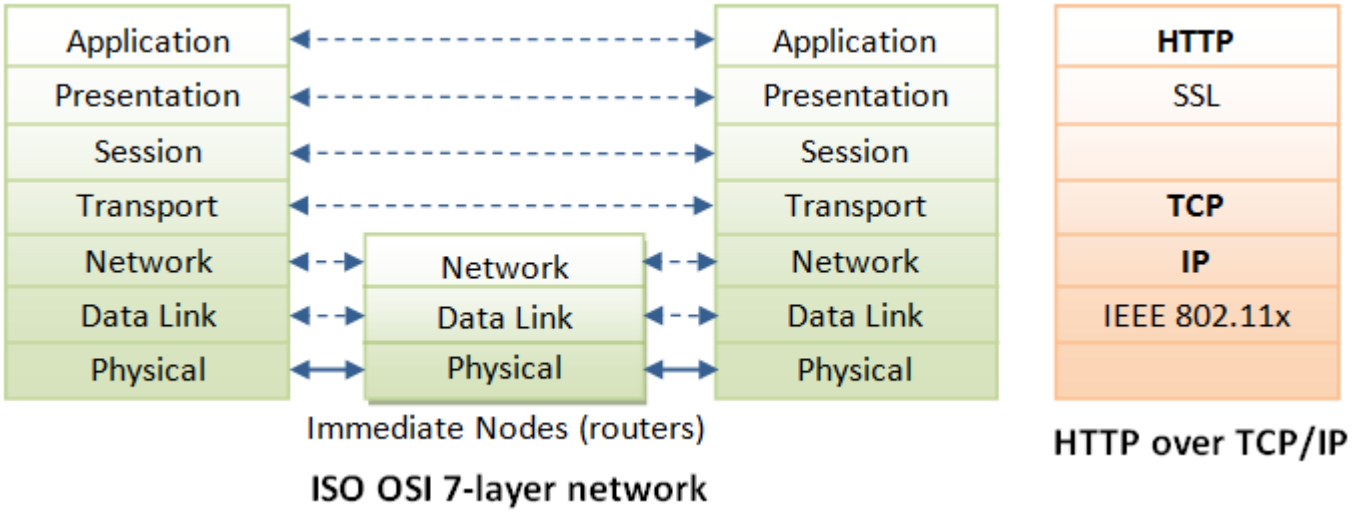
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>

Toward an HTTP Server

HTTP can be built on TCP/IP

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



- (HTTP need not run on TCP/IP. It only presumes a reliable transport. Any transport protocols that provide such guarantees can be used.)

TCP/IP

- TCP/IP (Transmission Control Protocol/Internet Protocol) is a set of transport and network-layer protocols for machines to communicate with each other over the network.

- TCP (Transmission Control Protocol) is a transport-layer protocol, responsible for establishing a connection between two machines.

- TCP consists of 2 protocols: TCP and UDP (User Datagram Package).

- TCP is *reliable*: each packet has a sequence number, and an acknowledgement is expected. A packet will be re-transmitted if it is not received by the receiver. Packet delivery is guaranteed in TCP.

- UDP does not guarantee packet delivery => not reliable.
  - But, UDP has less network overhead and can be used for applications such as video and audio streaming, when reliability is not as critical.

What does this say?

- Teh quik brwon fox jumes over teh lzay dog

- Similarly, you will still get the meaning of an audio or video transmission if blips like this occur during the transmission

- You will notice the lack of quality in the transmission however

TCP / **IP** Continued

- IP (Internet Protocol) is a network-layer protocol (network addressing and routing).
- In an IP network, each machine is assigned an unique IP address (e.g., 165.1.2.3), and the IP software is responsible for routing a message from the source IP to the destination IP.
- In IPv4 (IP version 4), the IP address consists of 4 bytes, each ranges from 0 to 255, separated by dots, which is called a *quad-dotted form*. This numbering scheme supports up to 4G addresses on the network. IPv6 supports more addresses (16 bytes worth)
- Since memorizing a 12 digit number is difficult for most of the people, an english-like domain name, such as www.test101.com is used instead.
- The DNS (Domain Name Service) translates the domain name into the IP address (via distributed lookup tables).
- A special IP address 127.0.0.1 always refers to your own machine. It's domain name is "localhost" and can be used for *local loopback testing*

