Lecture Notes 11
HTTP Protocol

Anand Tripathi

CSci 4131

Internet Programming

Topics

- URL (Uniform Resource Locators)
- HTTP Protocol
  - Reference following documents
    - RFC 2616 HTTP/1.1 Protocol
    - Key Differences between HTTP/1.0 and HTTP/1.1

Uniform Resource Locator (URL)

- http://www.w3.org/Addressing/URL/Overview.html
- It is a naming scheme for referencing resources in the Internet. See RFC 1738.
- It allows resources to be accessed without knowing the specifics of their underlying protocol, such as FTP or HTTP.
- It is location-dependent scheme. I.e., a URL name becomes invalid if the resource is relocated to another host or moved to a different part of the file system.

Uniform Resource Locator (URL)

- URL scheme is based on DNS. It uses DNS to identify the Internet host of the resource.
- URLs belong to a more general class of naming scheme called Uniform Resource Identifiers (URI). URI also defines a location independent naming scheme called Uniform Resource Names (URNs). See URI
- http://www.w3.org/hypertext/WWW/Addressing/Addressing.html

URL

A URL can be in one of the two forms:

- Absolute or Complete URL
  - It specifies the complete access path for the named resources in the Internet.
- Relative or Partial URL
  - It is meaningful only in the context of some other URL.
  - It is used when the referenced resource is on the same host machine as the referring resource.

Absolute URL

The specification of an absolute URL contains the following information:

- Protocol to be used to access the resource: ftp, http, mailto
- DNS name or IP address of the server that contains the resource.
- If needed, specification of the server's port number.
- The directory path within which the resource is contained.
- The name of the file representing that resource.
- Some specific named component within the resource, such as a named "anchor" within an HTML document.
- Query parameters to be passed to the resource.

URL Structure

protocol :// host : port / directory / file #fragment?Query

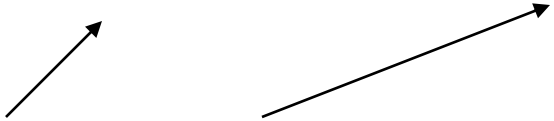
abs_path

- Port number can be omitted if the server is running on the well-known port for that service.
- For example, HTTP servers (port 80), HTTPS (port 443) FTP (port 21), Telnet (port 23).
- Fragment refers to a named "object" within the file (resource).
- Protocol can be:
  - http, telnet, ftp, mailto

Examples of URL

Example of a URL with encoded query with it.

http://www.cs.umn.edu/admissions/application.cgi?param1=value1&param2=value2



Parameters to be passed to the CGI program

This *(name, value) query* will be passed to the CGI program as the QUERY_STRING environment variable.

Examples of URL

- Example of URL with inclusion of single parameter to be passed to the resource.

http:// www.cs.umn.edu /admissions /application.cgi?someValue

Command line parameter to be passed to the CGI program

Telnet URL

telnet://hostName:optionalPortNumber

For example: Login to whale.itlabs.umn.edu

HTML code looks like this:

```
<a href="telnet://whale.itlabs.umn.edu"> Login to
    whale.itlabs.umn.edu </a>
```

MAILTO URL

mailto:user@host

For example:

Send mail to student@itlabs.umn.edu

HTML code looks like this:

```
<a href="mailto://student@itlabs.umn.edu"> Send mail
    to student@itlabs.umn.edu </a>
```

