Lecture

The HTTP Protocol

Copyright © 1999-2021 Ellis Horowitz & Marco Papa

What Does the WWW Server Do?

- Enables browser requests
- Mainly provides
  - Support for retrieving hypertext documents
  - Manages access to the Web site
  - Provides several mechanisms for executing server-side scripts
    - Common Gateway Interface (CGI)
    - Application Programmers Interface (API)
  - produces log files and usage statistics

How Does a Web Server Communicate?

- Web browsers and servers communicate using the ***HyperText Transfer Protocol*** (HTTP)
- HTTP is a **lightweight** protocol
  - different from the ftp protocol
    - ftp sessions are long lived and there are two connections, one for control, one for data
- Current HTTP protocol is version 1.1
- W3C updates to HTTP (last update: June 2014):
  - http://www.w3.org/Protocols/
- HTTP 2.0 under the IETF httpbis Working Group
  - http://datatracker.ietf.org/wg/httpbis/charter/
- HTTP/2 Home page:
  - https://http2.github.io/

HTTP History

- The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.
- The first version of HTTP, HTTP/0.9, was a simple protocol for raw data transfer across the Internet.
- **HTTP/1.0**, is defined by RFC 1945, see
  - http://www.w3.org/Protocols/rfc1945/rfc1945
- HTTP/1.0 allows messages to be in the format of **MIME**-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics.
- **HTTP/1.1**, is defined by RFCs 7230-7237 (supersedes RFC 2616) , see
  - http://tools.ietf.org/html/
- HTTP/1.1 extends the protocol to handle:
  - the effects of hierarchical **proxies**
  - **caching**
  - the need for **persistent connections**
  - **virtual hosts**

HTTP History (cont'd)

- **HTTP/2** is being worked on by **IETF Working Group**:
  - http://tools.ietf.org/wg/httpbis/
- HTTP/2 started as a copy of **Google SPDY** ("SPeeDY").
- HTTP/2 designed to speed up websites far larger than 10 years ago, using hundreds of requests/connections.
- One major feature of HTTP/2 is **header compression**:
  - https://httpwg.org/specs/rfc7541.html
- Google has dropped SPDY from Chrome and adopted HTTP/2:
  - http://techcrunch.com/2015/02/09/google-starts-fading-out-spdy-support-in-favor-of-http2-standard/
- See also:
  - https://en.wikipedia.org/wiki/HTTP/2
- Seen RFC 7540 (HTTP/2) & 7541 (HPACK):
  - https://httpwg.org/specs/rfc7540.html
- Dozens of implementations already available, including **Apache (2.4+)**, **Apache-Tomcat (8.5+)**, **Nginx (1.9.5+)**, etc.:
  - https://github.com/http2/http2-spec/wiki/Implementations
- **HTTP/3** is already being worked on!

MIME MEDIA TYPES

- HTTP tags all data that it sends with its **MIME type**
- HTTP sends the MIME type of the file using the line **Content-Type: mime type header**
- For example, here are 2 MIME type messages

Content-type: image/jpeg

Content-length: 1598

- Some important MIME types are
  - text/plain, text/html
  - image/gif, image/jpeg
  - audio/basic, audio/wav, audio/x-pn-realaudio
  - model/vrml
  - video/mpeg, video/quicktime, video/vnd.rn-realmedia, video/x-ms-wmv
  - application/*, application-specific data that does not fall under any other MIME category, e.g. application/vnd.ms-powerpoint

Multipurpose Internet Mail Extensions

- MIME is an Internet standard for **electronic mail**
  - Traditional e-mail was limited to ASCII text, limited line length, and limited size
- MIME has extended Internet e-mail to include
  - Unlimited text line and message length
  - Messages with multiple body parts or objects enclosed
  - Messages that point to files on another server and are automatically retrievable
  - International character sets in addition to US-ASCII
  - Formatted text including multiple font styles
  - Images
  - Video clips
  - Audio messages
  - Application-specific binary data
- It was formalized in RFC 2046

Facts About MIME

- MIME converts data that uses all **eight bits** into **7-bit ASCII**, sends it, and reconverts it at the other end. See:

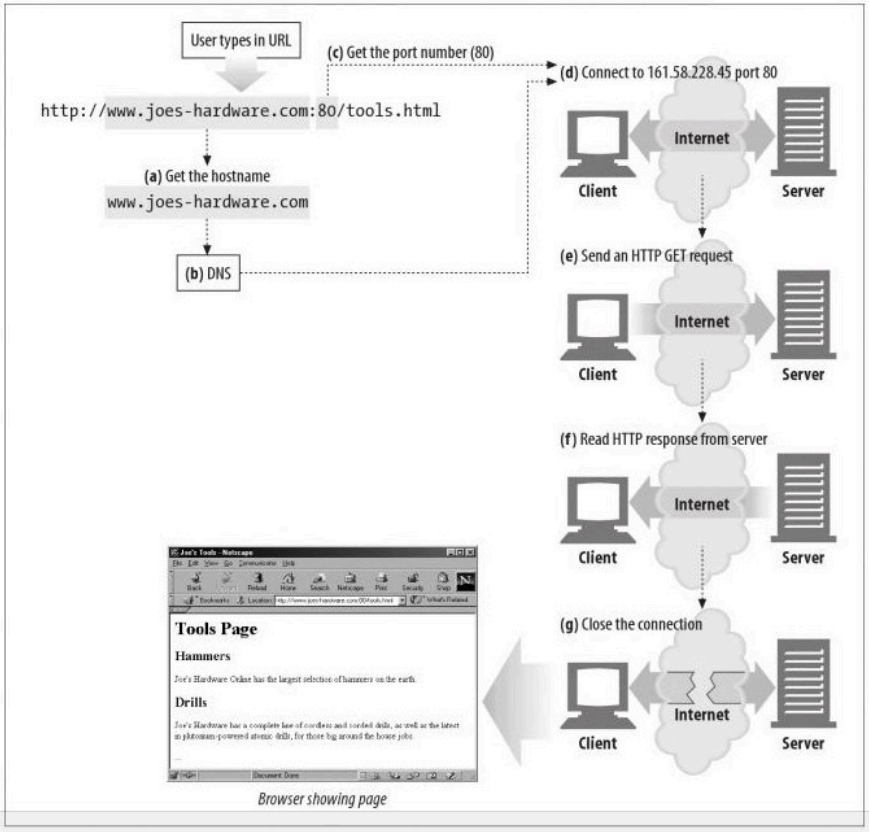
https://tools.ietf.org/html/rfc1652

- MIME headers at the front of the file define the type of data the message includes, e.g., here are a set of MIME types describing an attachment at an ftp site

```
Content-type: Message/External-Body
name="classnotes.ps"
site="ftp.usc.edu"
access-type=anon-ftp
directory="pub/cs665"
mode="image"
permission="read"
expiration="Wed, 15 Mar 2009 07:00:00 -0400 (PST)"
```

Description of a Browser Server Interaction

(a) The browser extracts the server's hostname from the URL.
(b) The browser converts the server's hostname into the server's IP address.
(c) The browser extracts the port number (if any) from the URL.
(d) The browser establishes a TCP connection with the web server.
(e) The browser sends an HTTP request message to the server.
(f) The server sends an HTTP response back to the browser.
(g) The connection is closed, and the browser displays the document.