TCP / IP Continued

- TCP *multiplexes* applications within an IP machine.
- For each IP machine, TCP supports (multiplexes) up to 65536 ports (or sockets), from port number 0 to 65535.
- An application, such as HTTP or FTP, runs (or listens) at a particular port number for incoming requests.
- Port 0 to 1023 are pre-assigned to popular protocols:
  - HTTP at 80,
  - FTP at 21,
  - Telnet at 23,
  - SMTP (Simple Mail Transfer Protocol) at 25,
  - NNTP (Nework News Transfer Protocol) at 119,
  - and DNS at 53.
  - Port 1024 and above are available to the users.

TCP / IP Final Thoughts / Summary

- TCP port 80 is pre-assigned to HTTP, as the default HTTP port number.
- This does not prohibit you from running an HTTP server at other user-assigned port number (1024-65535) such as 8000, 8080 (e.g., when testing a new server).
- You can also run multiple HTTP servers in the same machine on different port numbers.
- When a client issues a URL without explicitly stating the port number, the browser will connect to the default port number 80 of the host.
- The request from the client needs to explicitly specify the port number in the URL if they want to request something from a server that is not listening to the default url.

  e.g. http://www.test101.com:8000/docs/index.html will request

  docs/index.html

  from the server listening at port 8000

- **SUMMARY:**

  To communicate over TCP/IP, you need to know:

  (a) IP address or hostname, (b) Port number.

HW 4 Demo – A Tiny HTTP server

HW Assignment 4

- Build a simple HTTP server in python
- I'll go over Python 3 code that essentially provides the transport layer (Layer 6) in the 7 layer ISO protocol and post it to the class Moodle page, along with these lecture slides
- You can build your limited HTTP server for HW4 (Level 7 Application Layer Protocol) on top of that (You will have to refactor it, of course)

Idea

- YOU 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)

Lets have a look at Echo Client and Echo Server – in Python!

- ISO-OSI Level 6 Programs – socket level
- ***HTTP is an ISO-OSI Level 7 protocol –that is an Application-level protocol***
- It is built on level 6 (and below) applications
- You are free to use this code to construct the Web Server you are building for HW 4