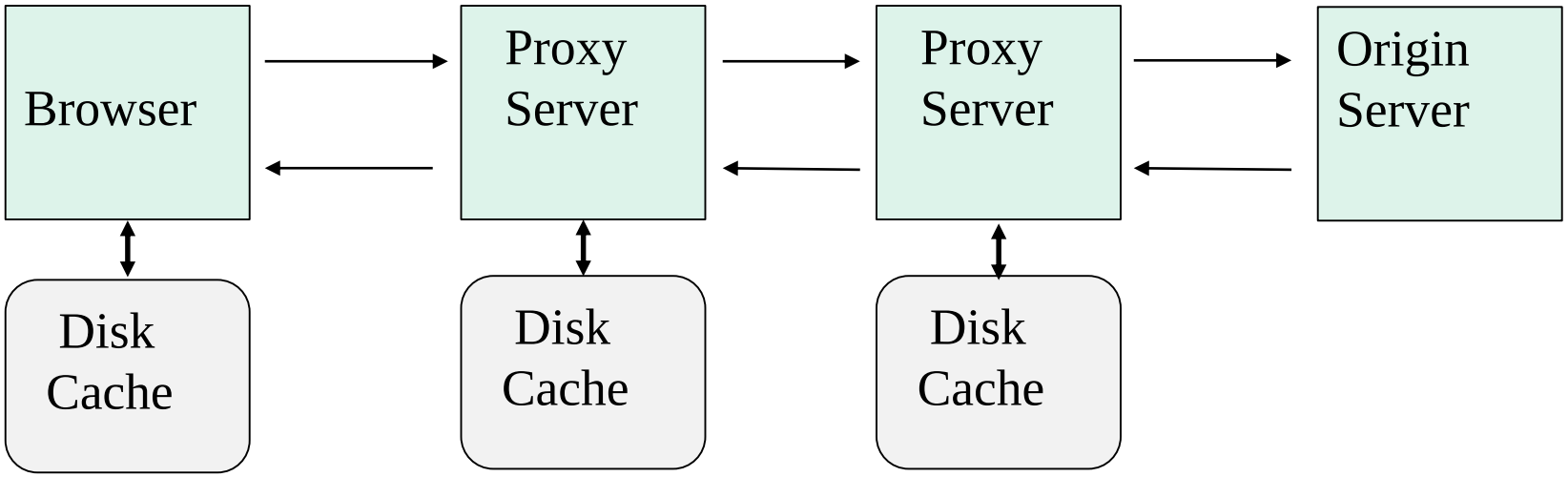
HTTP Protocol

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

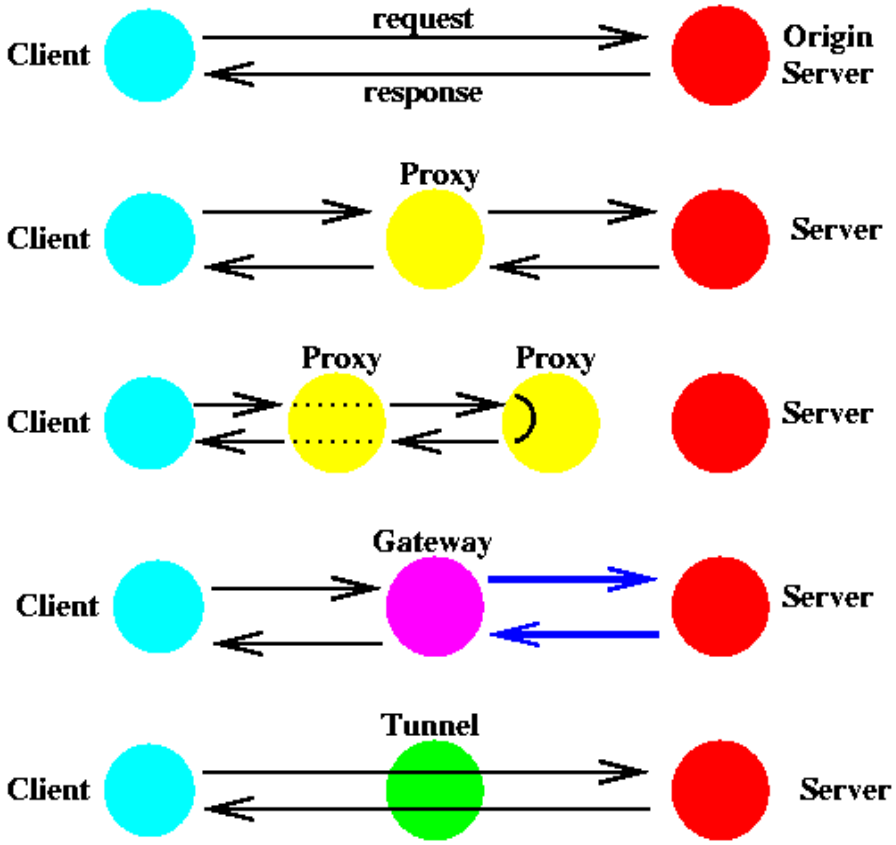
This serves as the foundation of the World Wide Web.

- Request messages contains 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 indicate the status of the request processing and possibly some response data.

Proxies and Caching



HTTP Protocol Terminology



HTTP Protocol Terminology

See RFC 2616 (HTTP 1.1) http://www.w3c.org/Protocols/

Read Section 1.3 of RFC 2616 for terminology.

Client: A process initiating a request

Server: A process accepting HTTP protocol requests.

Origin server: The server on which a given resource resides or is to be created.

Proxy: An intermediate program that can serve both as a client and a server. A "transparent proxy" does not modify a request or response beyond what is needed for proxy authentication.

HTTP Protocol Terminology

Gateway:

A server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway.

Tunnel:

An intermediary program which is acting as a blind relay between two connections. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel may have been initiated by an HTTP request. The tunnel ceases to exist when both ends of the relayed connections are closed.

HTTP Protocol

HTTP/0.9 This was a simple protocol for raw data transfer.

HTTP/1.0 (RFC 1945)

- Included support for MIME (Multipurpose Internet Mail Extensions) format for messages.
- Modifiers for request and response semantics
  - Conditional GET
- No support for caches and proxies
- Each request to a given server required a new connection to be opened
- No support for intermediate proxies and caching of responses

HTTP/1.1 Protocol

HTTP/1.1 (RFC 2616)

- Support for proxies and caching of data
- Persistent connection
  - Avoid creating new connection for each request
  - Supports pipelining of requests
- Support for dynamically generated content for which the total length of the message my not be known when the response message is created.
  - Chunked transfer coding
- Support for multi-homed servers
  - One machine (i.e. IP address) supporting multiple virtual hosts

Protocol Specification in Augmented BNF

- See Section 2 of RFC 2616

- Grammar rules described as:

  name = definition

  *element — 0 or more repetitions of element

  <n>*<m> element — at least n or at most m repetitions

  *1 means the same as 0*1

  [element] — optional element, 0 or 1 occurrence

  #element — a list of elements, separated by comma

  n#m element — a list or at least n and up to m elements

HTTP 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.

Example of HTTP Request Message

GET http://www-users.cselabs.umn.edu/classes/Spring-2014/csci4131 HTTP/1.1

Host: www-users.cselabs.umn.edu

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:28.0) Gecko/20100101 Firefox/28.0

Accept: text/html,application/xhtml+xml,application/xml;

Accept-Language: en-US

Accept-Encoding: gzip, deflate

Referer: http://www-users.cselabs.umn.edu/classes/Spring-2014/csci4131/menu.html

Cookie: __utmb=112294656.0.10.1395694183;

Connection: keep-alive

/* Blank line */

Example of HTTP Response Message

HTTP/1.1 200 OK
Date: Mon, 24 Mar 2014 20:54:44 GMT
Server: Apache
Last-Modified: Thu, 06 Mar 2014 00:11:07 GMT
Content-Encoding: gzip
Content-Length: 5195
Connection: close
Content-Type: text/html
/* blank line */
Entity body -- HTML document

REQUEST LINE

Method SP Request-URI SP HTTP-VERSION CRLF

Here CRLF stands for "carriage return" (C-UNIX character '\r' (ascii 13) and new line character '\n' (ascii 10).