An HTTP 1.0 "default" Scenario

- Communication takes place over a TCP/IP connection, generally on port 80

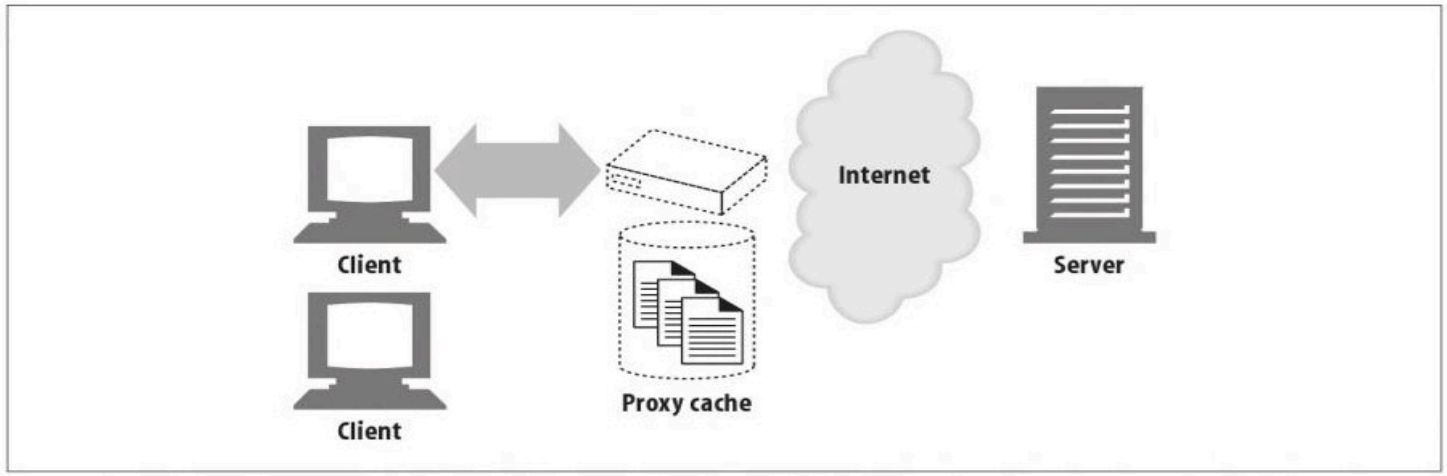
Client action	Server response
1. Client opens a connection	Server responds with an acknowledgment
2. Client sends HTTP request for HTML document	Server responds with the document and closes the connection
3. Client parses the HTML document and opens a new connection; it sends a request for an image	Server responds with the inlined image and closes the connection
4. Client opens a connection and sends another request for another image	Server sends the inlined image and closes the connection

A More Complicated HTTP Scenario

- Actually, communication between a browser and a web server can be much more complicated; communication can go between one or more **intermediaries**.
- There are three common forms of intermediary: proxy, gateway, and tunnel.
  - A ***proxy*** is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI.
  - A ***gateway*** is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server's protocol.
  - A ***tunnel*** acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

Caching Proxies

A web cache or caching proxy is a special type of HTTP proxy server that keep copies of popular documents that pass through the proxy **("forward" proxy)**. The next client requesting the same document can be served from the cache's personal copy.

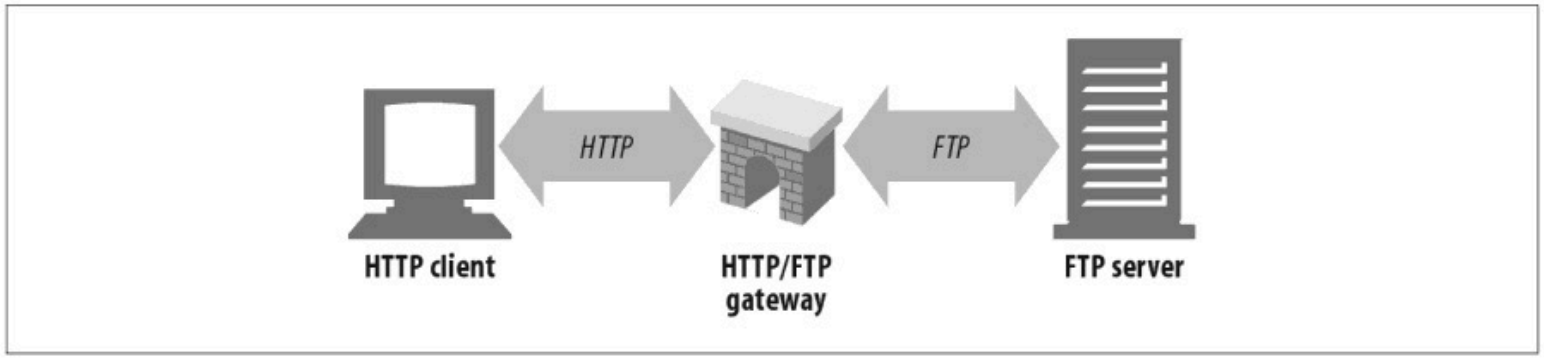


Gateways

Gateways are special servers that act as intermediaries for other servers.

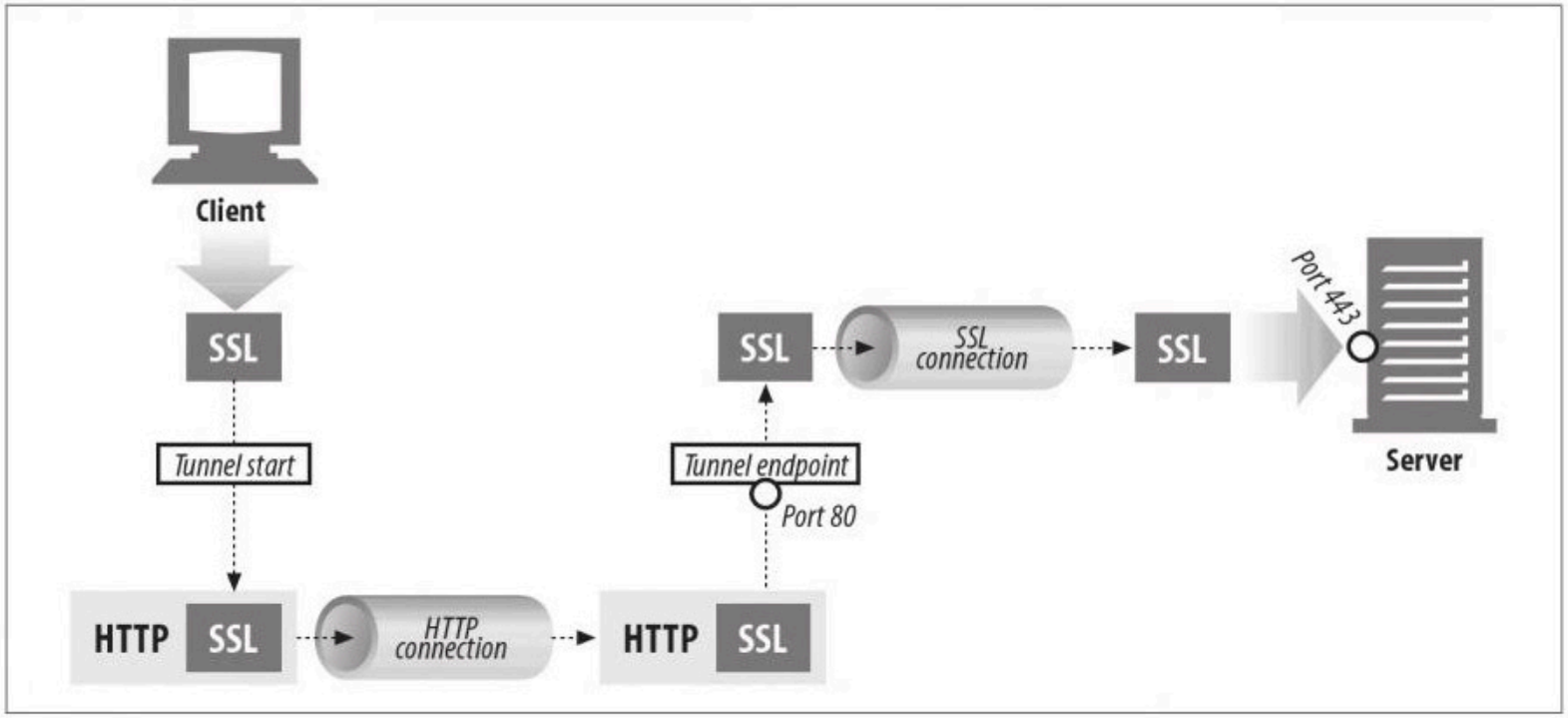
They are often used to **convert HTTP traffic to another protocol**. A gateway always receives requests as if it was the origin server for the resource. The client may not be aware it is communicating with a gateway.

For example, an HTTP/FTP gateway receives requests for FTP URIs via HTTP requests but fetches the documents using the FTP protocol. The resulting document is packed into an HTTP message and sent to the client.



Tunnels

Tunnels are HTTP applications that, after setup, blindly relay raw data between two connections. HTTP tunnels are often used to transport non-HTTP data over one or more HTTP connections, without looking at the data. A **VPN** is an example of a tunnel.



