Hypertext Transfer Protocol (HTTP)

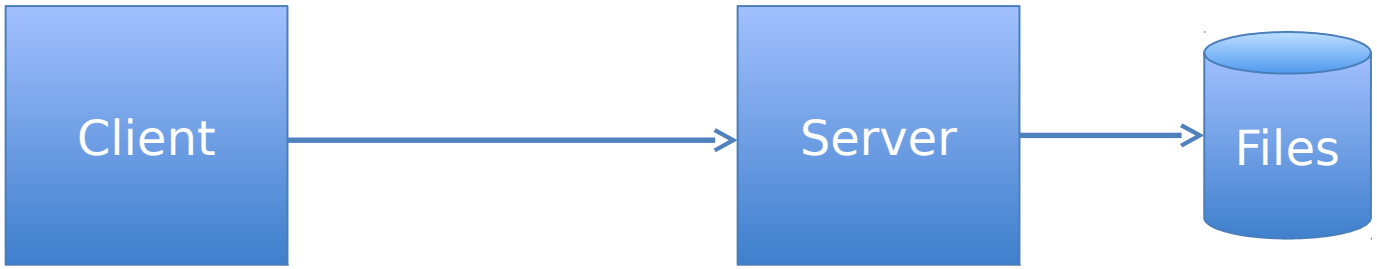
Devdatta Kulkarni

HTTP

- **H**yper**T**ext
  - Text and Binary data (images, doc files, pdf files, etc.)
- **T**ransfer
  - Act of moving the hypertext between client and server
- **P**rotocol
  - Set of agreed upon actions
- HTTP:
  - A protocol that defines how to move hypertext between client and server *over Internet*
- Over Internet – What does this mean?
  - IP address of the server is publicly accessible
  - The server process is running on port 80 (typically)

Problem

- What aspects do we need to consider when building a protocol that allows files of hypertext stored on a server to be requested by Client?



5 minute exercise

- Aspects to consider:
  - Standard way of identifying yourself
  - Integrity availability confidentiality of hypertext
  - How to support hypertext of different sizes?
  - Reliable communication between client and server
  - Uniform and standard way to represent hypertext
  - Identify specific resources and do actions on them
  - How to manage state?
  - How to handle failures?

Protocol design considerations

- Clients should be able to unambiguously request one file from another
- There should be a mechanism to differentiate good Client request from a bad request from a Client
- Files to serve may be large
- Clients should be able to request transfer of files only if they don’t already have them

Relevant HTTP mechanisms

- **URLs** to uniquely reference resources (files) on a server
- **Status codes** to distinguish good requests from bad requests
- **Chunking mechanism** to transfer large resources from the server to the client
- **Caching mechanisms** to conditionally request resources

HTTP

- Request-response protocol
- Where is it implemented?
  - Within browsers, web servers, proxy servers, load balancers, web application containers
- Version 1.0 published in 1996
  - http://www.rfc-base.org/txt/rfc-1945.txt
- Version 1.1 published in 1999
  - https://www.ietf.org/rfc/rfc2616.txt

HTTP

Client and Server interact with each other using HTTP

HTTP: Main concepts

- Resources
- Methods
- Protocol Operation
  - Request
  - Response

Resources

- A resource is an entity maintained by the web application running on the server
- Resources are organized in a hierarchy
  - Similar to file system hierarchy
- Examples:
  - "index.html" file when we visit a web page:
    - http://www.cs.utexas.edu/~devdatta/index.html
  - A repository stored on github.com
- A resource is identified by Uniform Resource Locator (URL)