SP stands for space.

Method can be one of: GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE, CONNECT (case-sensitive).

1. GET to retrieve the contents of a resource.
2. POST to update or modify a resource (e.g. adding item to a database).
3. HEAD to obtain meta-information related to the resource (e.g. last modification time).
4. PUT to create a resource at the server
5. OPTIONS is used to retrieve server options for communication
6. CONNECT is used for establishing a tunnel through a proxy server, usually for SSL (Secure Socket Layer) connection through proxy

See Section 9 of RFC 2616

General Structure of a Request Message

REQUEST-LINE
* (GENERAL-HEADERS
| REQUEST-HEADER
| ENTITY-HEADER
)
CRLF
[ENTITY-BODY]

See Section 10 or RFC 1945.

In general, all HTTP headers (general, request, response, entity) have the form:

FIELD-NAME ":" [FIELD-VALUE] CRLF

For example:

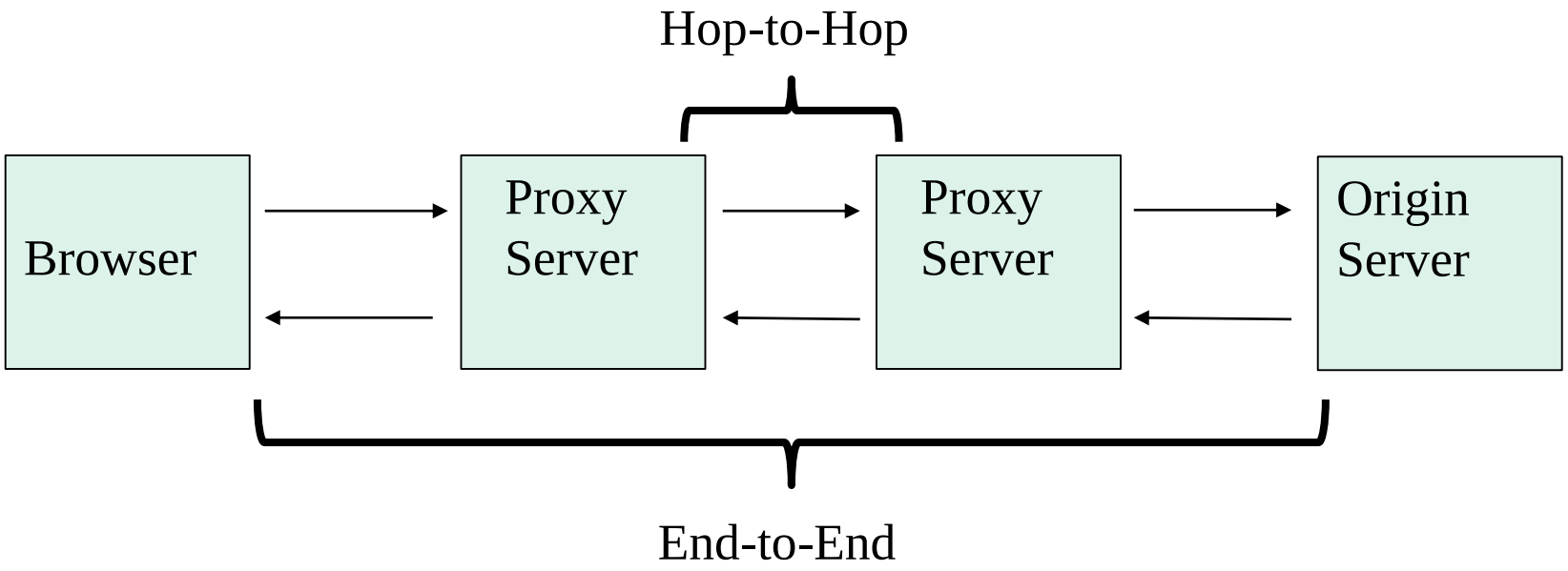
If-Modified-Since: Mon, 29 Sep 2008 12:43:49 GMT

Content-Length: 4687

Content-Type: text/html

- General headers are applicable to any kind (request or response) message.
  - For example "Date“, “Cache-Control”, “Transfer-Encoding”, “Connection”, “Trailer”
- Request headers are applicable to request messages.
  - For example: "If-Modified-Since“, “Host”
    - HOST field is always needed when a request is sent to an origin server.
- Entity headers are applicable to the the body of the enclosed entity.
  - For example: “Content-type” or "Context-length"
- The order of the headers is not important.

Hop-to-Hop and End-to-End Communication and Headers



End-to-End and Hop-to-Hop Header

See Section 13.5 of RFC 2616

- End-to-End Headers:
  - These are transmitted to the final destination of a request or a response.
- Hop-to-Hop Headers:
  - These are meaningful over a single transport level connection, and are not stored by caches or forwarded by proxies:
  - Examples:

    Keep-Alive : keep alive for a given time a connection for next request/response interactions

    Other headers: Connection, Proxy-Authenticate, TE, Trailers, Transfer-Encoding

  Hop-to-Hop headers must be removed from one hop to the next.

