Experiential Learning Workshop on Basics of HTTP

July 04, 2018

Dr. Ram P Rustagi
Professor, CSE Dept
KSIT, Bangalore
rprustagi@ksit.edu.in

Resources & Acknowledgements

- Resources
  - https://rprustagi.com/ELNT/Experiential-Learning.html
    - Articles in ACCS Journal https://acc.digital
    - www.github.com/rprustagi
  - https://www.rprustagi.com/workshops/ieee/smvdu
    - Slides
  - https://www.rprustagi.com/workshops/programs
    - Example web pages, and programs
- Acknowledgements:
  - Computer Networking: Kurose, Ross

Day 1: Basics of Networking

- Overview
- Introduction to basic networking Tools
- *Handson 1: using networking tools*
- IP and TCP Headers
- Analysis of layers in IP, TCP/UDP
- *Handson-2: Analyze IP and TCP headers*
- Fragementation and PMTU Discovery
- ICMP Errors, NAT
- *Handson-3: ICMP errors, NAT, PMTU*
- ARP, DHCP, Proxy, Gratuituous ARP
- *Handson-4: ARP protocol*
- Summary

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

Day 3: Basics of Transport Layer

- Overview: Transport layer, requirements
- Connection less and connection oriented transport
- *Handson-1: Analyze TCP 4-tuple and UDP 2-tuple*
- Pseudo headers in TCP/UDP
- Concurrent communications : UDP and TCP
- *Handson-2: Using data with same checksum*
- TCP and UDP Error control, TCP flags
- *Handson-3: Connection Mgmt, Queues, and states*
- TCP Streaming, Reliability misnomer,
- UDP message boundaries
- Handson-4: TCP Streams and UDP messages
- Summary

Day 4: Basics of Web Security

- Overview: HTTPS protocol
- Server certificate and server authentication
- Mixed content and browser warnings
- Locks icons and HTTP Status
- Handson-1: HTTPS website with mixed content
- MITM attack and ARP spoofing
- MITM with browser and information stealing
- Understanding HSTS, CSP
- Handson-2: Implementing ARP Spoofing
- Summary

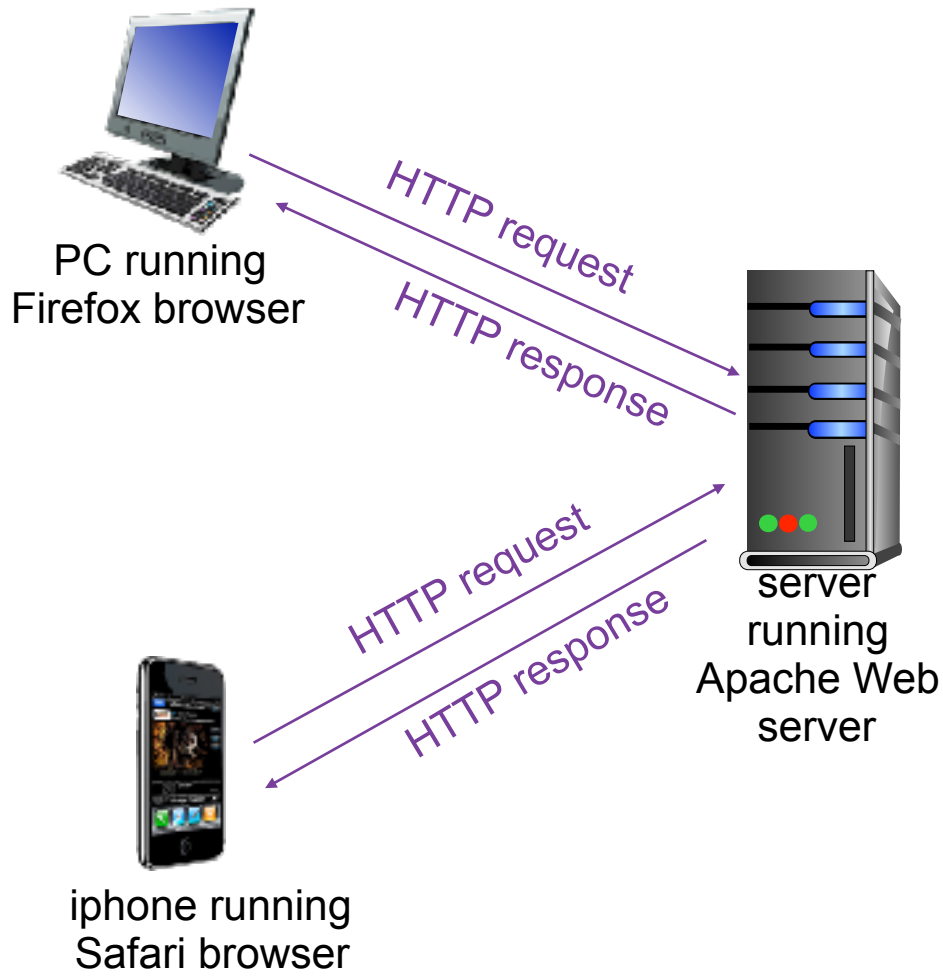
Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