HTTP URLs

http_url = "http:" "//”host[":”port]
[abs_path]

host: A legal Internet host domain name or IP address

port: 0 or more digits
if port is empty or not given, port 80 is assumed

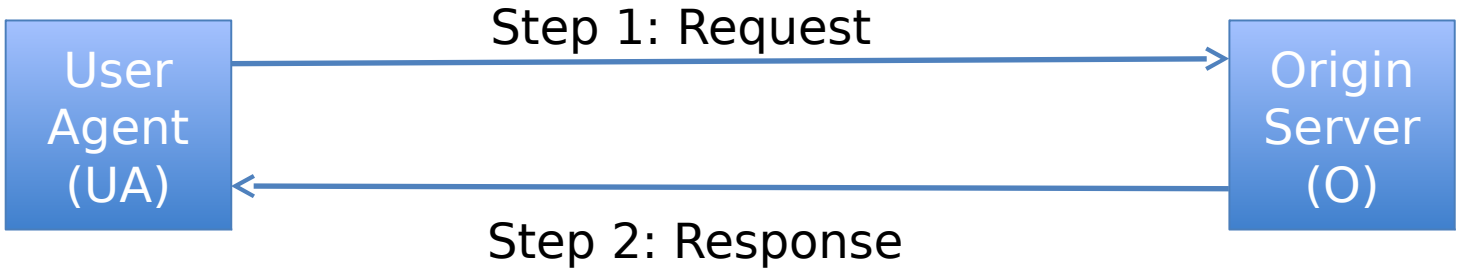
abs_path: path of resource on the server

Methods

- A method indicates what to do with a resource
  - Retrieve information
  - Update/delete information
  - Delete the resource
  - Create a new resource (by making a call to its parent resource)
- HTTP defines several methods which have pre-defined meaning. Example:
  - GET
    - Get information about a resource
  - POST
    - Create a resource
  - PUT
    - Modify information about a resource
  - DELETE
    - Delete a resource

Protocol Operation

Protocol Operation



Client/UA initiates a socket connection and
sends the request to the server

Server sends the response and closes the socket connection

Tools to interact with Web applications

- curl
- Chrome Apps and Extensions
  - Advanced Rest Client
  - Postman
- Firefox add-on
  - Poster
- Firefox built-in tools
  - Tools → Web Developer → Web Console → Network
- Chrome built-in tools
  - Tools → Developer tools → Network
- Java programs
  - TestClientSocket from https://github.com/devdattakulkarni/ModernWebApps/tree/master/HTTP

Examples

- curl -i http://www.cs.utexas.edu/~devdatta/test-file.txt

- curl -i http://www.cs.utexas.edu/~devdatta/test-file.txt \

-H "Host: www.cs.utexas.edu" \

-H "User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0) Gecko/20100101 Firefox/44.0" \

-H "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" \

-H "Accept-Language: en-US,en;q=0.5" \

-H "Accept-Encoding: gzip, deflate" \

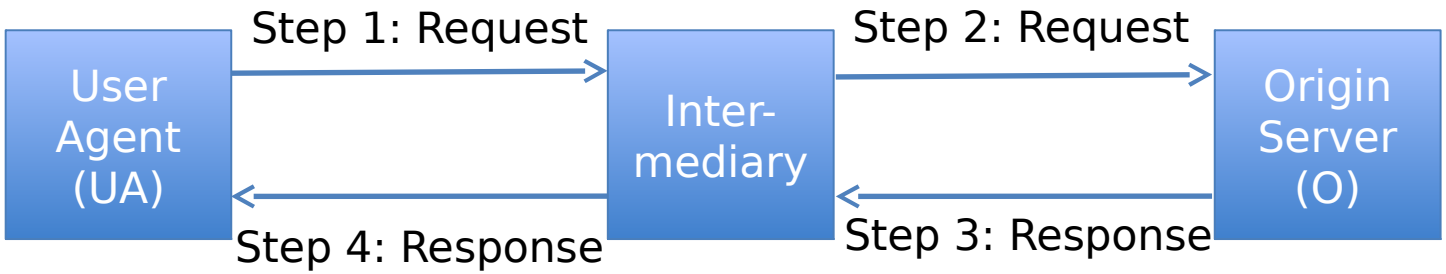
-H "Cookie: _ga=GA1.2.699921686.1471968877" \