Non-modifiable Headers

- A transparent proxy must not modify the following headers:
- Content-Location
- Content-MD5
  - Hash value of entity using MD5
- Etag
  - Some opaque validator for conent, eg. Timestamp, or hash
- Last-Modified
- Expires
- A proxy must not add or modify the following fields if cache control specifies *no-transform*:
  - Content-Encoding, Content-Range, Content-Type

Example of HTTP Request

A sample example request:

GET /~tripathi/index.html HTTP/1.1
HOST: www.cs.umn.edu
Accept: image/gif
Accept: image/jpeg
User-Agent: Mozilla/4.05 [en] (X11; I; Linux 1.3.20 i486)
From: tripathi@cs.umn.edu
* a blank line *

REQUEST-URI

- REQUEST URI can be in one of the following two forms:
- absoluteURI: such as
  http://www.w3.org/pub/WWW/Project.html
- File-path: or abs_path only referring to the path information in the URL.
  For example:
  /pub/WWW/Project.html

- Absolute URI should be used only when sending request to a proxy. Otherwise, when sending a request to an origin server, abs_path form of URI should be used.

REQUEST HEADERS

- Authorization: Supply credientials to the server in the field value.
- From: email address of the user

  From: webmaster@cs.umn.edu
- If-Modified-Since: used in "Conditional GET" requests

  If-Modified-Since: Mon, 29 Sep 2008 12:43:49 GMT
- Referer: this is the URI of the document referring the requested resource The URI can be absolute or relative.

  Referer:
  http://www.cs.umn.edu/5131/coursepage.html
- User-Agent: information about the browser
- See Section 5.3 of RFC 2616

REQUEST HEADERS

- Host: This field indicates the DNS hostname and port number for the origin server for the requested resource. It is extracted from the URL for the resource.
- Example:

  Get /PUB/www HTTP/1.1

  Host: www.w3.org
- HTTP 1.1 protocol requires that Host field must be included in a request when sending request to the origin server.

ENTITY HEADERS

- Context-Encoding: encoding of the resource

  Content-Encoding: x-gzip
- Content-Length: length in bytes of the entity

  E.g. Content-Length: 5689
- Content-Type: type/subtype

  Content-Type: image/gif

  Content-Type: text/plain

  Content-Type: text/html
- Expires: expiration time in GMT

  Expires: Mon, 29 Sep 1998 12:43:49 GMT
- Last-Modified: last modification time in GMT

  Last-Modified: Mon, 29 Sep 1998 12:43:49 GMT

RESPONSE MESSAGE

HTTP/1.1 200 OK
Date: Tue, 29 Sep 1998 12:43:49 GMT
Server: Apache/1.3.1 (Unix)
Last-Modified: Tue, 22 Sep 1998 14:43:08 GMT
Connection: close
Content-Type: text/html

<HTML>
<HEAD>
< TITLE> University of Minnesota
-- Department of Computer Science and Engineering </TITLE>
.....

General Structure of a Response Message

STATUS-LINE
* (GENERAL-HEADERS
| RESPONSE-HEADER
| ENTITY-HEADER)
CRLF
[ENTITY-BODY]

The general format of the STATUS LINE is:
HTTP-VERSION Status-Code ReasonsePhrase CRLF

RESPONSE HEADERS

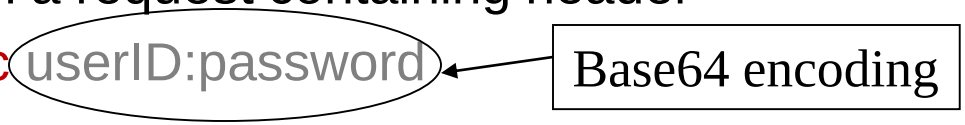
- Date Date and time when the response was generated
- Location: indicates the exact location of the resource by its absoluteURI. For 3xx responses, it indicates server's preferred location for redirection.

  Location: http://www-users.cs.umn.edu/~tripathi
- Server: information about the software used by the origin server.

  Server: Apache/1.3.1 (Unix)
- WWW-Authenticate: must be included with 401 (unauthorized) response. WWW-Authenticate: one or more "challenges" or authentication parameters

  WWW-Authenticate: Basic realm="4131Students"
- User agent responds with a request containing header

  Authorization: Basic userID:password



DATE FORMATS

- There are three formats for specifying date/time.
  All formats express time as GMT.
- Tue, 29 Sep 1998 12:43:49 GMT

  // RFC 1123 THIS IS THE PREFERRED FORMAT.

- OTHER TWO ARE:
  Tuesday, 29-Sep-98 12:43:49 GMT

  // RFC 850 - NOT TO BE USED
  Tue Sep 29 12:43:49 1998 // ANSI C format

Response Codes:

Category 2xx is used to indicate SUCCESS

Code Significance and Use

200 OK

201 New resource created at the server

202 Accepted for processing. Not completed.
Non-committal.

203 Non-authoritative content

204 No contents

205 Server has processed the request and client
should reset the document view (e.g. form data cleared).

206 Partial content

Response Codes:

Category 3xx is used for re-direction

Code	Significance and Use
300	Multiple choices for accessing the resource
301	Moved permanently to another location This response gives new location.
302	Moved temporarily
303	See other (resource available under different URI)
304	Not modified In response to conditional-GET
305	Use proxy

Response Codes:

Category 4xx is used for indicating client error.

Code	Significance and Use
400	Bad request
401	Unauthorized
402	Payment required. (Return this code to get rich.)
403	Forbidden. Server understands what you are trying to do. Authorization would not help.
404	Not found
405	Method not allowed
406	Not acceptable
410	Gone

Response Codes:

Category 5xx is used for indicating internal server errors.