The Most General HTTP Scenario

Communication between browser and server should be regarded as a **request chain** goes left to right ----->

UA ---v--- **A** ---v--- **B** ---v--- **C** ---v--- **O**

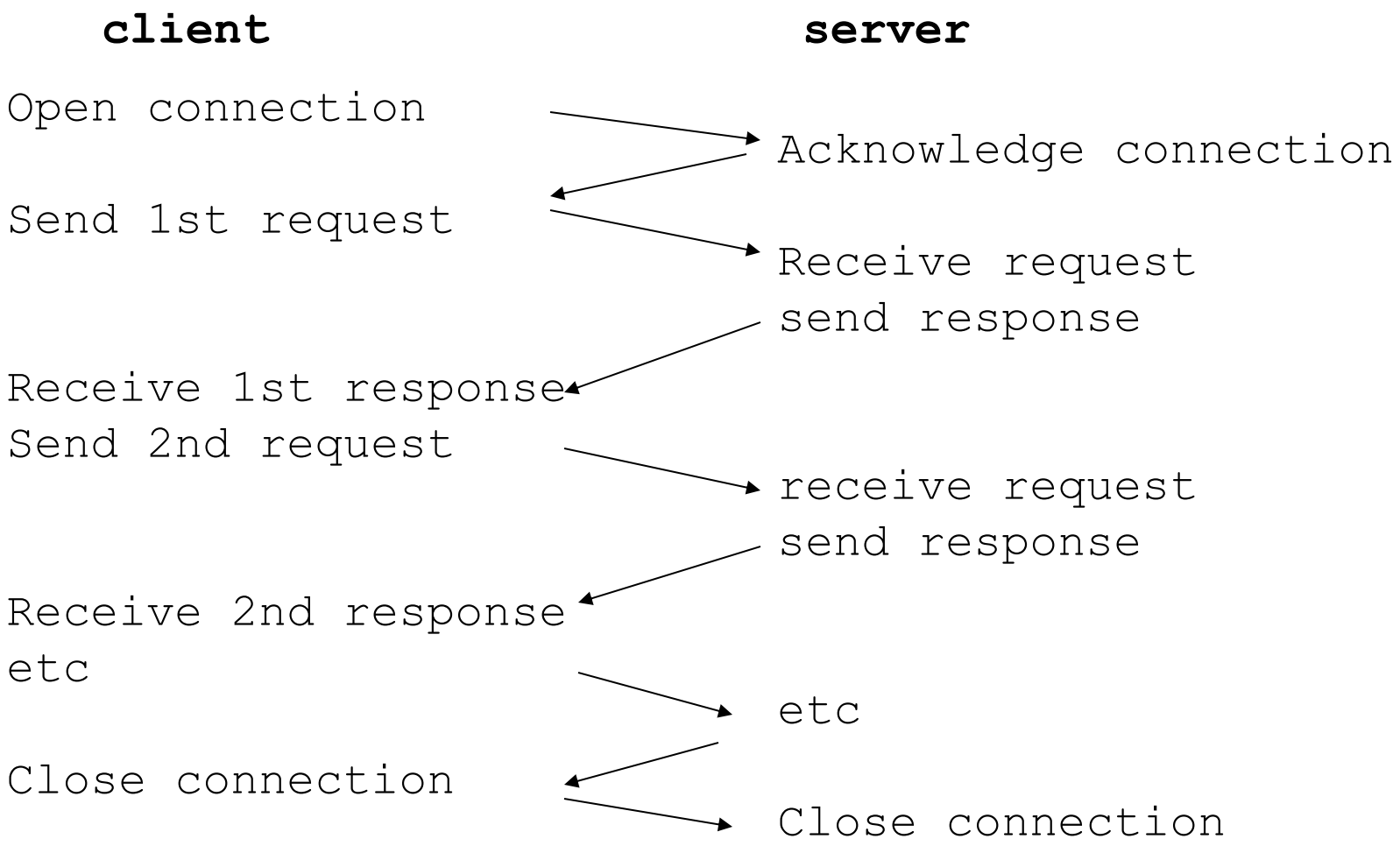
<----- and a response chain goes right to left

- **A**, **B**, and **C** are three intermediaries between the user agent and origin server. A request or response message that travels the whole chain will pass through four separate connections.
- **UA** stands for User Agent, typically a browser
- **O** stands for the origin server; the server that actually delivers the document

Persistent Connections

- In the original HTTP protocol, each request was made over a new connection
  - so, an HTML page with n distinct graphic elements produced n+1 requests
- **TCP** uses a **three-way handshake** when establishing a connection, so there is significant latency in establishing a connection
  - client sends SYN, server replies ACK/SYN, client responds with ACK
- HTTP 1.0 introduced a **keep-alive** feature
  - the connection between client and server is maintained for a period of time allowing for multiple requests and responses
  - a.k.a. **Persistent connection**

HTTP/1.0 Keep Alive Connections



HTTP/1.1 Keep Alive Extensions

- **Persistent connections** are now the **default**
- Request Header to set timeout (in sec.) and max. Number of requests, before closing:

  Keep-Alive: timeout=5, max=1000
- Client and server must explicitly say they do NOT want persistence using the header

  Connection: close
- HTTP permits multiple connections in parallel

1. client requests a page and server responds
2. client parses page and initiates 3 new connections, each requesting a different image

- Above scheme is NOT always faster, as multiple connections may compete for available bandwidth
- Generally, browsers severely limit multiple connections and servers do as well

Example of a GET Request

- Clicking on a link in a web page or entering a URL in the address field of the browser causes the browser to issue a GET request, e.g.
- Suppose the user clicks on the link below:

```
<A HREF="http://www.usc.edu/html/file.html">click here </A>
```

- The request from the client may contain the following lines

```
GET /html/file.html HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:15.0) Gecko/20100101
   Firefox/15.0.1
Referer: http://www.usc.edu/html/prevfile.html
If-Modified-Since: Wed, 11 Feb 2009 13:14:15 GMT
{there is a blank line here which terminates the input}
```

Response of the Server to GET

- In response to the previous client request, the server responds with the following

```
HTTP/1.1 200 OK
Date: Monday, 29-May-09 12:02:12 GMT
Server: Apache/2.0
MIME-version: 1.0
Content-Type: text/html
Last-modified: Sun, 28-May-09 15:36:13 GMT
Content-Length: 145
{a blank line goes here }
{the contents of file.html goes here }
```

Client HTTP Requests

- The general form of an HTTP request has four fields:

HTTP_method, identifier, HTTP_version, Body

  - **HTTP_Method** says what is to be done to the object specified in the URL; some possibilities include GET, HEAD, and POST
  - **identifier** is the URL of the resource or the body
  - **HTTP_version** is the current HTTP version, e.g. HTTP/1.1
  - **Body** is optional text

HTTP Request Methods

- Most common HTTP request methods are
  - **GET**, retrieve whatever information is identified by the request URL
  - **HEAD**, identical to GET, except the server does not return the body in the response
  - **POST**, instructs the server that the request includes a block of data in the message body, which is typically used as input to a server-side application
  - **PUT**, used to modify existing resources or create new ones, contained in the message body
  - **DELETE**, used to remove existing resources
  - **TRACE**, traces the requests in a chain of web proxy servers; used primarily for diagnostics
  - **OPTIONS**, allows requests for info about the server's capabilities

HTTP Headers

- HTTP/1.1 divides headers into four categories:
  - **general**, present in requests or responses
  - **request**, present only in requests
  - **response**, present only in responses
  - **entity**, describe the content of a body
  - extension, new headers not already defined
- Each header consists of a name followed by a colon, followed by the value of the field, e.g.

```
Date: Tue, 3 Oct 2009 02:16:03 GMT
Content-length: 12345
Content-type: image/gif
Accept: image/gif, image/jpeg, text/html
```