-H "Connection: keep-alive" \

-H "If-Modified-Since: Tue, 30 Aug 2016 01:42:06 GMT" \

-H "If-None-Match: "17a1e7f-2-53b4016909fa7"" --compressed

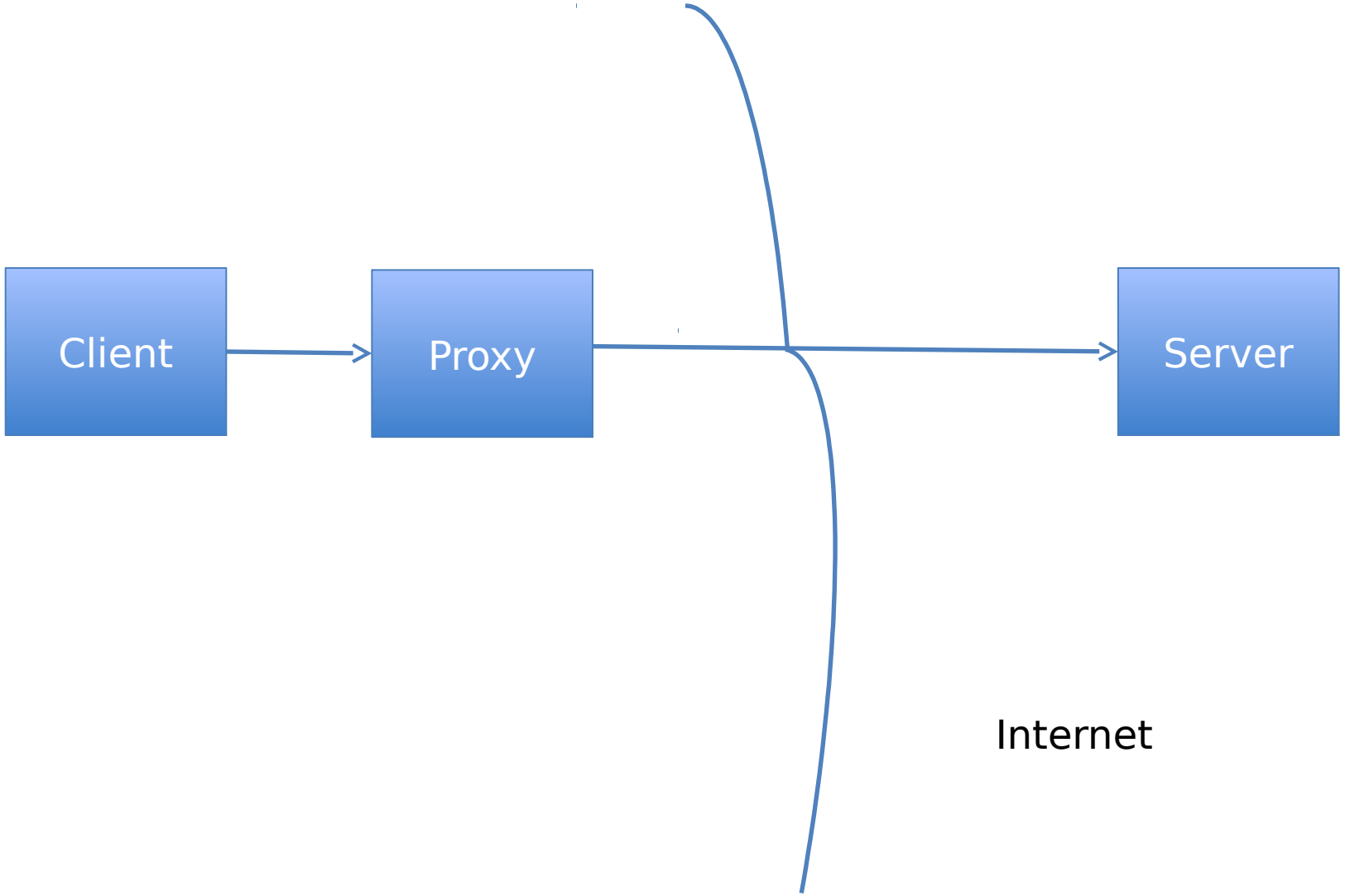
Protocol Operation



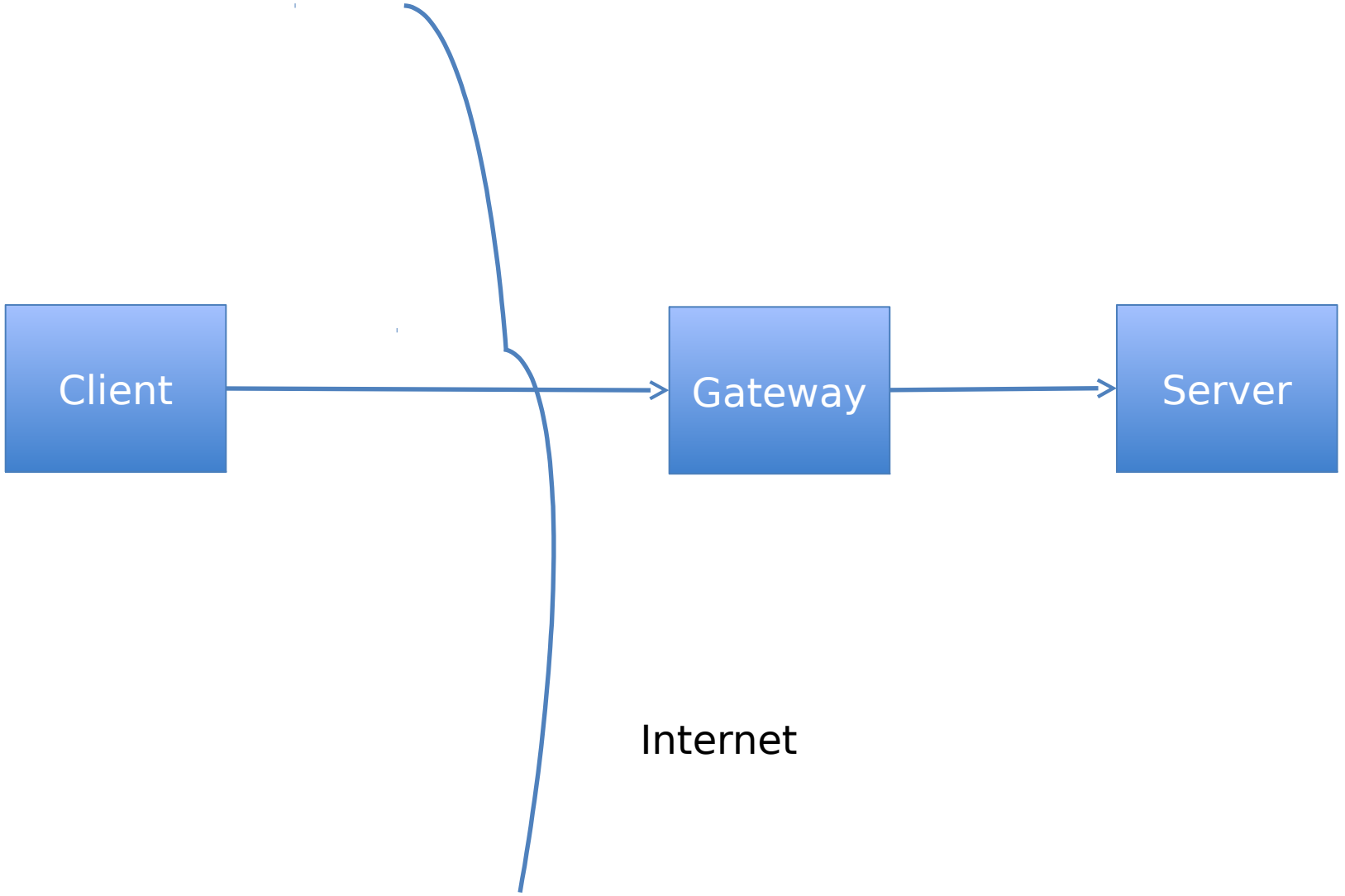
Intermediary: Proxy or Gateway

A Proxy or Gateway may cache responses

Proxy



Gateway / Reverse Proxy



Protocol communication

- HTTP needs reliable transport mechanism
  - TCP: Okay
  - SMTP: Okay
  - UDP: Not okay
- HTTP communication typically happens over port 80
- HTTPS (Secure HTTP)
  - Communication happens on port 443