Demo

Layer 6 (sockets) Echo Client & Server Code Review

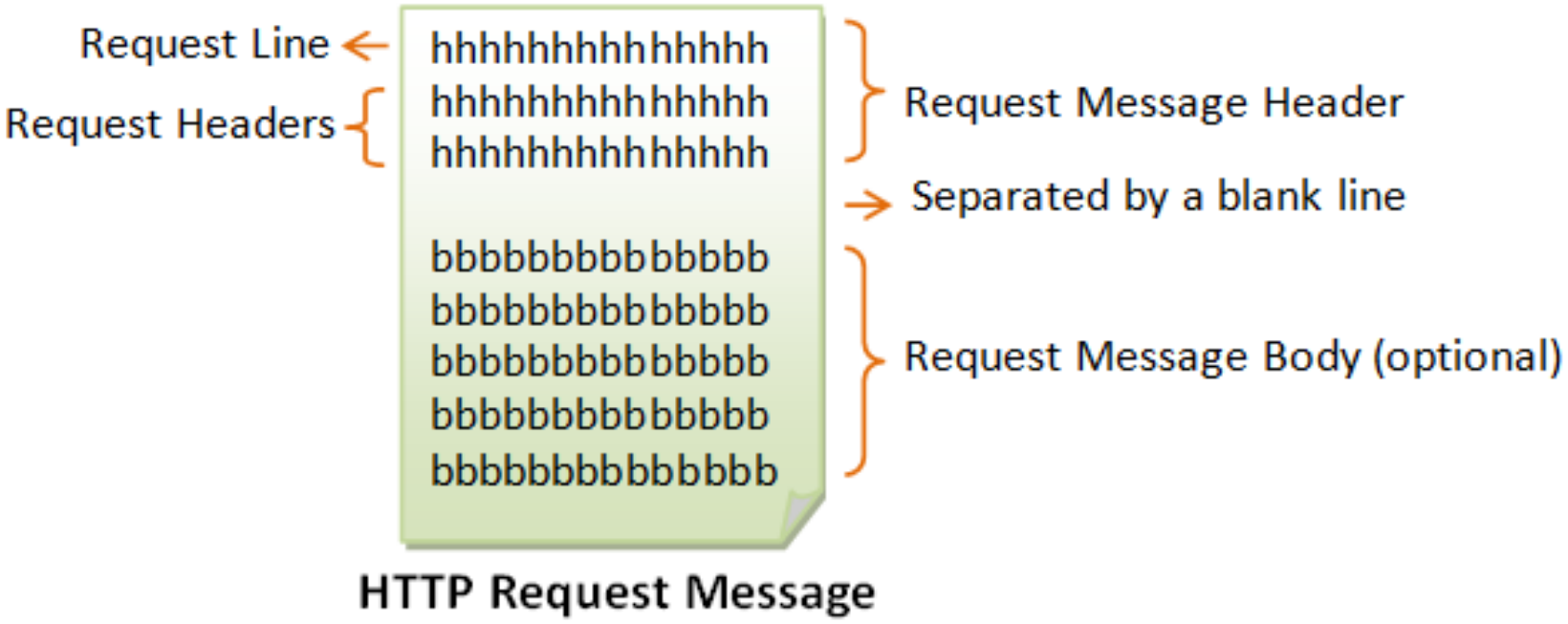
EchoClient.py and Echo Server are Available on Moodle in the Lecture 5 Materials, You can pull them down now and review them as I go through them on my machine and the Board

Details, Details – You need to understand them to build your Server

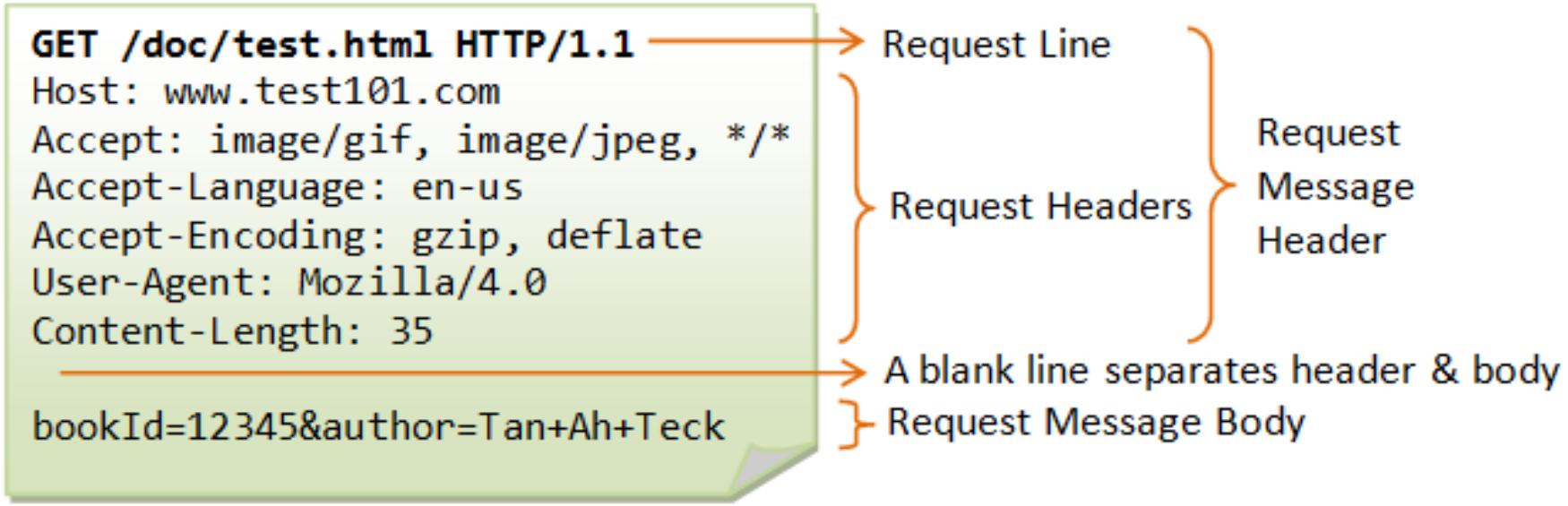
- http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html
- http://www.w3c.org/Protocols/