HTTP overview

HTTP: hypertext transfer protocol

- Application layer protocol
- Client/server model
  - *client:* browser requests, receives, and "renders" Web objects
  - *server:* sends objects in response to requests
- **Stateless protocol**
- Uses underlying TCP protocol



Source: Kurose, Ross: Computer Networking, A Top Down Approach

HTTP Protocol

- First interaction/implementation
  - A subset of intended protocol
  - (unofficially) labeled as HTTP 0.9
- HTTP 0.9
  - Client-server, request-response protocol
  - ASCII protocol, running on TCP/IP
  - Design to xfer HTML document
  - Connection is closed after each request
  - No meta data (HTTP headers)

HTTP 1.0

- Key protocol changes
  - Request has multiple header lines
  - Response is prefixed with status line
  - Response has its own header lines
  - Response can be non-HTML
    - A plain text file, image, other contents
  - TCP connection closed after response served
  - Other supports
    - Content encoding, character set, multi-part
    - Authentication, caching, proxy behaviours,
    - Date formats …

HTTP 1.1

- RFC 2068 - First official standard (Jan 1977)
- RFC 2616 - Current standard (June 1999)
- A lot of performance optimizations
  - Keep alive connections
  - Chunked encoding transfers
  - Byte range requests
  - Additional caching mechanims
  - Request pipelines
  - Language negotiations
  - Caching directives

HTTP/2

- Goals:
  - Impove transport performance
  - Lower latency and higher thruput
  - No changes in high level semantics
    - All headers, values, use cases are same
  - Any existing HTTP application should work without modification
  - Any server upgrades should be transparent to majority of users

HTTP/2

- Goal: reduce latency
  - Make applns faster, simpler & robust
- Mechanism
  - Undo workarounds of HTTP 1.1
    - Make protocol less sensitive to RTT
  - Enable request/response multiplexing
  - Minimize protocol overhead
  - Enable header compressions
  - Request prioritization
  - Server push

HTTP/2

- What does not change from HTTP 1.1
  - No semantics changes to HTTP
  - All core concepts remains the same
    - HTTP methods, Status codes
    - URIs, Header fields
- What is changed
  - How data is formatted (framed)
  - How data is transported
  - Hides complexity from application
    - With new framing layer

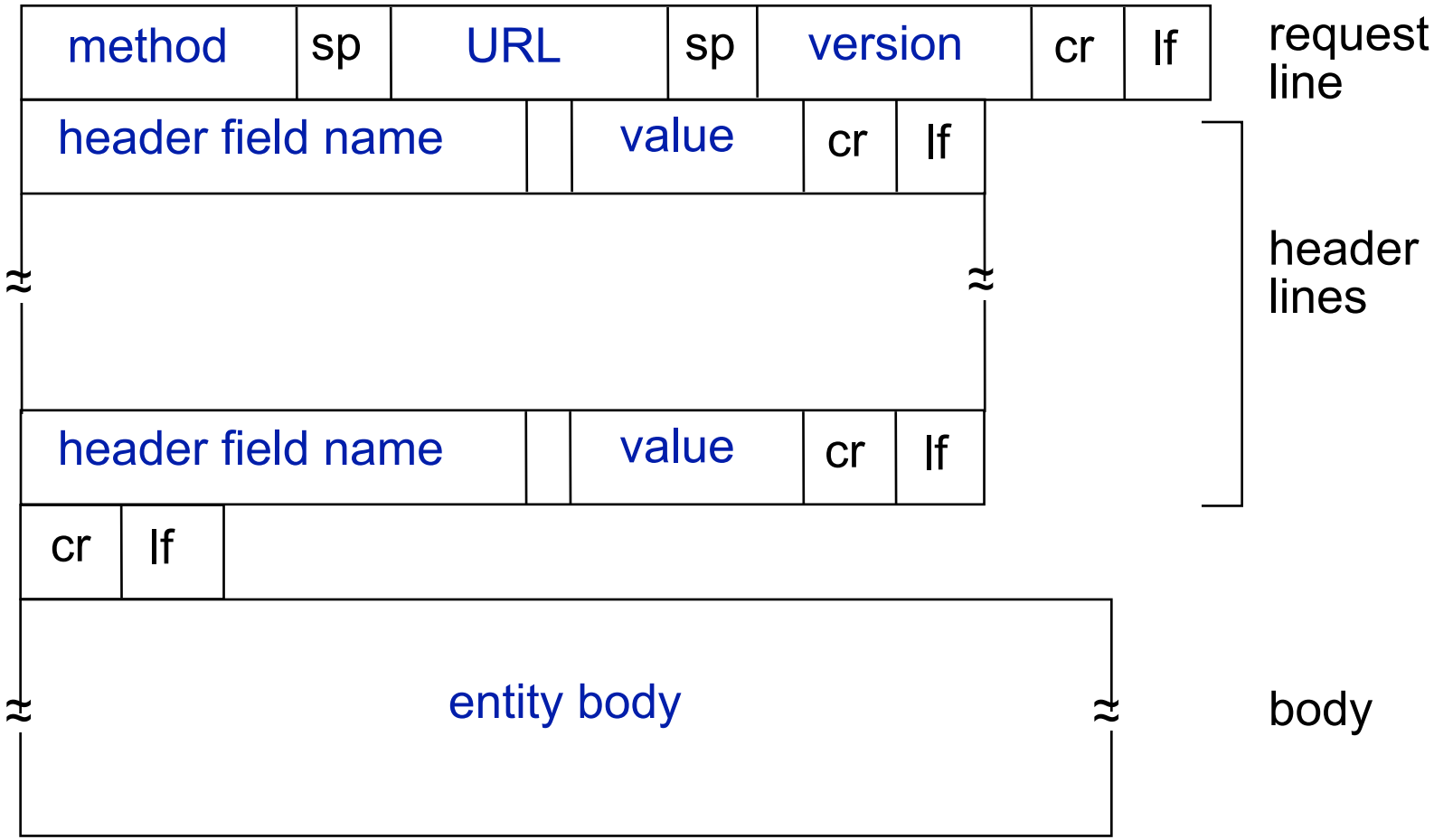
Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