HTTP Request

HTTP Request

A request consists of four components:

- Request-line
- Request Headers
- CRLF (Newline)
- Request Body

Request-Line

Method <space> URI <space> HTTP-Version CRLF

CRLF: Newline

E.g:
GET /~devdatta/index.html HTTP/1.1\n
Host: www.cs.utexas.edu

URI vs. URL

- URI: Uniform Resource Identifier
  - Identifier/Name for a resource
    - Could be path of the resource (such as file path)
- URL: Uniform Resource Locator
  - Network location of resource
    - Essentially, specifies how to access the resource

HTTP Request Headers

- Request headers allow the client to pass additional information about the request or the client itself to the server
- Some HTTP 1.0 Request Headers
  - Authorization
  - User-Agent
  - If-Modified-Since

Authorization Header

- A user agent can authenticate itself with the server by including the “Authorization” header

- HTTP supports
  - Basic Authentication
  - Digest Authentication

Basic Authentication

- “Basic” authentication scheme is based on the model that the user agent must authenticate itself with a user-ID and a password
- Client sends Base64 encoded string of userID:password
- Example:
  - Suppose username=‘Aladdin’ password=‘open sesame’
  - Client sends
    - Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
    - Example: java Base64EncodeDecode

Basic Authentication

- Not secure
  - username:password are ‘encoded’ not ‘encrypted’
- Works under the assumption that the transport layer between the client and the server is trusted

User-Agent Header

- The User-Agent request header contains information about the user agent originating the request

- Server can use this value for tailoring the response to be sent to the client
  - Sending images of different resolution to desktop browser vs. Android browser

HTTP Response

HTTP Response

A Response consists of four components:

- Status-Line
- Response headers
- CRLF (essentially a newline)
- Response Body

Response Status-Line

Status-Line =
- HTTP-Version <space> Status-Code <space> Reason-Phrase CRLF

- Status codes:
  - 3 digit integer
  - First digit defines the class of response
    - 1xx – Informational
    - 2xx – Success
    - 3xx – Redirection (further action must be taken)
    - 4xx – Client Error (bad syntax or cannot be fulfilled)
    - 5xx – Server Error

Example Status Codes

- http://eavesdrop.openstack.org/
- 200 – OK
  - java TestClientSocket eavesdrop.openstack.org / html
- 301 – Moved Permanently
  - java TestClientSocket eavesdrop.openstack.org /irclogs html
- 400 – Bad Request
  - java TestClientSocket eavesdrop.openstack.org index.html html
- 404 – Not found
  - java TestClientSocket eavesdrop.openstack.org /irclogs1 html
- 401 – Unauthorized
- 500 – Internal Server Error

Things that client care

- Timestamp
  - When the data was last modified
- Till when the data is valid per the timestamp
- Data format
- How large the response is?
- If the client has the authorization
  - Determines if the Authorization header needs to be included

HTTP 1.0 Response Headers

- Date
  - Indicates the date/time when the message originated
- Last-Modified
  - Date/time when the resource was last modified
    - For files, it may be just the file system last-modified time.
      - How to deal with a group of resources?
- Invariant:
  - *Date value is always greater than value of Last-Modified header*
- Content-Length
  - Indicates the size of the Entity-Body (response that contains entity)
  - The value is decimal
    - Example: java TestClientSocket www.cs.utexas.edu /~devdatta/test-file.txt txt
      - Show Content-Length
      - Show serveroutput.txt
- Content-Type
  - Indicates the media type of the Entity-Body

HTTP 1.1: Chunked transfer encoding

- Transfer Coding:
  - Encoding transformation that has been applied to an entity-body
  - Chunked transfer coding
    - transfer-coding: chunked
    - Chunked-body = *chunk
      last-chunk
      trailer
      CRLF

chunk = chunk-size CRLF chunk-data CRLF
chunk-size = string of HEX digits
last-chunk has chunk-size “0”