Code	Significance and Use
500	Internal server error.
501	Requested method is not implemented.
502	Bad gateway. The server, while acting as a proxy or gateway, received an invalid response from some upstream server.
503	Service unavailable. Overload possibility.

HTTP 1.1 Protocol

- Faster response for dynamically-generated pages, by supporting *chunked encoding*, which allows a response to be sent before its total length is known.
- Retrieval of partial data of an object.
- Several conditional GET options in addition to IF-MODIFIED-SINCE
- A server can be home of web servers for multiple domains.
- Allows multiple transactions to take place over a single *persistent connection*. → Faster response
- Provides extensive support for cache management.

Key Differences between HTTP 1.0 and HTTP 1.1

See the paper: "Key Differences between HTTP/1.0 and HTTP/1.1" (It is posted in the References section.)

- Extensibility
- Message Transmission (encoding)
- Caching
- Bandwidth optimization
- Connection management
- Multi-home servers (IP address conservation)
- Authentication
- Content Negotiation

Extensibility of the HTTP Protocol

Extensibility

- HTTP/1.1 was designed to be compatible with HTTP/1.0.

- Support for future extensibility:
  - New type of headers can be included
  - If an implementation does not understand a header then it must ignore the header.
- The intermediate proxy servers may use different protocol
  - HTTP protocol is between two hops in route.
    - Hop-to-Hop headers (between two ends-points of a hop)
    - End-to-End header (between the client and the origin-server)
  - Via header can be used to indicate the protocol versions of the proxy servers from client to the origin-server

Message Transmission

Message Transmission

- Recipient of a message should be able to determine the end of a message.
  - Message may be of arbitrary length
- Content-Length header is used to indicate the length of a message.
  - Limitation when content is dynamically generated.
    - Message transmission can begin only after the entire document is generated
- Chunked transfer coding
  - This is to support dynamically generated documents whose content length is not known till the document has been generated.
  - Server can transmit chunks of data as they are produced.

Chunked Transfer-Encoding

- If a server wants to start sending a response before knowing its total length (e.g. CGI program output), it might use the *chunked transfer-encoding*, which breaks the complete response into smaller chunks and sends them in series as they are generated.

- You can identify such a response because it contains the header:

  Transfer-Encoding: chunked

- All HTTP 1.1 clients must be able to receive chunked messages.

Chunked Transfer-Encoding

- A chunked message body contains a series of *chunks*, followed by a line with "0" (zero), followed by optional footers (just like headers), and a blank line. Each chunk consists of two parts:
  - a line with the size of the chunk data, in hex, possibly followed by a semicolon and extra parameters you can ignore (none are currently standard), and ending with CRLF.
  - the data itself, followed by CRLF.

Example of Chunked Transfer Encoding

HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/plain
Transfer-Encoding: chunked

1a; ignore-this-is-a-comment
abcdefghijklmnopqrstuvwxyz
10
1234567890abcdef
0
some-footer: some-value
another-footer: another-value
[blank line here]

Previous Example is equivalent to the one below

HTTP/1.1 200 OK

Date: Fri, 31 Dec 1999 23:59:59 GMT

Content-Type: text/plain

Content-Length: 42

some-footer: some-value

another-footer: another-value

abcdefghijklmnopqrstuvwxyz1234567890abcdef

Notion of Trailers

- When data is dynamically generated and transmitted as chunks, some information cannot be included in the header. E.g.
  - Content length
  - Content-MD5 for cryptographic checksum (hash value).
  - These items can be computed only after generating the complete data.
- In HTTP/1.1 a chunked message may include a trailer after the last chunk.
- Trailer is a set of header fields
- A header called Trailer is included in the usual headers to alert the recipient that the transmitted message contains some trailers.

Rule for including Trailers

A server can include a trailer under one of the two conditions:

1. The information included in the trailers is optional.

2. Client has sent a TE: trailers header to indicate that it is willing to receive trailers.

Content Length

- If the message is sent using “chunked” encoding, there will not be any Content-Length header or trailer.

- See Section 4.4 for determining message length
- A response with 1xx, 204, 304, and for HEAD request must not include a message-body
- “chunked” transfer indicates its chunk lengths
- Content-Length field indicates the message-body length
- Connection closing by the server during the response transmission indicates end of the message.

IP Address Conservation

Host Header

- Starting with HTTP 1.1, a server at one IP address can be *multi-homed*, i.e. the home of several Web domains. For example, "www.host1.com" and "www.host2.com" can live on the same server.

- Therefore, every HTTP request must specify which host name (and possibly port) the request is intended for, with the Host: header.

- A complete HTTP 1.1 request might be

  GET /path/file.html HTTP/1.1
  Host: www.host1.com
  [blank line here]

Network Connection Management

Connection Header

- Hop-to-Hop headers apply only to a given connection between the two end points of a hop. Example:
  - Transfer-Encoding (sent by the server in response)
  - TE (sent by client to indicate what encoding it can accept)
  - Keep-Alive
- These header should not be forwarded to the next hop destination.
- Connection header lists all such headers which should be removed when a message is forwarded along the next hop.
- Connection header may also have some connection-tokens which are Boolean flags:
  - close to indicate not to have persistent connection

Persistent Connections

- In HTTP 1.0 , TCP connections are closed after each request and response, so each resource to be retrieved requires its own connection.

- Opening and closing TCP connections takes a substantial amount of CPU time, bandwidth, and memory.