Examples of HTTP Headers - *Request*

- **Accept: text/html, image/***

  indicates what media types are acceptable
- **Accept-Charset: iso-8859-5**

  indicates acceptable character sets. By default all are acceptable
- **Accept-Encoding: compress, gzip**

  indicates acceptable encodings
- **Accept-Language: en, fr=0.5**

  indicates language preferences, English preferred, but French also accepted
- **Authorization:**

  used to pass user's credentials to the server

Examples of HTTP Headers - *Request*

- **From: name@site.com**

  requesting user's email address, rarely present
- **Host: www.usc.edu:8080**

  hostname and port of the requesting URL
- **Referer: http://www.usc.edu/index.html**

  the URL of the document that contains the reference to the requested URL
- **User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:18.0) Gecko/20100101 Firefox/18.0**

  reports the client software name and version and possibly platform, see

http://www.javascriptkit.com/javatutors/navigator.shtml

Byte Range Headers

- Requests
  - **If-Range: "entity-tag"**

  used with byte range requests to guarantee that any new byte range responses are generated from the same source object. The entity-tag is quoted
  - **Range: bytes=0-512, 2048-4096**

  used to request a byte range
- Responses
  - **Accept-ranges: bytes**

  indicates the server can respond to range requests
- Entity
  - **Content-Range: 0-399/2000**

  response to byte range request giving the byte ranges actually returned, e.g. the first 400 bytes of a 2000 byte document

Examples of HTTP Headers - *Response*

- **Age: 1246**

  age in seconds since response was generated
- **Location: http://www.myco.com/page.html**

  indicates that re-direction is desired
- **Public: GET, HEAD, POST, OPTIONS, PUT**

  methods supported by this web server
- **Server: Apache/1.3.1**

  identifies the server
- **WWW-AUTHENTICATE:**

  sent with 401 Unauthorized status code, it includes authorization parameters
- **Retry-after: 240**

  used with Service Unavailable status, indicates requested data will be available in 4 minutes

Examples of HTTP Headers - *Response*

- A URL may point to a document with multiple representations: languages, formats (html, pdf), or html features based upon user-agent
- if a French version is requested and cached, then a new request may fail to retrieve the English version
- HTTP/1.1 introduces Vary: accept-language, user-agent

  the header specifies acceptable languages and browsers,
- **e.g. the request is**

  GET http://www.myco.com/ HTTP/1.1

  User-agent: Mozilla/4.5

  Accept-language: en
- **the response is**

  HTTP/1.1 200 OK

  Vary: Accept-language

  Content-type: text/html

  Content-language: en

The proxy must store the fact that this doc has variants and when requested, get the proper variant

Examples of HTTP Headers - *Response*

- Warning: 10 proxy-id "Revalidation failed" messages indicating status information of the resource; HTTP/1.1 defines the following status codes

Code	Meaning
10	Response is stale
11	Revalidation failed
12	Disconnected operation
13	Heuristic expiration
14	Transformation applied
99	Miscellaneous warning

Entity Tags

- An **ETag** or **entity tag**.
  - one of several mechanisms that HTTP provides for web cache validation, and which allows a client to make conditional requests.
  - This allows caches to be more efficient, and saves bandwidth, as a web server does not need to send a full response if the content has not changed.
- An ETag is an opaque identifier assigned by a web server to a specific version of a resource found at a URL.
  - If the resource content at that URL ever changes, a new and different ETag is assigned.
  - ETags are similar to **fingerprints**, and they can be quickly compared to determine if two versions of a resource are the same or not.
- An ETag is a serial number or a checksum that uniquely identifies the file
  - caches use the **If-None-Match** condition header to get a new copy if the entity tag has changed
  - if the tags match, then a **304 Not Modified** is returned

Examples of HTTP Headers - Entity

- **Allow: GET, HEAD, PUT**
  lists methods supported by the URL
- **Content-Base: http://www.usc.edu/somedir**
  all relative references are taken wrt the base
- **Content-Encoding: gzip**
  indicates the encoding of the entity body; content-type indicates the media after encoding
- **Content-Language: en**
  identifies the language of the entity
- **Content-Length: 7890**
  specifies the length of the entity in bytes
- **Content-Location: http://www.usc.edu/myfile.htm**
  specifies the URL of the accessed resource

Examples of HTTP Headers - Entity

- **Content-MD5: base-64 encoded MD5 signature**

  contains the MD5 signature of the body as created by the web server
- **Content-type:text/html**

  indicates the MIME type of the object
- **Etag: "7776cdb01f44354af8bfa40c56eebcb1378975"**

  specifies the entity tag for the object, which can be used for re-validation; tags are unique ids determined by the server; this line is normally sent as a response
- **Expires: Wed, 30 Dec 2002 03:43:21 GMT**

  specifies the expiration date/time of the object; a cached copy should not be used beyond; Expires 0/now is immediate
- **Last-Modified: Wed, 30 Dec 2002 01:20:34 GMT**

  specifies the creation or last modification time of the object on the web server

HTTP Status Codes - *Informational*

- After receiving and interpreting a request message, a server responds with an HTTP response message.
- Syntax of response is

```
Status-Line
*( general-header | response-header | entity-header ) CRLF [ message-body ]
```

where the Status line is composed of

```
Status-Line = HTTP-Version Status-Code Reason-Phrase CRLF
```

HTTP Status Codes - *Informational*

Code	meaning
100	Continue, the client may continue with its request; used for a PUT before a large document is sent
101	Switching Protocols, switching either the version or the actual protocol

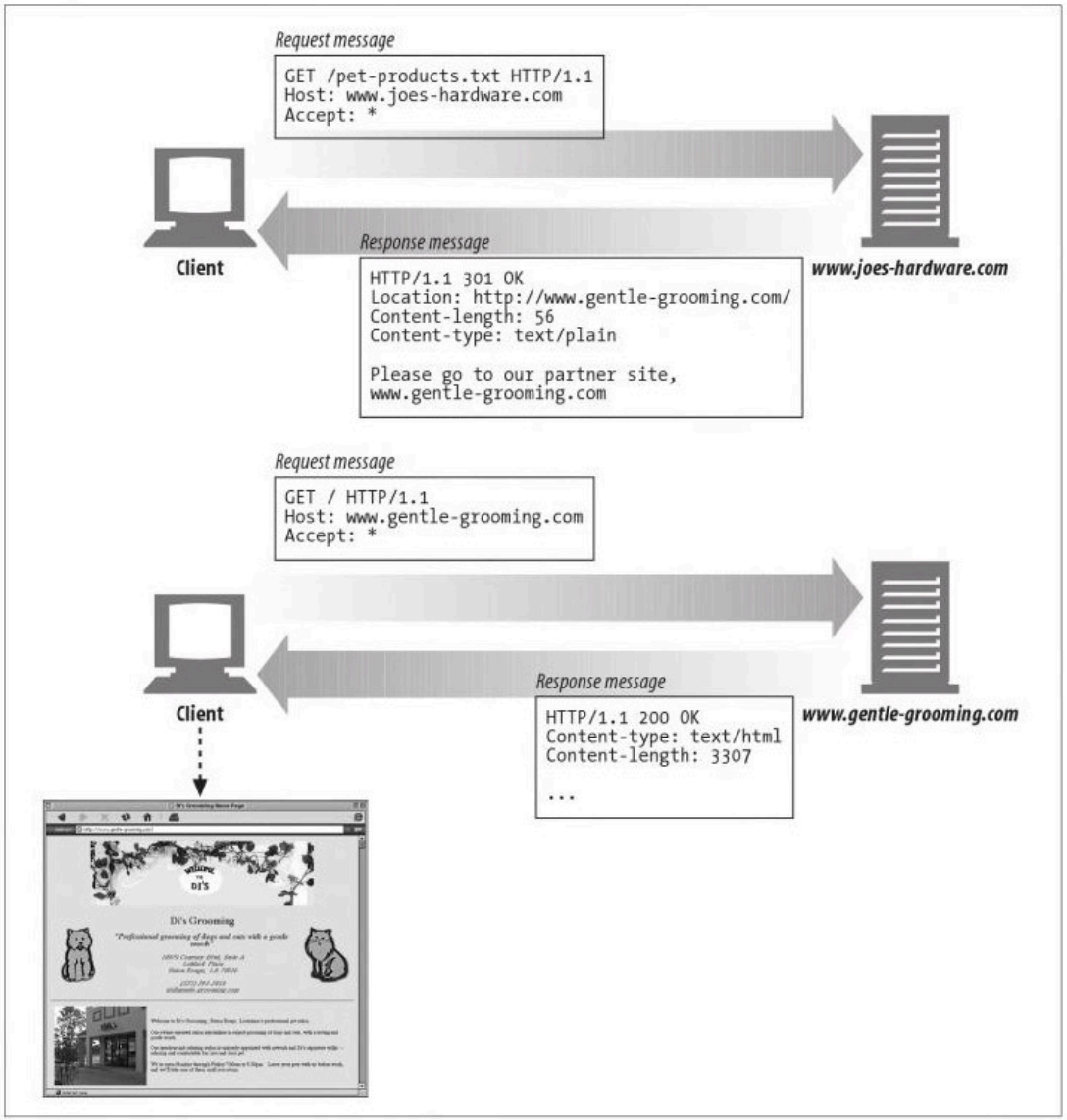
HTTP Status Codes - *Successful*

Code	meaning
200	OK, request succeeded
201	Created, result is newly created
202	Accepted, the resource will be created later
203	Non-authoritative information, info returned is from a cached copy and may be wrong
204	No content, response is intentionally blank, so client should not change the page
205	Reset Content, notifies the client to reset the current document, e.g. clear a form field
206	Partial content, e.g. a byte range response