HTTP 1.1: Chunked transfer encoding

- Example:
  - Step 1: java TestURLConnection http://eavesdrop.openstack.org/irclogs >& output.txt
  - Step 2: Remove headers from output.txt
  - Step 3: java TestClientSocket eavesdrop.openstack.org /irclogs/ html >& output1.txt
  - Step 4: http://www.binaryhexconverter.com/hex-to-decimal-converter
  - Step 5: Check the results

HTTP Caching

HTTP Caching

- Goal
  - From client side:
    - Eliminate the need to send requests to reduce the number of ‘round-trips’ required for many operations
  - From server side:
    - Eliminate the need to send full responses to reduce the ‘network bandwidth’ requirements

HTTP Caching

- Both HTTP/1.0 and HTTP/1.1 support caching
- Caching is managed via request and response headers

Caching: Overall Operation

- User Agent (UA) requests a resource
- Server sends the resource along with some caching related headers
- An intermediate Cache/Proxy_server saves the resource and the headers
- Cache/Proxy_server applies a "caching algorithm" to:
  - Satisfy *subsequent requests* from the UA

HTTP 1.0: Caching Headers

- Expires (Response header)
  - Date/time after which the resource should be considered stale
  - Applications/Cache/Proxies must not cache the entity beyond the given date
  - If the value is equal to or earlier than the value of the ‘Date’ header then the entity should not be cached
- If the Expires header is absent then the response should not be cached
- If-Modified-Since (Request header)
  - Used with the GET method to make it conditional
    - If the requested resource has not been modified since the time specified in the field, a copy of the resource will not be returned from the server; instead a ‘304’ status will be returned

If-Modified-Since Header

- Which date to use in the If-Modified-Since header?
  - When the resource was first requested, server should have sent either the “Date” header or the “Last-Modified” header in the response
  - That date should be used with the “If-Modified-Since” header
  - If both are present, use the Last-Modified header

- Example:
  - Open http://www.openstack.org in
    - Chrome->View->Developer->Developer Tools
  - Show request headers for “clear.png”

HTTP 1.0: Caching Headers

- Pragma: no-cache (Request header)
  - An intermediary should forward the request toward the origin server even if it has a cached copy of what is being requested
  - Client's way to tell an intermediary to not serve the response from the cache

Problems with HTTP 1.0 Caching

- Too tied with absolute dates and times
  - May not work appropriately if the clocks on the server and client are not synchronized
- Not fine-grained enough
  - Cannot work if a common cache is present for multiple different clients
    - “Shared” cache

Problems with HTTP 1.0 Caching

- Too tied with absolute dates and times
  - May not work appropriately if the clocks on the server and client are not synchronized
- Not fine-grained enough
  - Cannot work if a common cache is present for multiple different clients
    - “Shared” cache

HTTP 1.1: Caching Headers

- Cache-control
- Entity-tag cache validators

HTTP 1.1: Cache-Control Headers

- The Cache-Control header allows a client or server to transmit a variety of directives in either requests or responses.
  - These directives typically override the default caching algorithms
  - Must be obeyed by all caching mechanisms along the request/response chain
- Cache-Control Headers:
  - max-age: Servers specify explicit expiration times using either the Expires header or the ‘max-age’ directive
    - If both are present then the ‘max-age’ directive takes the precedence
  - s-maxage: For “shared” cache the specified maximum age overrides max-age and Expires header
- Cache-Control: max-age=0
  - Client’s way to force a recheck with the origin server by an intermediate cache
  - This is same as the ‘Pragma: no-cache’ header of HTTP 1.0

HTTP 1.1: Entity Tag Cache Validators

- What is an Entity Tag?
  - Think of an Entity Tag as a “signature” of the resource
- Its purpose is to validate that the cached entity responses are good enough to be used
- Two kinds of validators
  - Strong validators and Weak validators
  - Strong validators
    - A server changes the validator for every change in entity
  - Weak validator
    - A server changes the validator only for semantically significant changes to an entity
- Example
  - Web application for support tickets; Changing ‘ticket name’ from lower case to mix case may not trigger this; but addition of new email address to ticket’s support account attribute will