- Most Web pages consist of several files on the same server, so much can be saved by allowing pipelining of several requests and responses to be sent through a single *persistent connection*.

Persistent Connections

- Persistent connections are the default in HTTP 1.1, so nothing special is required to use them.
- Just open a connection and send several requests in series (called *pipelining*), and read the responses in the same order as the requests were sent.
- You need to be careful to read the correct length of each response, to separate them correctly.

Keep-Alive Header

- Hop-to-Hop header
- Keep-Alive is used by the

Connection: Keep-Alive

Keep-Alive: timeout=10, max=20

- Timeout indicates the time for which an idle connection will be maintained open
- Max indicates the maximum number of requests a client will make or a server will accept on a persistent connection.

Persistent Connections

- If a client includes the "Connection: close" header in the request, then the connection will be closed after the corresponding response.

- Use this if you don't support persistent connections, or if you know a request will be the last on its connection.
-
- Similarly, if a response contains this header, then the server will close the connection following that response, and the client shouldn't send any more requests through that connection.

Persistent Connections

- A server might close the connection before all responses are sent, so a client must keep track of requests and resend them as needed.
- When resending, don't pipeline the requests until you know the connection is persistent.
- Don't pipeline at all if you know the server won't support persistent connections (like if it uses HTTP 1.0, based on a previous response).

Bandwidth Optimization

Motivations for HTTP/1.1 Features

Motivation is to conserve network bandwidth:

- Do not send data for an entire object when the client needs only some parts of it.
  - "Range" requests

- Clients should not send a large object to a server without knowing if the server would accept it.
  - Negotiate with server before sending a large data item

Range Header

- Range header allows a client to request parts of a resource.
- A request can specify one of more contiguous ranges of bytes.

Examples:

Range: bytes=1024-8191

Range: bytes=0-128,1024-8191

Range: bytes=-100 //last 100 bytes

Range: bytes=0-0,-10 // first byte, and last 10 bytes

Accept-Ranges allows a server to indicate the range requests it supports. Value “none” means it does not support range requsts.

Range Response

- Response message will have status 206 to indicate partial data.
- Content-Range header indicates the offset and length of the returned range.
- MIME type for such data is multipart/byteranges

Use of Range Requests

- Get only some important part of an object, e.g. image geometry for layout rendering.
- Read tail part of a growing object.
- If an object was received only partially due to connection interruption, then request the missing part only.
  - In such cases make a conditional request to get the missing range only if the object has not changed.

"100 Continue" Response

Purpose of 100-Continue response:

Suppose that a client wants to send a large data item to a server, but before transmitting the data it wants to make sure that the server will accept the request.

A client may just send the request headers and wait for some time before sending the entity body, this is to avoid useless transmission of data if the sever may not serve that request.

Client sends Request Header

Expect: 100-Continue

It waits for a response from the server

HTTP/1.1 100 Continue

before transmitting the data

"100 Continue" Response

- During the course of an HTTP 1.1 client sending a request to a server, the server might respond with an interim "100 Continue" response. This means the server has received the first part of the request, and can be used to aid communication over slow links.
- The "100 Continue" response is structured like any HTTP response, i.e. consists of a status line, optional headers, and a blank line.
- Unlike other responses, it is always followed by another complete, final response.
- Client must send

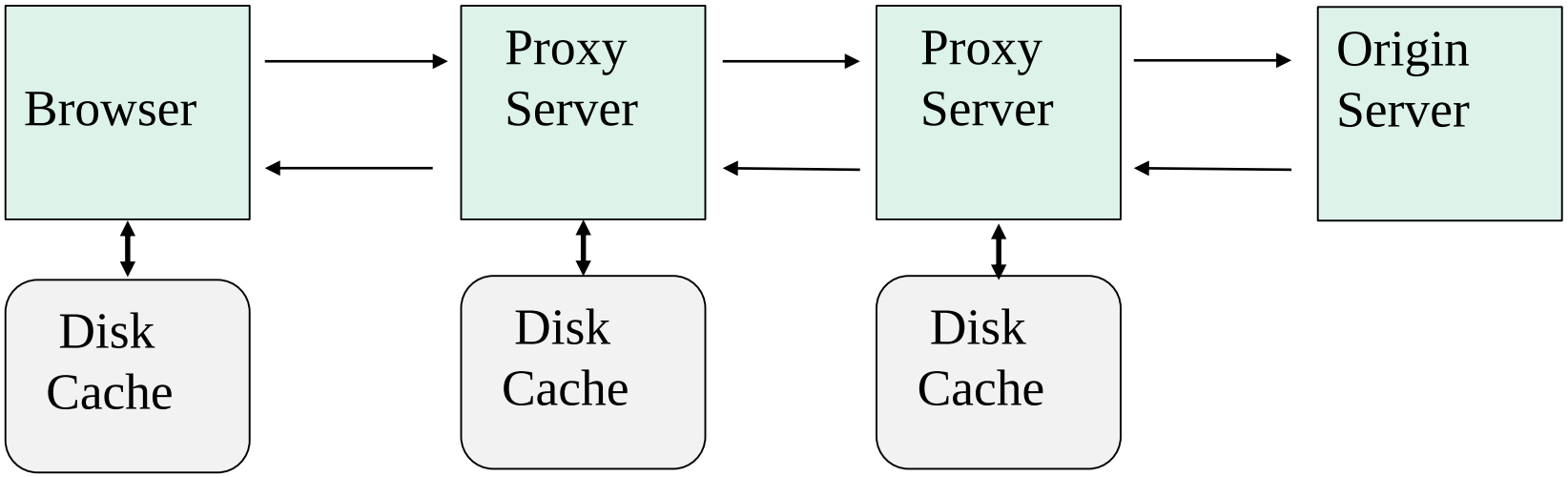
  Expect: 100-continue in the request headers