HTTP Status Codes - *Redirection*

Code	meaning
300	Multiple choices, the document has multiple representations
301	Moved permanently, new location is specified in Location: header
302	Moved temporarily: similar to above
303	See Other: used to automatically redirect the client to a different URL
304	Not Modified: the client or proxy copy is still up-to-date
305	Use Proxy: make the request via the proxy
306	Proxy Redirection: a proposed extension to HTTP/1.1 still not specified
307	Temporary Redirect

Redirection Example



HTTP Status Codes - *Client Error*

Code	meaning
400	Bad request, server could not understand
401	unauthorized, authorization challenge
402	Payment Required, reserved for future use
403	forbidden, server refuses to fulfill request; e.g. check protections
404	Not found, document does not exist
405	Method not allowed, request method is not allowed for this URL
406	Not Acceptable, none of the available representations are acceptable to the client
407	Proxy Authentication Required, authentication is being challenged
408	Request Timeout, client did not send a request within a time given by server

HTTP Status Codes - *Client Error (cont'd)*

Code	meaning
409	conflict, the requested action cannot be performed
410	Gone, requested resource is no longer available
411	Length Required, Content-length header is missing
412	Precondition failed, a precondition for the request has failed, so request is canceled
413	request entity too large, entity is too large for server
414	Request URI too large, the request URL is too large
415	Unsupported media type, request entity is of an unsupported type

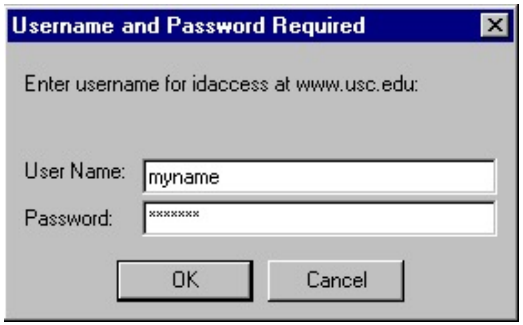
HTTP Status Codes - *Server Error*

Code	meaning
500	Internal server error, generic error code for the server
501	Not implemented, request could not be serviced because server does not support it
502	Bad gateway, intermediate proxy server received a bad response
503	Service unavailable, due to high load or maintenance on the server
504	Gateway timeout, intermediate proxy server timed out waiting for response from another server
505	HTTP version not supported

HTTP Authentication

- The web server can maintain secure directories and request authentication when someone tries to access them
- Procedure
  - web server receives a request without proper authorization
  - web server responds with

    401 Authentication Required
  - client prompts for username and password and returns the information to the web server
- we will show how to cause an authentication request when we discuss the web server features



Example HTTP Basic Authentication

- **Client makes a request on a secure page**

```
GET /secure/mypage.html HTTP/1.0
```

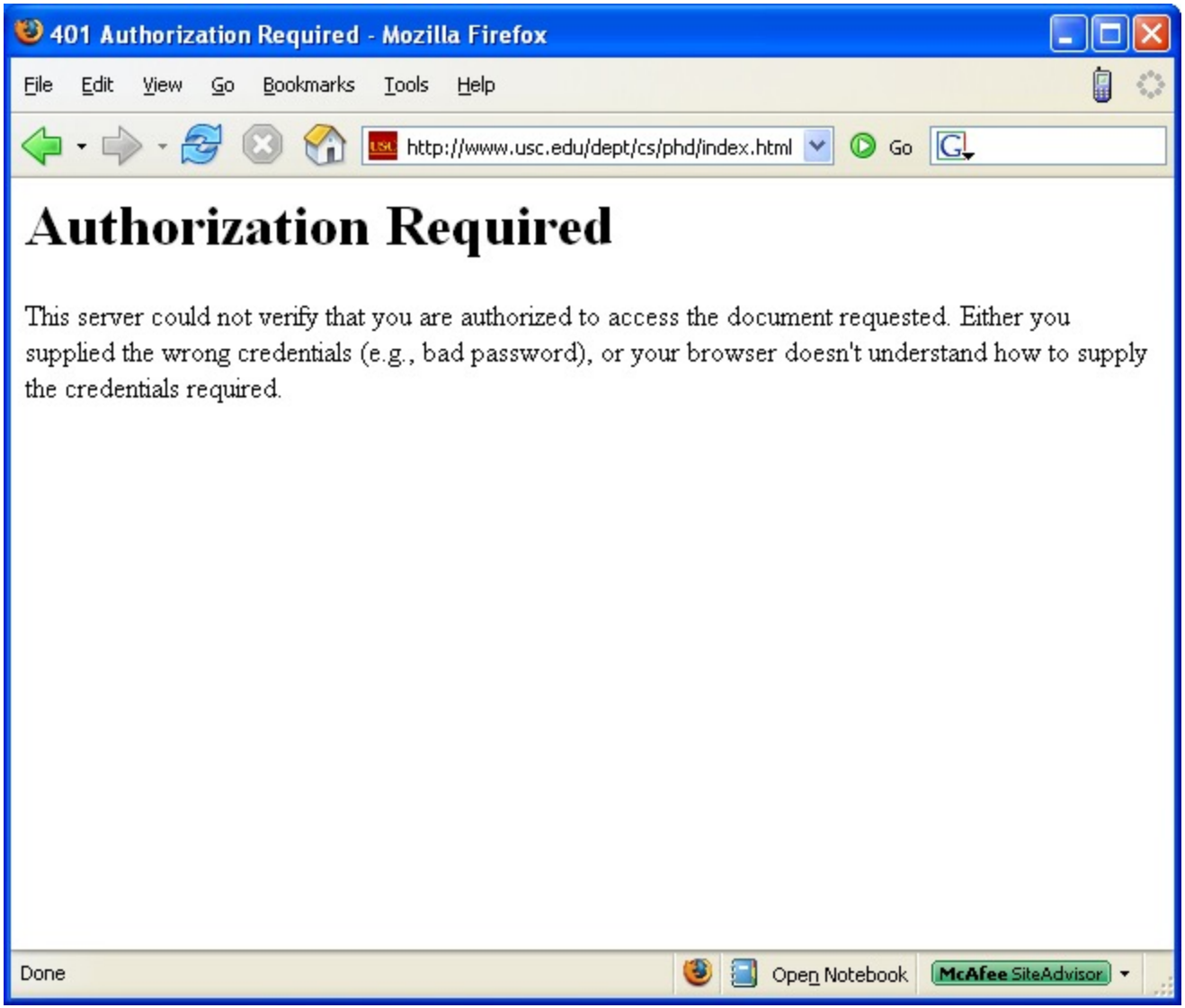
- **Server responds with**

```
HTTP/1.0 401 Unauthorized
Server: Apache 2.0
Date: Wed, 23 Dec 2002 15:42:00 GMT
WWW-Authenticate: Basic realm="notes server"
```

- **Client prompts for username and password and uses base-64 printable encoding; it re-issues its request, e.g.**

```
GET /secure/mypage.html HTTP/1.0
User-agent: Mozilla/4.5
Accept: text/html, image/gif, image/jpeg
Authorization: Basic MQDadmverWPUsvd=
```

Authorization Denied



META HTTP-EQUIV

- This is a mechanism for authors of HTML documents to set HTTP headers, in particular HTTP responses
- two common uses are
  - to set the **expiration time** of a document
  - to cause a **refresh** of a document
- E.g., here is an HTTP response header which causes the current page to be replaced by another page in 5 seconds

Refresh: 5; http://xyz.com/htdocs/mynewpage.html

this is equivalent to the following HTML

<META HTTP-EQUIV="Refresh" CONTENT=5,

http://csci571.com/index.html">

</HEAD><BODY> This page will be replaced by the csci571 home page in 5 seconds.