HTTP Messages

- Two types
  - Request Message
  - Response Message
- Data is in clear text
  - Readable by humans
- Structure
  - Message line
  - Header lines
  - Empty lines
  - Data

HTTP request message: general format



Source: Kurose, Ross: Computer Networking, A Top Down Approach

HTTP request message

- Two types of HTTP messages: ***request, response***
  - ASCII (human-readable format)

carriage return character

line-feed character

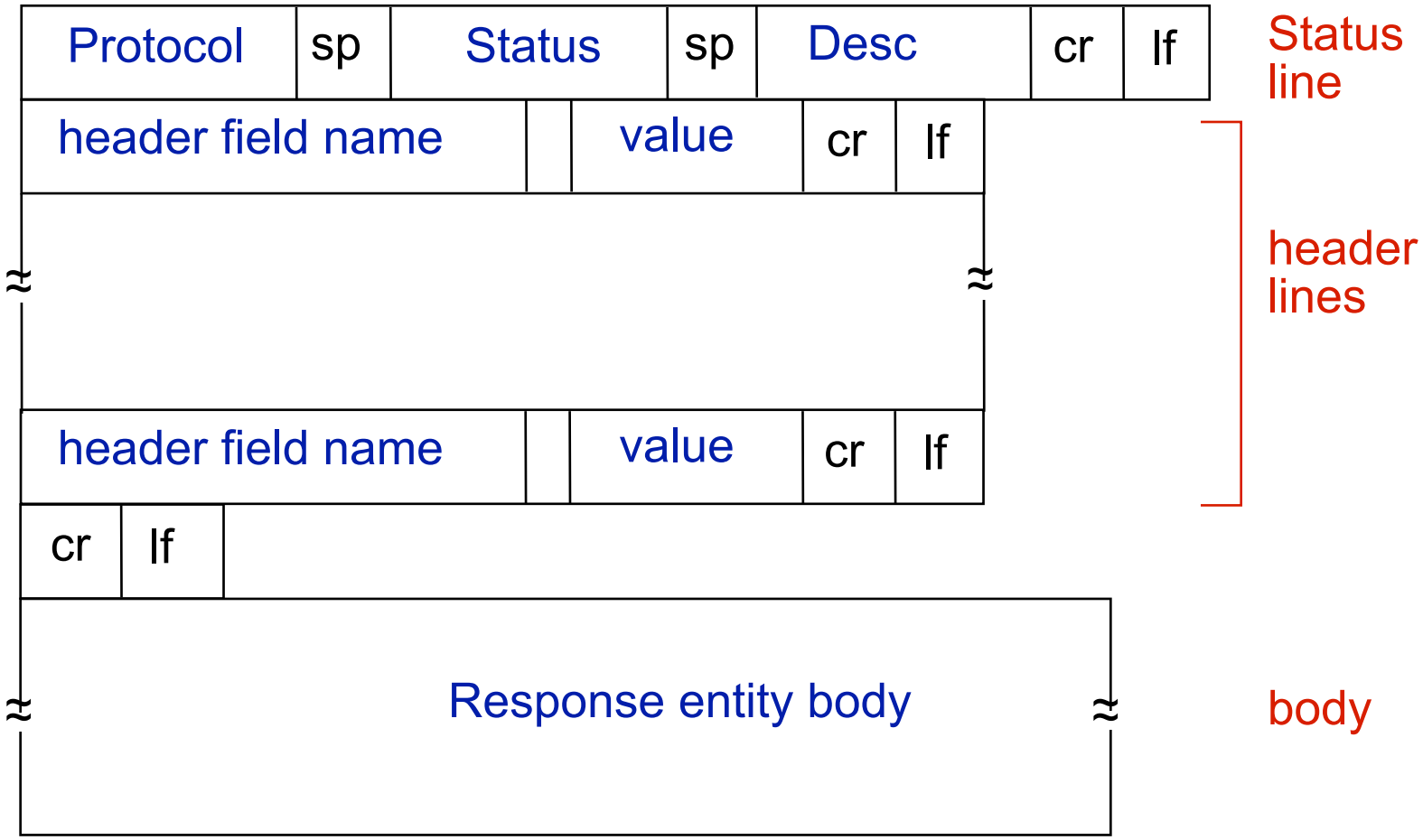
request line (GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /Workshop/IEEE/hello.html HTTP/1.1\r\n
Host: 10.1.12.2\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
If-Modified-Since: Tue, 26 Jul 2016 05:47:12 GMT
\r\n
```

HTTP reponse message: general format



Source: Kurose, Ross: Computer Networking, A Top Down Approach

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Tue, 26 Jul 2016 08:33:58 GMT\r\n
Server: Apache/2.4.7 (Ubuntu)\r\n
Last-Modified: Tue, 26 Jul 2016 05:47:12 GMT\r\n
ETag: "a5-538836eb6aa69-gzip"\r\n
Accept-Ranges: bytes\r\n
Content-Encoding: gzip\r\n
Content-Length: 132\r\n
Keep-Alive: timeout=5, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html\r\n
\r\n
data data data data data ...
```