Data Compression

- Conserve bandwidth by compressing data.
- Distinction between content-coding, and transfer-coding
  - Content coding ➔ end-to-end
  - Transfer coding ➔ hop-to-hop
  - Compression may be applied at these two levels
- Content-Encoding header is used specifying for end-to-end encoding
  - It may be inherent to the resource. E.g. zip file
    - Client may specify Accept-Encoding header for acceptable encodings
- Transfer-Encoding header is hop-to-hop
    - Client may specify TE header to specify acceptable encodings

Caching

Proxies and Caching



Advantages of Caching

- Reduces latency perceived by clients if data is local
  - No need to send request to the origin server.
- Reduces network traffic and bandwidth consumption
  - Helps in reducing network latency for other requests
- Reduces load on the origin server

- Requirement: Semantically transparent caching
  - The response returned from the cache should be the same as what one would have obtained from the origin server.

Limitations of HTTP/1.0 Caching

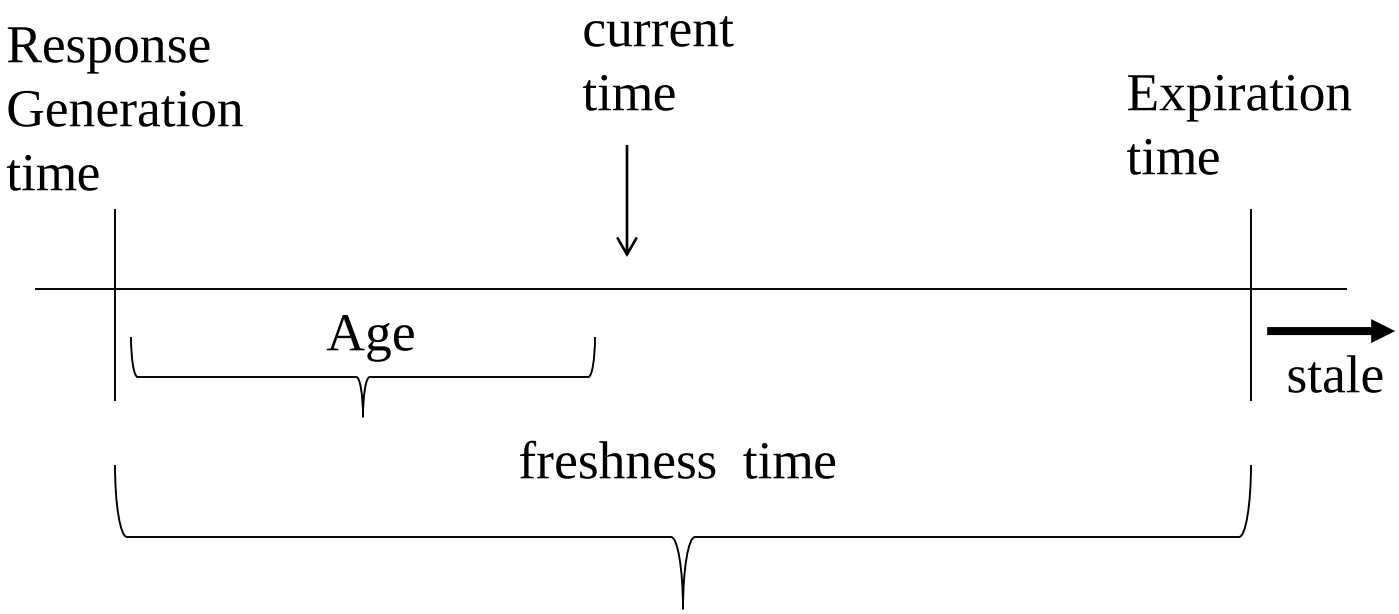
- HTTP/1.0 provided cache management using conditional GET using If-Modified-Since.
  - Response is 304 (Not Modified) or new data
- One would make a use this in conjunction with the Last-Modified header of the cached response.

- This did not work well for several reasons:
  - Only 1 second time resolution in conditional GET
  - Clocks of hosts on the Internet typically have large time-skew.
  - Incorrect caching some responses which should never be cached.

Cache Control

- HTTP/1.1 protocol provides several cache control primitives as part of request and response messages.
- It specifies how proxies should behave under different conditions.
- A proxy can cache data if permitted.
- Cache data may be private or public.
- A cache response has “age”:
  - *Age* is current time minus the time when the response was generated. (expressed in seconds)
  - A response may specify “max-age” or expiration time.