X-Frame-Options: SAMEORIGIN

Provides **Clickjacking protection**. Values: *deny* - no rendering within a frame, *sameorigin* - no rendering if origin mismatch, *allow-from: DOMAIN* - allow rendering if framed by frame loaded from *DOMAIN*

The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a **<frame>** or **<iframe>**. Sites can use this to avoid clickjacking attacks, by **ensuring that their content is not embedded** into other sites; options include:

DENY
   The page cannot be displayed in a frame, regardless of the site attempting to do so.
SAMEORIGIN
   The page can only be displayed in a frame on the same origin as the page itself.
ALLOW-FROM *uri*
   The page can only be displayed in a frame on the specified origin.

See:
   https://tools.ietf.org/html/rfc7034
   https://developer.mozilla.org/en-US/docs/Web/HTTP/X-Frame-Options

HTTP Strict-Transport-Security (HSTS)

HSTS is a security feature that lets a web site tell browsers that it should only be **communicated with using HTTPS**, instead of using HTTP.

The HSTS feature lets a web site inform a browser that it should never load the site using HTTP, and browser should automatically convert all attempts to access the site using HTTP to HTTPS requests instead. Supported by all "modern" browsers. Ignored when site accessed using HTTP.

Enabling this feature for your site is as simple as **returning the Strict-Transport-Security HTTP header** when your site is accessed over HTTPS:

Strict-Transport-Security: max-age=expireTime [; includeSubdomains]

expireTime
   The time, in seconds, that the browser should remember that this site is only to be accessed using HTTPS.
includeSubdomains (Optional)
   If this optional parameter is specified, this rule applies to all of the site's subdomains as well.
See:
   https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
   http://news.cnet.com/8301-1009_3-57524915-83/web-security-protocol-hsts-wins-proposed-standard-status/

Cross-origin resource sharing (CORS)

Cross-Origin Resource Sharing (CORS) allows many resources (e.g, fonts, JavaScript, etc.) on a web page to be requested across domains.

With CORS, **AJAX calls can use XMLHttpRequest across domains**. Such "cross-domain" requests would otherwise be forbidden by web browsers.

The CORS standard adds new HTTP headers. To initiate a CORS request, a browser sends the request with an "Origin" HTTP header. Suppose a page from http://www.social-network.com attempts to access user data from online-personal-calendar.com. If the browser supports CORS, this header is sent:

Origin: http://www.social-network.com

If the server at online-personal-calendar.com allows the request, it sends an Access-Control-Allow-Origin (ACAO) header in the response. The value of the header indicates what origin sites are allowed (*** = all sites**). For example:

Access-Control-Allow-Origin: http://www.social-network.com

If the server does not allow the CORS request, the browser will deliver an error instead of the online-personal-calendar.com response.

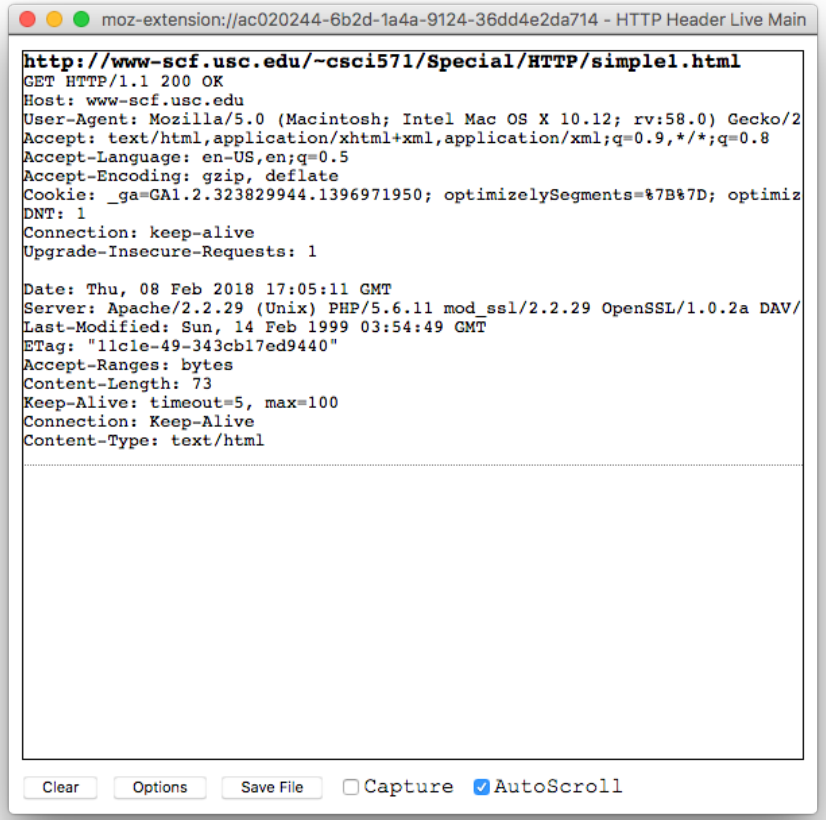
Firefox 3.5+, Safari 4+, Chrome3+, IE 10+, Opera 12+ support CORS. See:
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
- http://enable-cors.org/server_apache.html

Exercises to Examine the HTTP Protocol

1. HTTP Header Live
2. WireShark
3. Postman

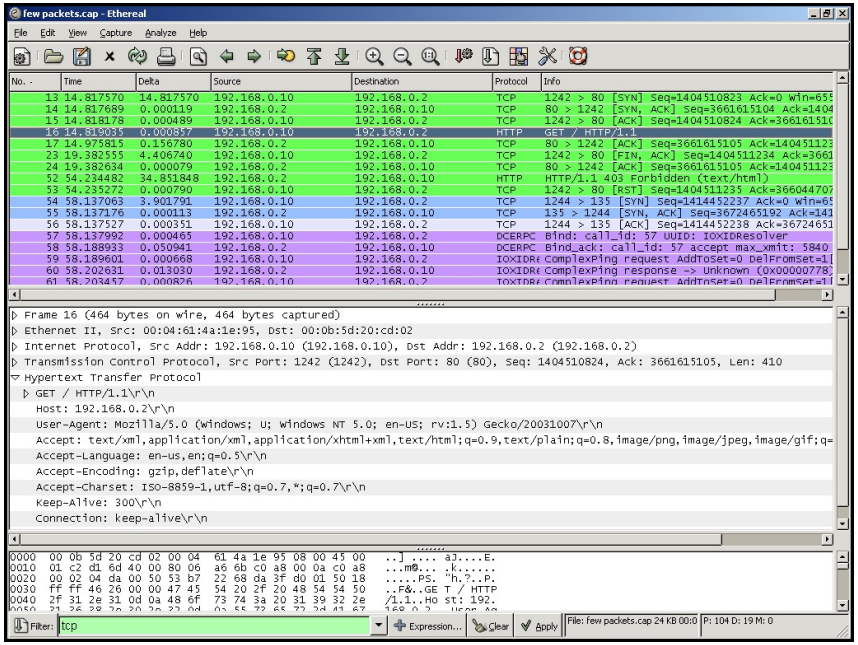
Firefox HTTP Header Live add-on

- Download at
  - https://addons.mozilla.org/en-US/firefox/addon/http-header-live/
  - Compatible with Firefox Quantum 57+