MIME Types

- Originally for email
- Specifies the form of content served
- Type specifications
  - Examples
    - `text/plain,text/html,image/gif,`
    - `audio/mpeg, video/quicktime,`
    - `application/msword`
- Server gets type from file name suffix
- Experimental types
    - `video/x-msvideo`
    - Server sends helper application

HTTP response status codes

- `1xx` - Informational
  - Request received, continuing process
- `2xx` - Success
  - Action successful, understood and accepted
- `3xx` - Redirection
  - Further action must be taken to complete 4xx
- `4xx` - Client Error
  - Request contains bad syntax or cannot be filled
- `5xx` - Server Error
  - Server failed to fulfil an apparently valid request

HTTP response status codes

200 OK

- req succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request (example ?)

- request msg not understood by server

404 Not Found (example ?)

- requested document not found on this server

500 Internal Server Error

- http://<host>/cgi/badcgi.sh

505 HTTP Version Not Supported

HTTP Headers

- `Accept-Language: en-US`
  - Determines your preference
- `Accept-Encoding: gzip, deflate`
  - Determines compressed download
- `User-Agent:`
  - Determines client type
  - Mobile, web, tablet

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

Hands-On 1

- Resource:
  - https://acc.digital/experiential-learning/
- Installing apache
  - `sudo apt install apache2`

Hands-On 1

- Resource:
  - https://acc.digital/experiential-learning/
- Status code 200
  - `Content-Type: text/html, text/plain`
  - `Content-Type: image/jpg, text/plain`
  - `Accept-Language: hn-IN, en-US`
- Status code 404 Not Found
- Status code 403 Forbidden
- Status code 400 Bad Request
- Status code 301/302 Found
  - (Header `Location:`)
  - `wget -d` http://google.com

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

HTTP connections

- *Non-persistent HTTP*
  - At most one object sent over TCP connection
    - Connection then closed
  - Multiple objects requires multiple connections
- *Persistent HTTP*
  - Multiple objects can be sent over single TCP connection between client, server
- Question?
  - Explain in non-technical context
  - E.g. Using radio-taxi

Non-persistent HTTP

suppose user enters URL:

`myweb.com/mypage.html`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

2. HTTP server at host www.someSchool.edu waiting for TCP conn at port 80. "accepts" conn, notifies client

3. HTTP client sends HTTP *req msg* (containing URL) into TCP conn socket. Msg indicates that client wants object someDet/home.index

4. HTTP server receives req msg, forms *resp msg* containing requested object, and sends message into its socket

Source: Kurose, Ross: Computer Networking, A Top Down Approach

time

Non-persistent HTTP (cont.)