Age, freshness lifetime



Cache Control in Request Message

- Cache-Control is a general header (See Section 14.9 of RFC 2616)

- A request may contain the following values for this field
  - no-cache (do not serve from cache but send this to origin server)
  - no-store (do not store this data in caches – security sensitive)
  - max-age (age of response does not exceed this value)
  - max-stale
  - min-fresh (to specify the remaining freshness value)
  - only-if-cached

Cache Control in Response Message

- A response may contain the following values for the Cache-Control field
  - public
  - private (cache for some specific user)
  - no-cache (do not cache this response)
  - no-store (do not store, security sensitive)
  - no-transform (do not change encoding in transfer)
  - must-revalidate
  - max-age=seconds (cache validity life time)
    - Overrides Expiration time if any given in the response
  - smax-age max age for shared cache

Caching and Invalidation

- Certain kinds of responses MAY be cached subject to caching constraints (Section 13.4 of RFC 2616)
  - Responses with status 200, 203, 206, 300, 301 , 410

- Certain kinds of requests would cause any cached copy of the accessed resource to be invalidated (Section 13.10)
  - PUT
  - DELETE
  - POST

Etag Validator

- Timestamp are not strong validators:
  - Comparison may lead to errors due to 1 second resolution.
- HTTP/1.1 supports the notion *entity tag* (Etag)
- If two responses have the same Etag value then they must be identical.
- Typical Etag may be based on hash-code or fine-grained timestamps

Condition GET for Validation

- If-Modified-Since was supported by HTTP/1.0
- HTTP/1.1 provides several additional mechanisms for conditional GET

If-None-Match client presents one or more Etags

If-Match one or more Etags

Cache Revalidation and Reload

- See Section 14.9.4 of RFC 2616
- End-to-end Reload
  - Use “no-cache” in Cache-Control in a Request
- Specific end-to-end
  - Set max-age=0 in a request along with a conditional validator
  - This forces each proxy to validate with the next up-stream
- Non-specific end-to-end
  - Max-age=0 with no conditional validator specified.
- In validating a request, using conditional GET with If-Modified-since or If-None-Match, when 304 response is received, some of the headers in the cache may need to be updated.
- Weak vs. Strong Validators (Section 13.3.3)
  - Modification Time stamps are weak because of 1 second granularity

More on caching guidelines

- In a request, GET, POST, HEAD are case-sensitive.
  - If-Modified-Since can be use for cache validation.
- Applications must not cache POST requests.
- Response status 304 is returned on conditional get when the document is not modified. Any new header information in this response should update its cached entity. (Section 9.3).
- Responses to requests containing “Authorization” request header are not cacheable. Section 10.2.

HTTP 1.1 Protocol

- If "Pragma: no-cache" is specified by the client, the proxy server should pass the request to the origin server. This allows a client to get authoritative response if it knows that the cache contents are stale.

Authorization

- HTTP/1.0 mechanisms based on “Basic” authentication scheme is insecure.
  - Passwords are unencrypted

- HTTP/1.1 provides “Digest Access Authentication: which can include appropriate cryptographic certificates to be used as challenge-response based authentication.

Cookies

Cookies

- HTTP Protocol is stateless.
- To establish a relationship between a client's current request with any of the previous requests, cookies are used.
- Cookies are name=value pair of strings.
- Cookies are sent from the server to client. The client resends the cookies back to the server in any of the subsequent requests.
- See RFC 2109 and RFC 2965 (Now obsolete)
- Current standard is RFC 6265

Set-Cookie in Response Header

- Set-Cookie: *NAME*=*VALUE*; expires=*DATE*; path=*PATH*; domain=*DOMAIN_NAME*; secure
- This information is stored by the browser.
- *NAME*=*VALUE*

  This string is a sequence of characters excluding semi-colon, comma and white space. If there is a need to place such data in the name or value, some encoding method such as URL style %XX encoding is recommended, though no encoding is defined or required. This is the only required attribute on the Set-Cookie header.

Set-Cookie…default actions

Action by the browser when the following are missing in a Set-Cookie response:

Domain -- default to use request-host

Path – default to path in request-URI

Expires is missing, the browser will discard the cookie when it exits.

Set-Cookie…

- expires=DATE
- The expires attribute specifies a date string that defines the valid life time of that cookie. Once the expiration date has been reached, the cookie will no longer be stored or given out.
- The date string is formatted as:

- Wdy, DD-Mon-YYYY HH:MM:SS GMT

Set-Cookie…

- **path**=*PATH*
  - The **path** attribute is used to specify the subset of URLs in a domain for which the cookie is valid. If a cookie has already passed **domain** matching, then the pathname component of the URL is compared with the path attribute, and if there is a match, the cookie is considered valid and is sent along with the URL request. The path "/foo" would match "/foobar" and "/foo/bar.html".
  - The path "/" is the most general path. If the **path** is not specified, it as assumed to be the same path as the document being described by the header which contains the cookie.

Rule for accepting Set-Cookie

Reject the cookie under any of these cases:

- Path attribute in Set-Cookie is not a prefix of the request URI
- If the Domain attribute contains no embedded dot.

  Example: Reject when D= ".com"
- Reject when Domain attribute is a "public suffix"

  Example: Reject when D= "co.uk"
- If the host name in request does not domain-match the Domain attribute in the cookie:

  x.y.com domain matches y.com

  Cookie from x.y.acme.com with Domain=".foo.com" will be rejected but similar cookie from y.acme.com will be accepted.
- A cookie sent by foo.example.com will accepted or rejected as given by the following example:

  Accept when D="example.com" or "foo.example.com"

  Reject when D="bar.example.com"

Sending Cookies

- When searching the cookie list for valid cookies, a comparison of the **domain** attributes of the cookie is made with the Internet domain name of the host from which the URL will be fetched.
- If there is a tail match, then the cookie will go through **path** matching to see if it should be sent.
- "Tail matching" means that **domain** attribute is matched against the tail of the fully qualified domain name of the host. A **domain** attribute of “.acme.com" would match host names "anvil.acme.com" as well as "shipping.crate.acme.com".
- The default value of **domain** is the host name of the server which generated the cookie response.

Cookie in a Request Header

- A request may contain one or more header like

Cookie: NAME1=STRING1; NAME2=STRING2 ...

The server will take the appropriate actions based on the cookies attached with a request.