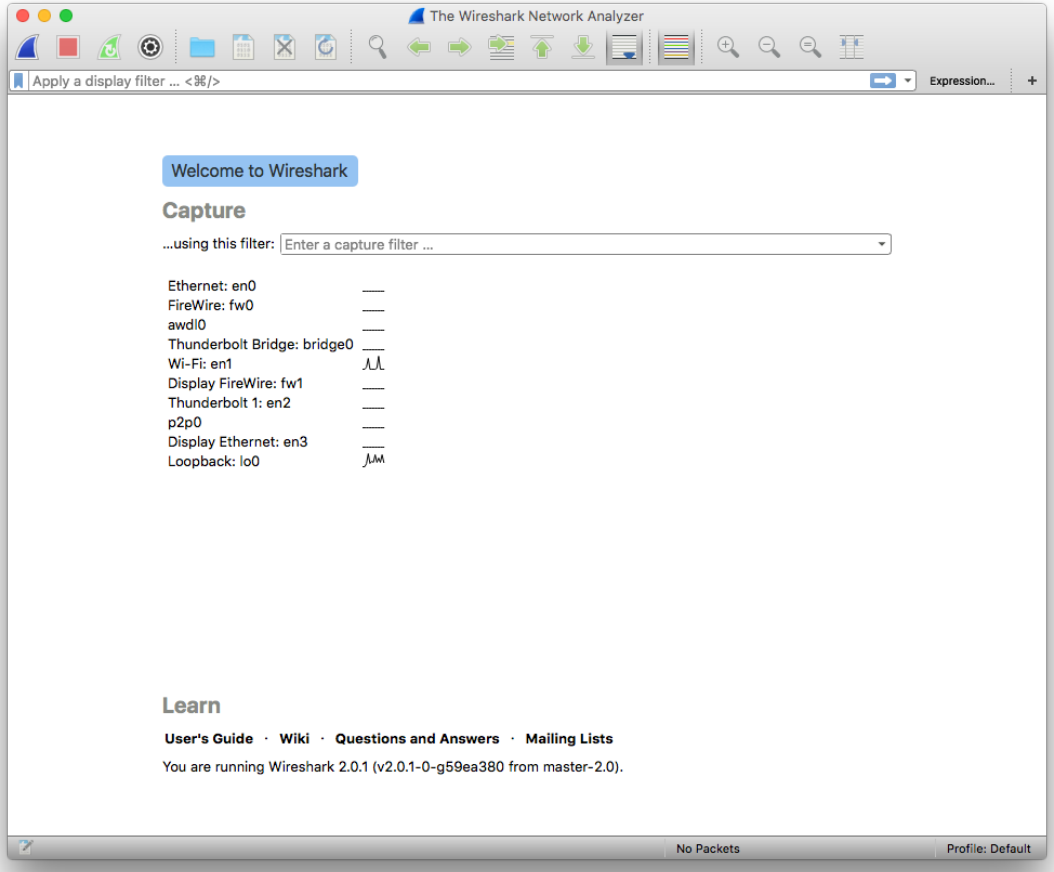
Wireshark Protocol Analyzer

- Install a protocol analyzer (a.k.a. a "sniffer"), such as **WireShark** (formerly Ethereal), available at
  - http://www.wireshark.org/
- Captured network data can be browsed via a GUI
- Supports 1300+ protocols
- Multiplatform: Windows, macOS, Linux, and many more



www.wireshark.org

- Download the latest "stable" release of WireShark (currently 3.4.6)



Initial Screen (new interface)

Copyright © 1999-2021 Ellis Horowitz & Marco Papa

Filter HTTP requests

- In your browser open up the page

http://csci571.com/Special/HTTP/proxy.html

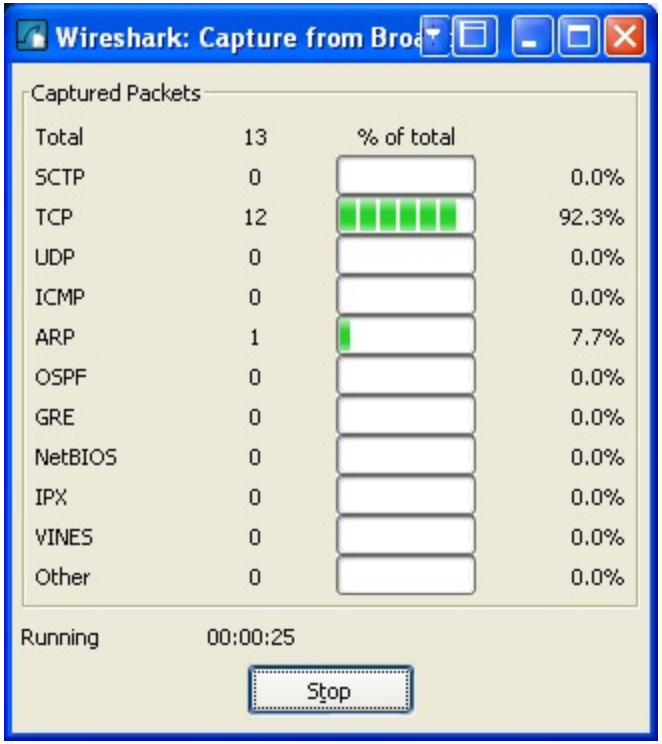
- In WireShark
  - Click on Expression on the right-hand side of the third row
  - Select HTTP and click on OK
  - Click on Apply on the third row at the right-hand end
- In WireShark
  - Click on Capture | Interfaces and then click the Start button that is on the line related to your wired/wireless connection to the Internet
  - A window appears as shown on the next slide

Start up the web Traffic

- Return to the browser and paste the first test URL into the address field

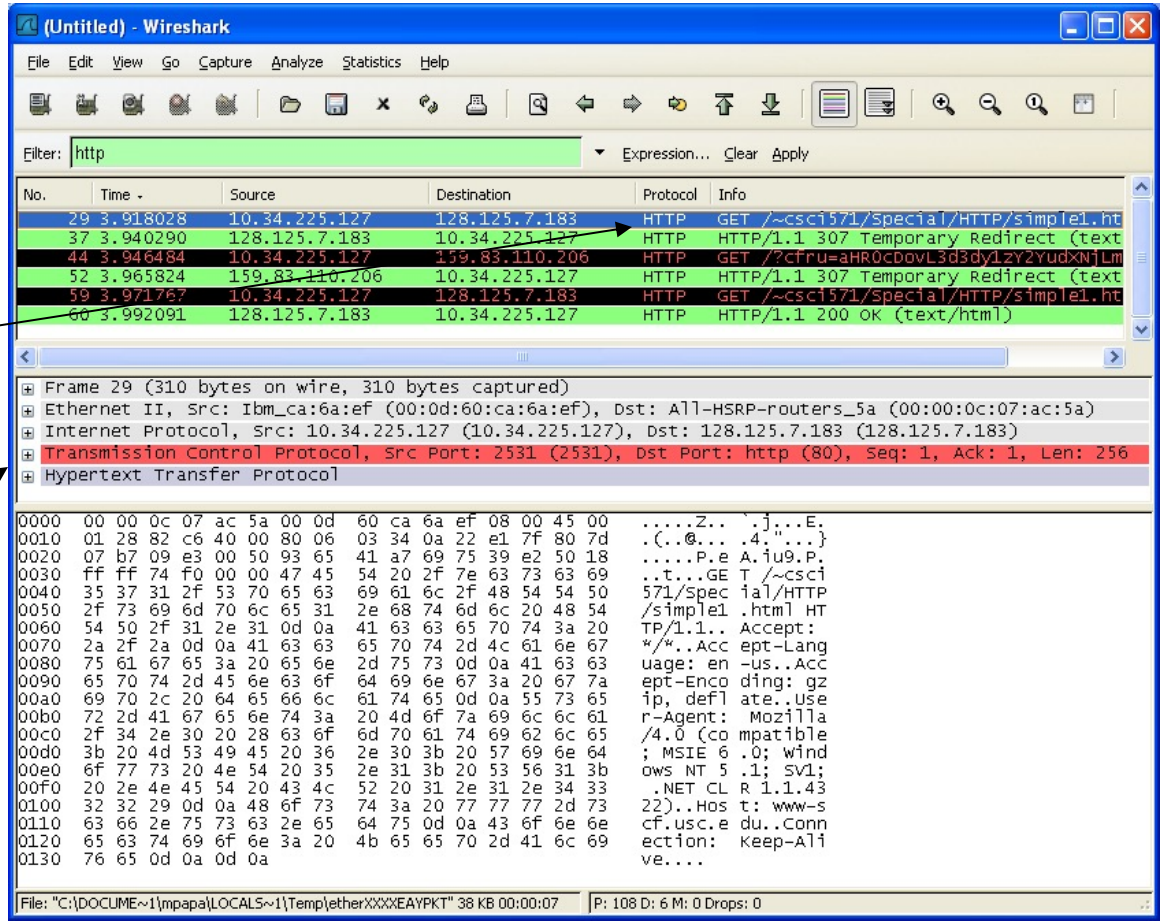
http://csci571.com/Special/HTTP/simple1.html

- Once the requested page appears in the browser return to the window that is shown on the right and click on the Stop button
- A new WireShark screen appears as shown on the next slide