5. HTTP server closes TCP connection.

6. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

Repeat steps 1-6 for each of 10 jpeg objects

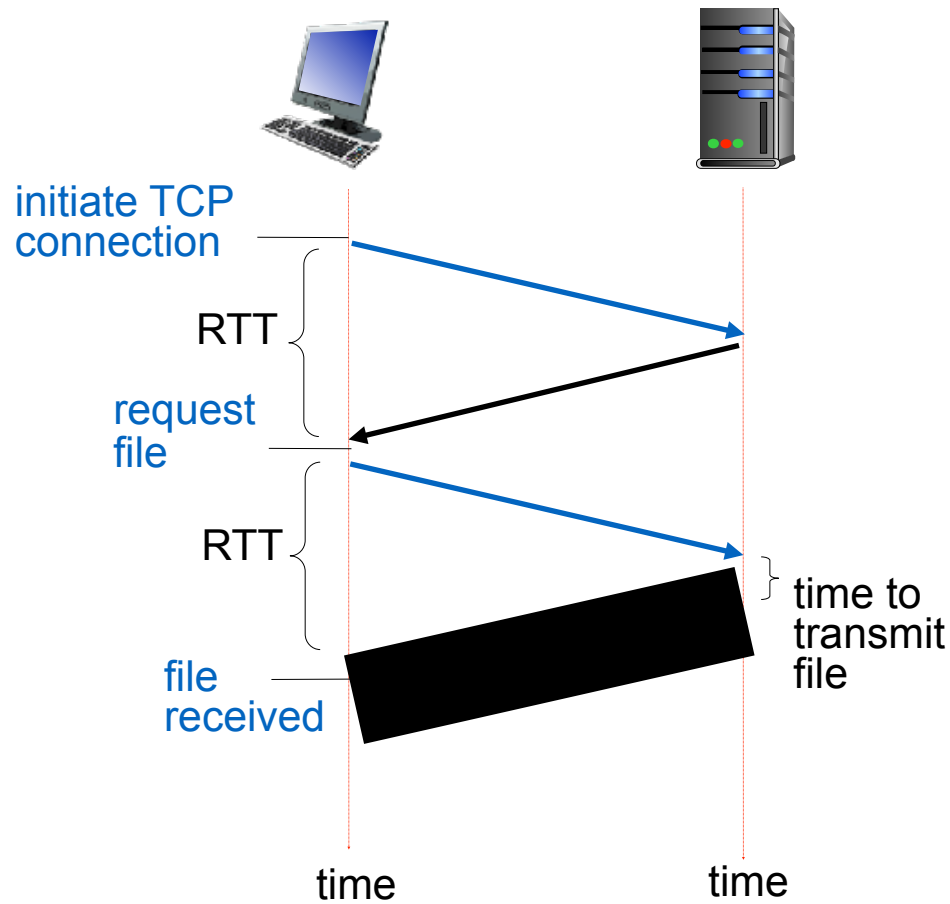
Source: Kurose, Ross: Computer Networking, A Top Down Approach

Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

- 1 RTT to setup TCP
- 1 RTT for HTTP request and first few bytes of HTTP response to return + file transmission time
- non-persistent response time = 2RTT+ file xmit time



Persistent HTTP

non-persistent HTTP

issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

Q: Persistent vs Non-Persistent HTTP

- A web page consists of 10 embedded objects.
- Consider browser uses 3 parallel connections.
- Consider that RTT time is 1 second
- Assume that transmission time is zero and display time by the browser after receiving contents is also zero.
- Find out the time taken to display this web page, when

★Browser uses non-persistent HTTP connections?

★Browser uses persistent HTTP Connections?

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

Persistent Connections

- Apache Config

```
KeepAlive On
MaxKeepAliveRequests 100
KeepAliveTimeout 50
```

- Browser (firefox) config
  - URL "about:config"
  - change the value of (default 6)
    - network.http.max-persistent-connections-per-server
- In the browser (firefox) use the URL
  - pictures.html
  - Analyze the capture in wireshark

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

Hands-On 2

- Persistent and Non-persistent connections.
  - Create a web page with embedded images
    - e.g. rprustagi.com/workshops/web/pictures.html
  - Configure Apache with keepalive: off
    - Access webpage and analyze wireshark capture
  - Configure Apache with keepalive on
    - Configure in Firefox max concurrent connections

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

Overview

- Basics of Cache
- Conditional-get
- Web cache example
- Type of cache
- Benefits of cache
- Exercises

Web Cache and Proxy

- A Good resource on Web Cache
  - https://www.mnot.net/cache_docs/
  - https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching?hl=en
- Resources and acknowledgements
  - http://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198700.cw/index.html

Conditional GET

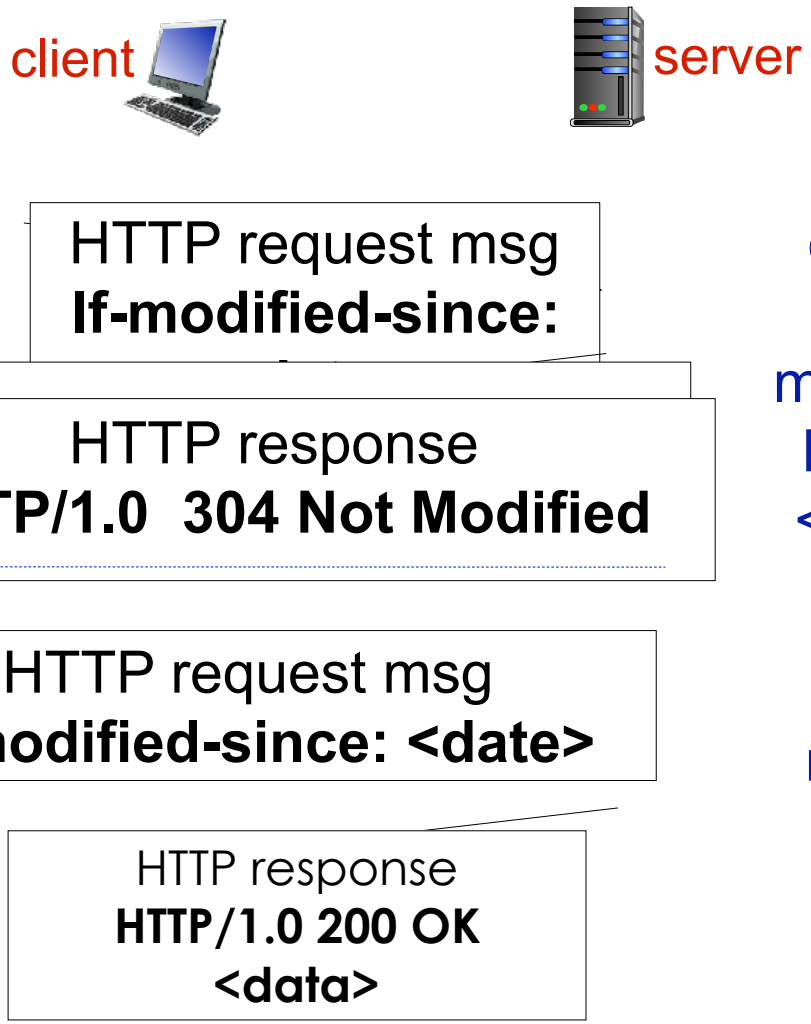
- *Goal:* don't send object if cache has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *cache:* specify date of cached copy in HTTP request