- It is up to the server when to calculate the entity tag
  - Strong validator corresponds to calculating the entity tag / signature on every change of the resource
  - Weak validator corresponds to calculating the entity tag / signature on “major” changes to an entity

Entity Tag Cache Validators

- ETag (Response header):
  - The ETag response-header field provides the current value of the entity tag for the requested resource

- How can client use the value read from the ETag header?
  - Client sends the value back to the server and may want the server to perform an action only if:
    - The current value of ETag *does not match* one in the request header
    - The current value of Etag *does match* in the request header

- If-None-Match (Request header):
  - Perform the action only if the value specified in the header *does not match* that of the Etag of that resource on the server
    - Show example with GET on
      - http://www.openstack.org/themes/openstack/images/auto-banner.jpg
      - Use Chrome Developer Tool and Firefox RESTClient to see difference in behavior

Entity Tag Cache Validator

- If-None-Match (Request header):
  - Works in tandem with If-Modified-Since header
  - If the Entity tag has not been modified on the server, it may still perform the specified action if the resource’s modification date fails to match that specified in the If-Modified-Since header

Entity Tag Cache Validators

- If-Match (Request header):
  - Perform the action only if the value specified in the header *matches* that of the Etag of that resource on the server
  - Used as a ‘precondition’ mechanism for performing a resource modification action
  - Server returns ‘412’ precondition failed

Persistent Connections

Persistent Connections

- HTTP is a request response protocol
  - Client establishes the connection; server sends the response and closes the connection
  - For next request a new connection is opened
- Problem
  - Opening up a new socket connection for each request is a slow process
- Solution is to use “persistent connections”

- Both HTTP 1.0 and HTTP 1.1 support persistent connections
- HTTP 1.0
  - Persistent connections need to be explicitly negotiated
    - Connection: Keep-Alive
- HTTP 1.1
  - Persistent connections by default
    - Server will maintain a persistent connection with the client
      - Unless ‘Connection: close’ header is sent in the request
    - A single user client SHOULD NOT maintain more than 2 connections with a server

Persistent Connections

- Advantages:
  - Lower CPU and memory consumption on the client and the server
  - Lower latency to subsequent requests
- Disadvantages:
  - Keeping connection open may exhaust server socket resources
    - If appropriate timeouts are not used this can become a cause denial-of-service attack on the server

Summary

HTTP 1.0 vs. HTTP 1.1

- Caching
  - HTTP 1.0
    - Has limited number of cache control headers
      - Date, Last-Modified, If-Modified-Since, no-cache
  - HTTP 1.1
    - Additionally has
      - Etag, If-None-Match, If-Match, max-age
- Persistent connections
  - HTTP 1.0
    - Need to explicitly negotiated using “Connection: Keep-alive” header
  - HTTP 1.1
    - By default
- *Host* request header
  - HTTP 1.0 – not required
  - HTTP 1.1 - required

HTTP Request Examples

- HTTP 1.0

  GET /irclogs/ HTTP/1.0

- HTTP 1.1

  GET /irclogs/ HTTP/1.1

  Host: <host-name>

  Host header required for HTTP 1.1

Reference

- HTTP 1.0
  - http://www.rfc-base.org/txt/rfc-1945.txt
- HTTP 1.1
  - https://www.ietf.org/rfc/rfc2068.txt
  - https://www.ietf.org/rfc/rfc2616.txt
- Differences between HTTP 1.0 and HTTP 1.1
  - http://www8.org/w8-papers/5c-protocols/key/key.html

Additional Material

Authentication vs. Authorization

- Authentication:
  - Server is trying to answer the question, “Do I know this client?”

- Authorization:
  - Server is trying to answer the question, “Does this client have permission to perform this method on this resource?”
  - Authentication information can be used to perform authorization
    - If a client can correctly authenticate then it can access the specified resource

Referer Header

- Referer:
  - The address (URI) of the resource from which the Request-URI was obtained
    - Show example:
      - Open www.openstack.org and show in Chrome Developer tool
    - Useful for building the ‘back pointer’ graph