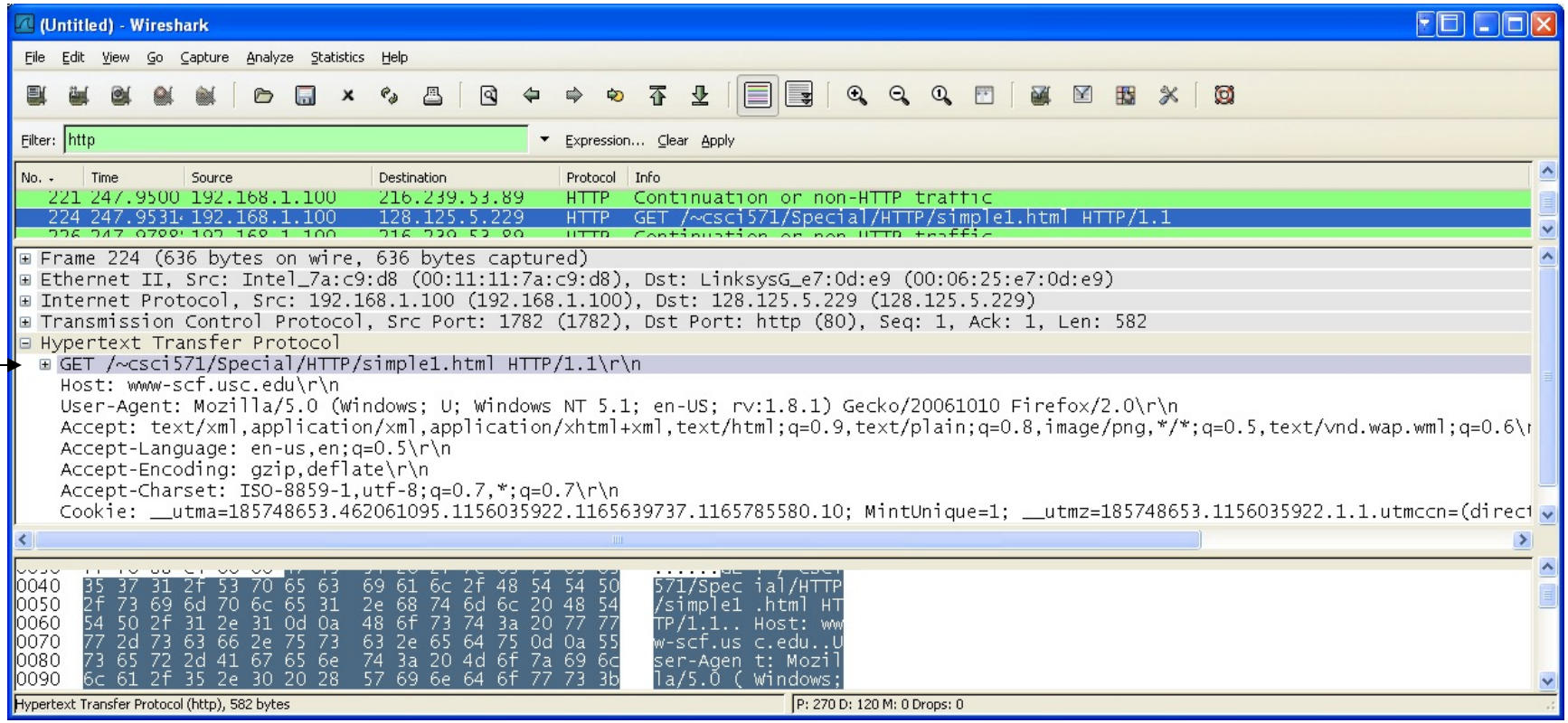


WireShark Output

- Scroll the green window until a line with HTTP GET appears;
- Click on the line;
- Then, in the middle window expand the line labeled Hypertext Transfer Protocol



Expansion of HTTP request



Notice the:

- GET request
- Host
- User-Agent
- Accepts
- Cookie

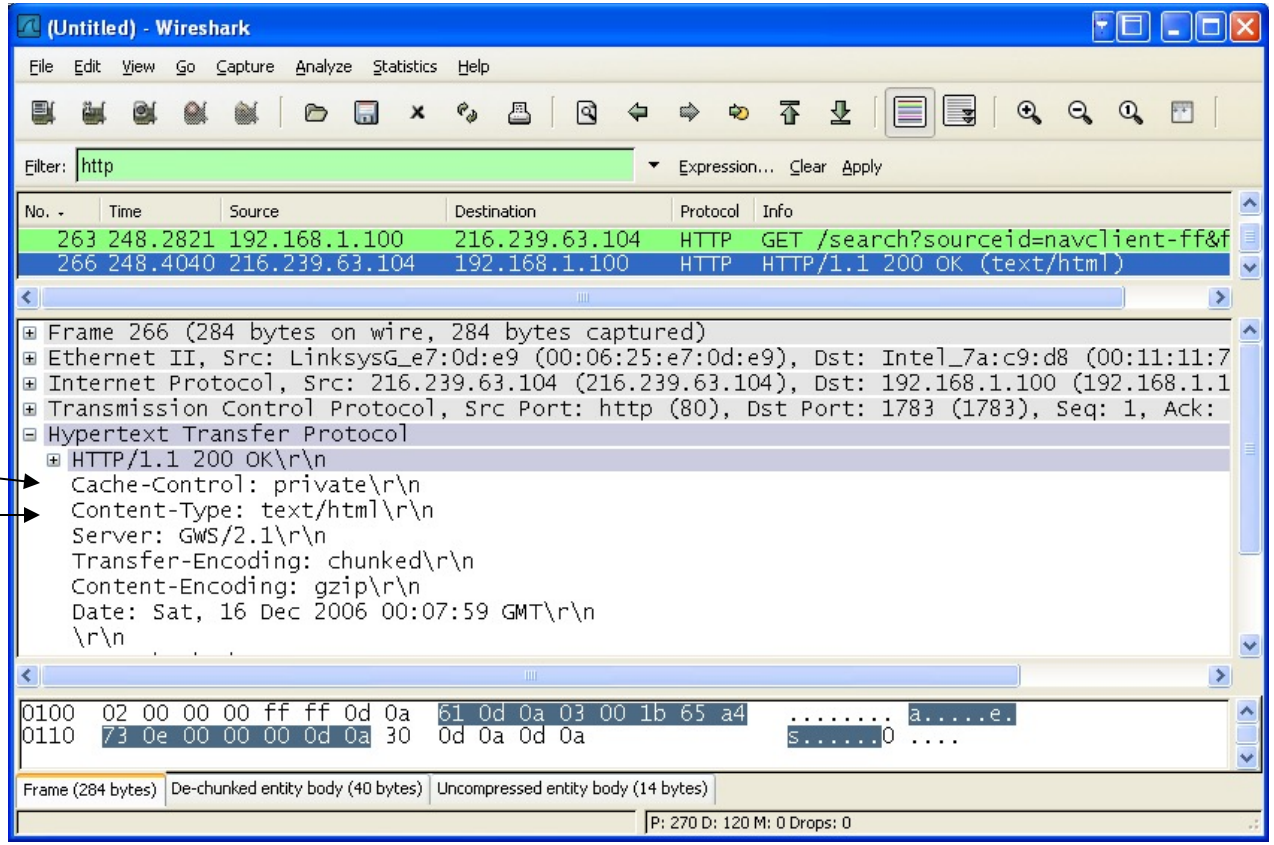
Expansion of HTTP response

- Scroll down again in the upper window and click on the HTTP/1.1 200 OK line, as shown to the right
- Notice the additional

lines that are sent including

- Cache-Control
- Content-Type
- Server

Etc.



Repeat the Exercise

- Using the remaining links on

http://csci571.com/Special/HTTP/proxy.html

do the following:

- In WireShark, select the Capture Start menu
- Click on the button “Continue without Saving”
- Enter the new URL into the browser
- Once the requested page appears click on Stop in the WireShark Capture window
- Examine the result in the main WireShark window
- Remember to note the HTTP commands, request, response and general commands

Postman

- Collaboration platform for API development
- Supports all HTTP commands. Download from: https://www.getpostman.com
- Free to individuals and small teams
- Free license includes 1,00 API calls / month
- Desktop app available for MacOS, Windows and Linux