  `If-modified-since: <date>`

  - *server:* response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`

Conditional GET



Source: Kurose, Ross: Computer Networking, A Top Down Approach

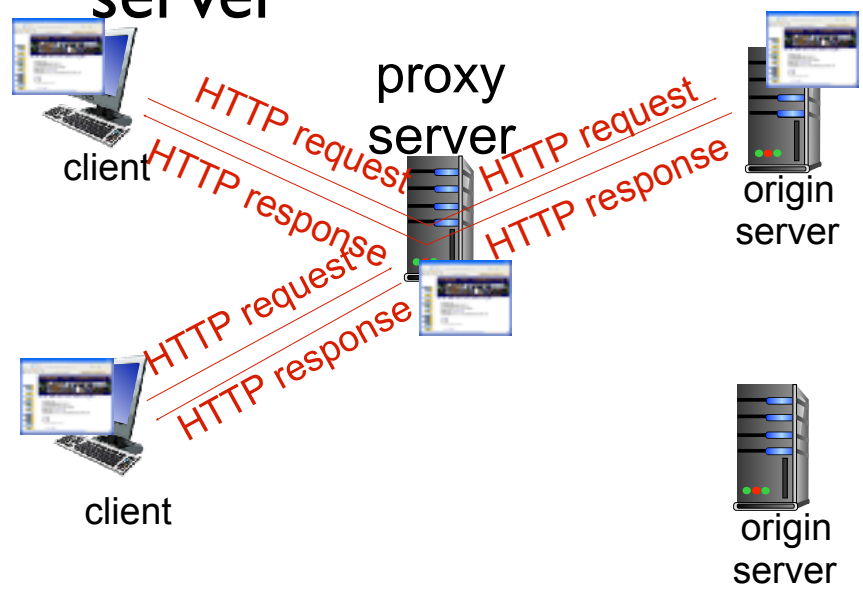
Web caches (proxy server)

- user sets browser: Web accesses via proxy server
- browser sends all HTTP requests to cache

- cache requests object from origin server, then returns object to client

- object in cache: cache returns object

goal: satisfy client request without involving origin server



Source: Kurose, Ross: Computer Networking, A Top Down Approach

More about Web caching

- Proxy server acts as both client and server
  - server for original requesting client
  - client to origin server
- typically proxy server is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