- Do at least the first reading,
- Refer to the second official Reference for clarification on the details of HTTP 1.1

Format of HTTP Request Message



HTTP Request Message Example



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.

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?

Question

- Can you give me example(s) of programs that can be used to construct and send - or just 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 (can't backspace)

OR Can use a Network Program

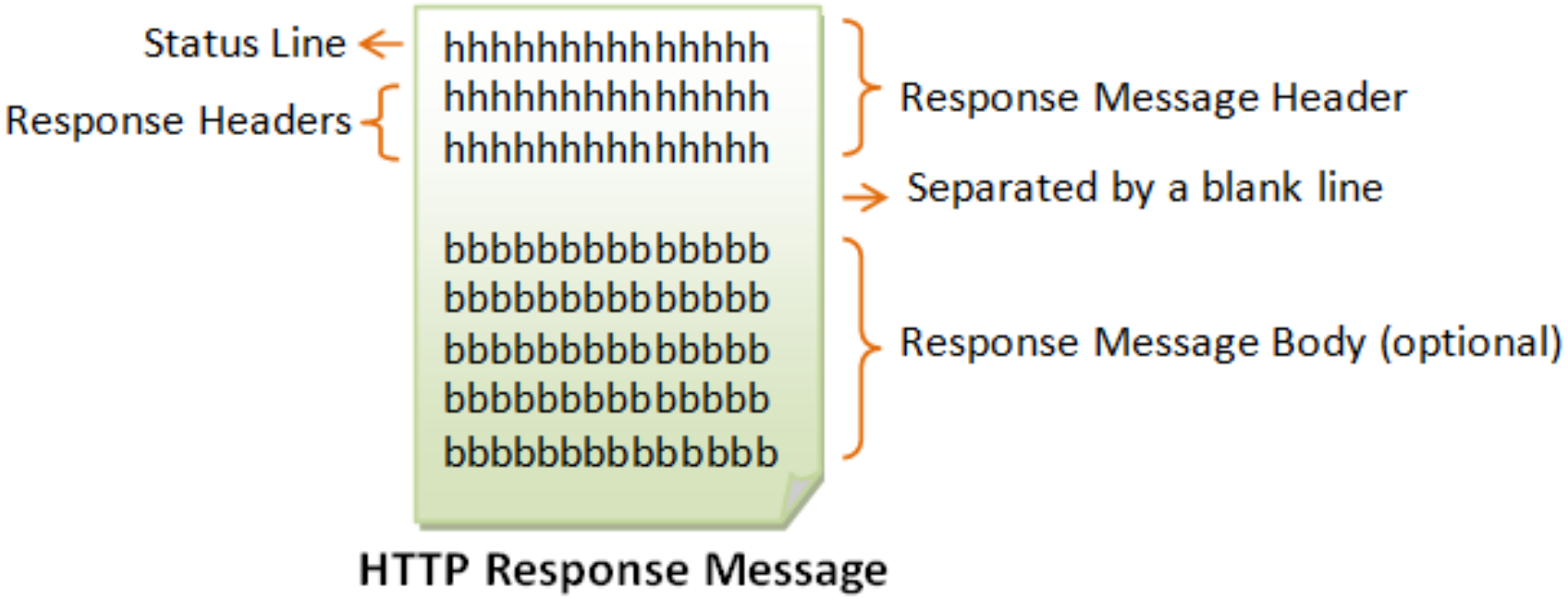
```
import java.net.*;
import java.io.*;

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

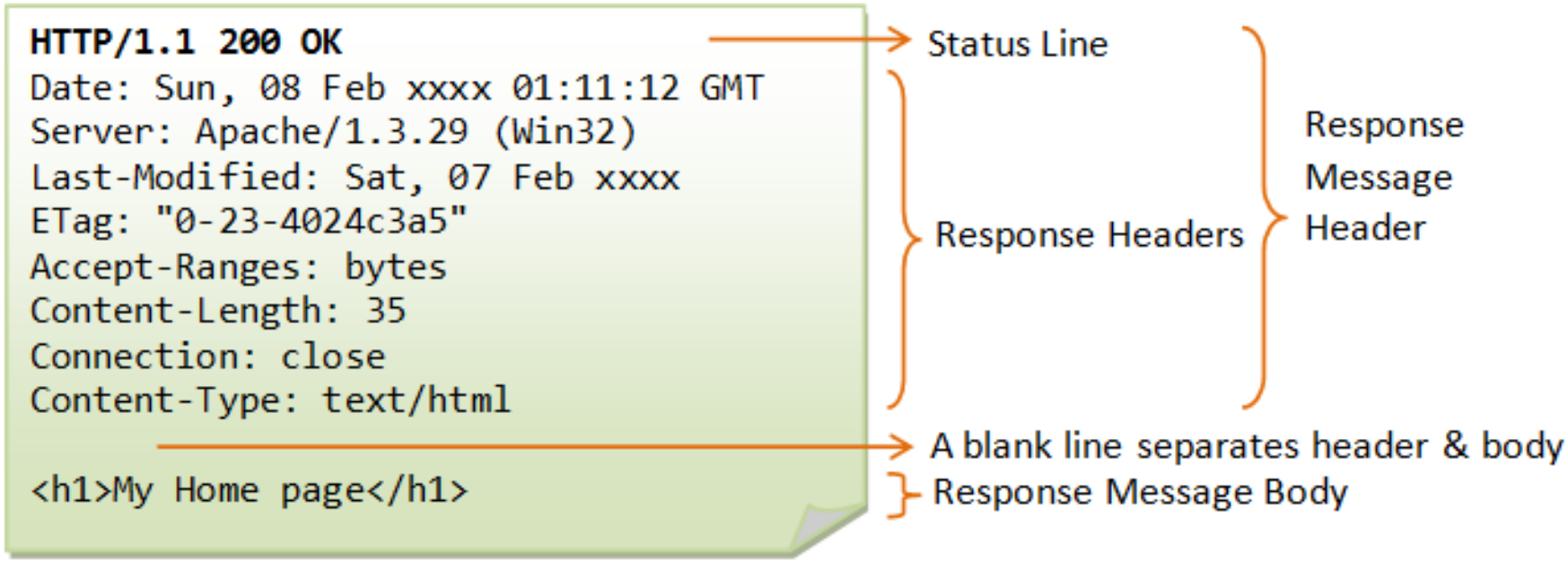
- Unix/Linux **cURL** command, which supports HTTP scripting
- Or Unix / Linux **wget** command
- Or can use Chrome POSTMAN application – need to add to your browser on the CSE Labs machines, but chrome apps are going away soon
- Or you can use POSTMAN application on your own machines.
- BUT, you must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

Or, any Web Browser (construct and send)!

HTTP Response Message (From the Server)



Example Response Message (in response to a GET request)



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.

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 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'

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

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

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

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

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…

Next Time

- HTTP and Assignment 4 Revisited and Wrap-Up (Regular Expressions Revisited…)
- Intro to Node.js
- Review for Midterm