Types of Cache

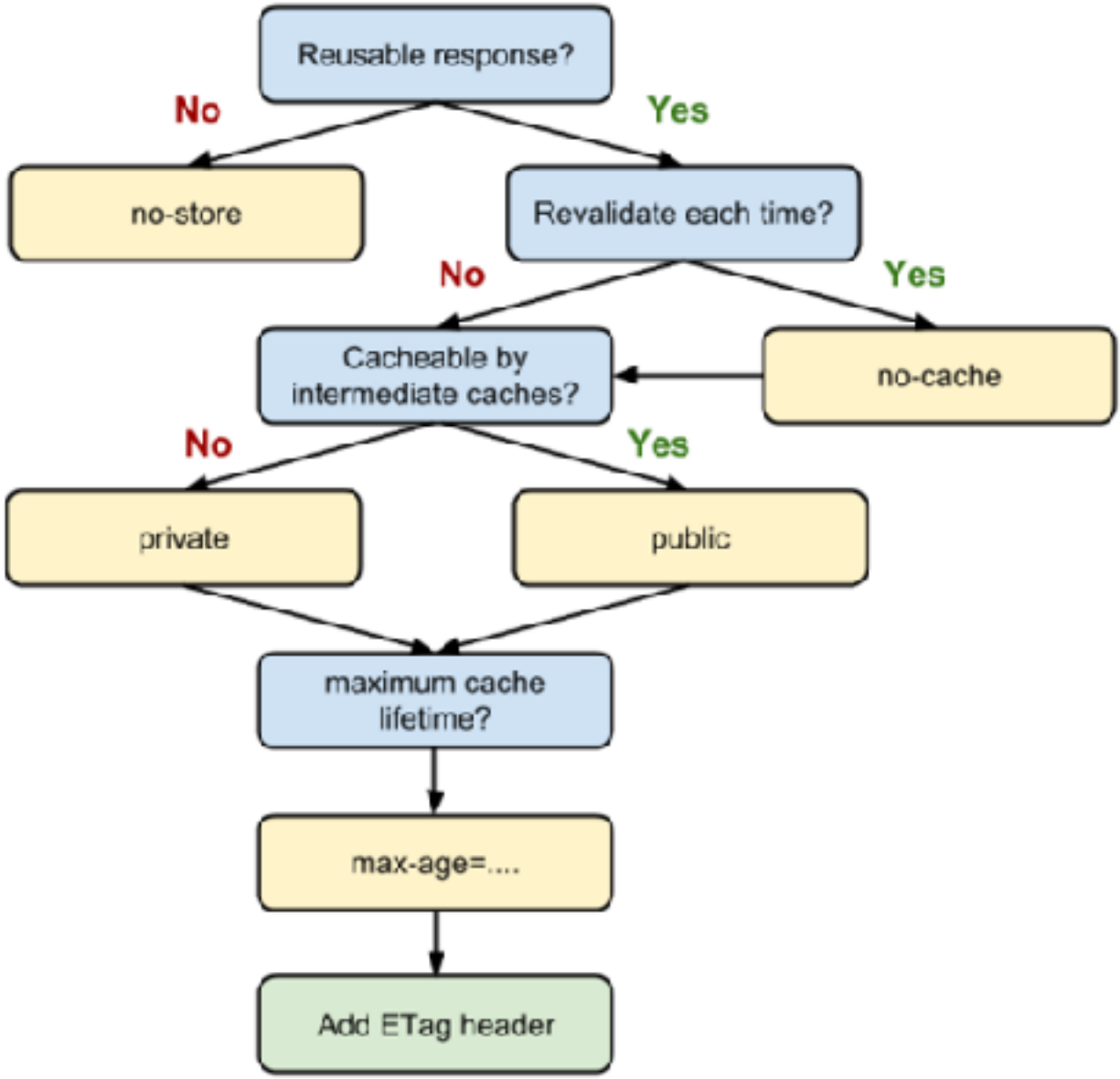
- private cache: Excluive browser cache
- public cache
  - proxy cache
    - reduces bandwidth requirements
    - reduces delays
  - gateway cache : (aka reverse proxy cache)
    - deployed by web masters for scalability
    - examples: memcached, varnish

Cache Headers

- Last-modified
- If-modified-since / If-unmodified-since
- Etag
- If-none-match
- Vary
- Age
- Pragma directive
- Date
- Expires
- Cache-Control

Cache-control

src: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching?hl=en



Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cacheing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

Hands-On 3

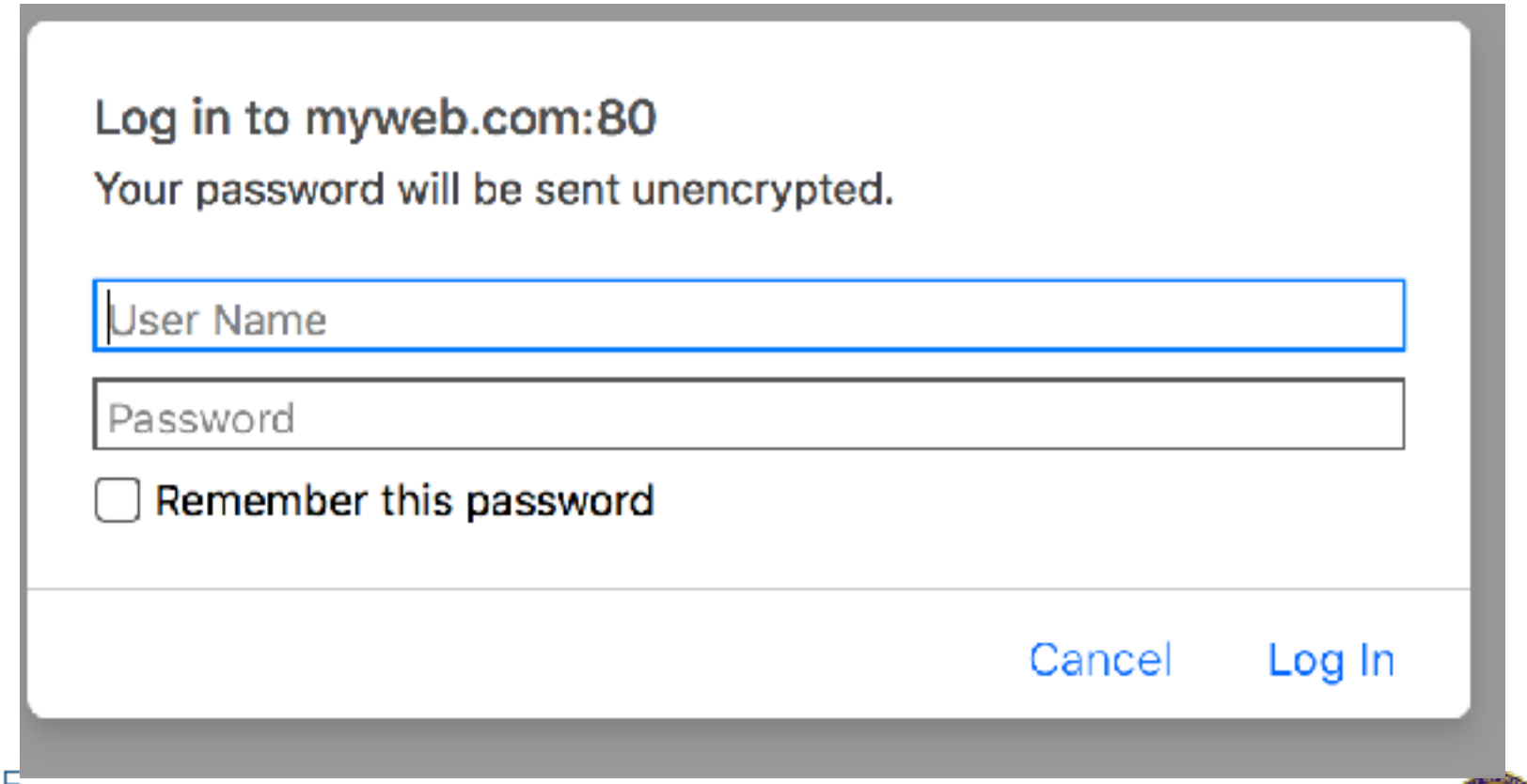
- Cache-Control
  - Access an image directly analyze the response
  - Analyze the header: If-Modified-Since
  - Update the date/time of image and re-access
  - Use nc to pass the headers.
    - If-Modified-Since and E-Tags
    - update date/time and send Etags as well
    - Analyze the response
  - Use a PHP program to define max-age
    - Access the webpage before and expiry of age

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

HTTP Authentication

- Access /private/abcd.html
  - Should see the response as below
  - Enter the username/pass



Apache Config - Authentication

- Authorization: Basic
  - Uses Base64 encoding
- Apache configuration

```
<Directory /var/www/html/private>
 AuthType Basic
 AuthName "For HTTP Learning"
 AuthBasicProvider file
 AuthUserFile /etc/apache2/passwdfile
 Require user student
</Directory>
```

- Commands to create passwords
  - `htpasswd [-c] /etc/apache2/passwdfile`
- Restart apache

Encoding of username/password

- Uses Base64 encoding.
  - Letters:
    - ‘A-Za-z0-9+/‘ # 64 letters (6 bits)
    - ‘=‘ a filler.
    - For input data, take 6 bits at a time and use the corresponding encoding.
  - Example: ‘**bits**’ is 0x62697473, i.e.

```
01100010 01101001 01110100 01100100
```

  - First 6 bits: 011000 i.e. value 24 i.e. letter Y (0—>A)
  - Second 6 bits: 100110 i.e. value 40, letter ’o’
- Username and password are separated by ‘:’ (Colon)
- Transmitted in clear text

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

Dynamic Web

- Content is generated when URL is requested
  - It is not a static content
  - Content is produced by executing program
  - Executing program also generates the HTTP headers
  - Mechanism of external program execution by web server is defined as CGI
    - Common Gateway Interface
    - Program gets the input from web server
    - Program outputs content to web server.
  - Web server expects a proper response
    - Valid HTTP headers (syntax)
    - Proper separation of headers and content

Dynamic Web

- Invoking CGI
  - Apache default
    - `/usr/lib/cgi-bin/`
  - Apache config

```
<Directory /var/www/html/cgi>
  Options ExecCGI
  SetHandler cgi-script
</Directory>
```

  - Enabling CGI as module
    - `sudo a2enmod cgi`

Working of `cgi-bin`

- Web server executes the program referred in URL
  - Program could be written in any programming language
    - C, C++, java,perl, python, php, shell etc.
- If program crashes (exits improperly)
  - HTTP headers could be corrupted/improper
- When web servers sees inconsistency,
  - Given `500 Internal Server Error`

Example of 500 error

- Sample CGI script : `cgi-good.sh`

```
#!/bin/bash
echo "Content-Type: text/html";
echo "";
echo "<h1>Hello World!</h1>";
exit;
```

- Error noticed by web server
  - No empty line between HTTP headers and HTML content

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic web
- Handson-4: Authentication, 500 status codes
- Summary

Handson-4

- Authentication
  - Create a web page with authentication access
  - Create username password for this web page
  - Analyze the credentials exchange using Base64
- Dynamic Web
  - Create a simple working cgi-bin program
  - Create a bad cgi program which is erroneous
  - Create a cgi-bin program which crashes in the middle
  - Access these web pages and analyze 500 error

Day 2: Basics of HTTP

- Overview: HTTP and Versions
- Request and Response Format, Basic headers
- *Handson-1: Analyze HTTP headers, status codes*
- HTTP persistent and non-persistent connections
- Apache config support for persistent connections
- *Handson-2: Configuring persistent connections*
- Web caching, HTTP headers for cache control
- *Handson-3: Cachecing, E-tags*
- HTTP authentication
- Dynamic Web
- Handson-4: Authentication, 500 status codes
- Summary

Summary

- HTTP versions
- HTTP protocol : message formats
- HTTP headers
- HTTP Status codes
- Persistent and non-persistent connections
- Cacheing
- Authentication
- Dynamic web

Thank You