LAB 4 HTTP (HyperText Transfer Protocol)

CMPE 208: NETWORK ARCHITECTURE AND PROTOCOL

Prof. Shai Silberman

Submitted by

Team 12

Team Members

Akash Kumar Athghara	Kshama Shalini	Gunveet Singh Arora	Sashank Malladi
akash.athghara@sjsu.edu	kshama.shalini@sjsu.edu	gunveetsingh.arora@sjsu.edu	sashank.malladi@sjsu.edu
010757929	010763792	010641904	010466651

Contribution by each team member

Akash Kumar Athghara: Payload Format, URL encoding, Security and Conclusion
Gunveet Singh Arora: Lab setup, Header fields, caching and Learning outcomes
Kshama Shalini: Cover page, Contribution of team members, overview, parameters, messages
Sashank Malladi: Request, Response, Methods, Status codes

Overview

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.
Basically, HTTP is a TCP/IP based communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

There are three basic features that make HTTP a simple but powerful protocol:

- **HTTP is connectionless**: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.
- **HTTP is media independent**: It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.
- **HTTP is stateless**: As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

Lab Setup

Wireshark: In this lab we are using a free and open-source packet analyzer Wireshark to capture and examine a packet trace. It is a software used for network troubleshooting, analysis, software and communications protocol development, and education. A packet capture includes time stamped to every packet. Wireshark provides a graphical UI that helps us to capture the sequence of packets and understand the bit operation. It color-codes packets by their type, and has an inbuilt feature to filter and analyze packets to investigate the behavior of network protocols

The first step would be to install Wireshark on our system and then use the software to implement the http commands. In http we have commands such as get, post, put etc. Down shown below is the packet capture packet for http. Here we open any website in the browser and to check the corresponding http capture we put http command in the filter option in Wireshark software. HTTP allows for communication between a variety of hosts and clients, and supports a mixture of network configurations. Also, HTTP is a stateless protocol.

So the main commands we get in the HTTP command line are:

- **GET**: for *fetching* an existing resource. The URL contains all the necessary information the server needs to locate and return the resource.
- **POST**: for *creating* a new resource. POST requests usually carry a payload that specifies the data for the new resource.
- **PUT**: for *updating* an existing resource. The payload may contain the updated data for the resource.
- **DELETE**: for *deleting* an existing resource

We also have some of the lesser known verbs used by the HTTP. These are:

- **HEAD**: this is similar to GET, but without the message body. It's used to retrieve the server headers for a particular resource, generally to check if the resource has changed, via timestamps.
- **TRACE**: used to retrieve the hops that a request takes to round trip from the server. Each intermediate proxy or gateway would inject its IP or DNS name into the Via header field. This can be used for diagnostic purposes.
- **OPTIONS**: used to retrieve the server capabilities. On the client-side, it can be used to modify the request based on what the server can support.

Parameters

This chapter is going to list down few of the important HTTP Protocol Parameters and their syntax the way they are used in the communication. For example, format for date, format of URL, etc. This will help you in constructing your request and response messages while writing HTTP client or server programs. You will see the complete usage of these parameters in subsequent chapters while learning the message structure for HTTP requests and responses.

HTTP Version

HTTP uses a **<major>.<minor>** numbering scheme to indicate versions of the protocol. The version of an HTTP message is indicated by an HTTP-Version field in the first line. Here is the general syntax of specifying HTTP version number:

```
HTTP-Version   = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

Example

```
HTTP/1.0
```

or

```
HTTP/1.1
```

Uniform Resource Identifiers

Uniform Resource Identifiers (URI) are simply formatted, case-insensitive string containing name, location, etc. to identify a resource, for example, a website, a web service, etc. A general syntax of URI used for HTTP is as follows:

```
URI = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

Here if the **port** is empty or not given, port 80 is assumed for HTTP and an empty **abs_path** is equivalent to an **abs_path** of "/". The characters other than those in the **reserved** and **unsafe** sets are equivalent to their ""%" HEX HEX" encoding.

Example

The following three URIs are equivalent:
http://abc.com:80/~smith/home.html
http://ABC.com/%7Esmith/home.html
http://ABC.com:/%7esmith/home.html

Date/Time Formats

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. HTTP applications are allowed to use any of the following three representations of date/time stamps:
Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov 6 08:49:37 1994 ; ANSI C's asctime() format

Character Sets

We use character sets to specify the character sets that the client prefers. Multiple character sets can be listed separated by commas. If a value is not specified, the default is the US-ASCII.
Example
Following are the valid character sets:
US-ASCII
or
ISO-8859-1
or
ISO-8859-7

Content Encodings

A content encoding value indicates that an encoding algorithm has been used to encode the content before passing it over the network. Content coding are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity.
All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding and Content-Encoding header fields which we will see in the subsequent chapters.
Example
Following are the valid encoding schemes:
Accept-encoding: gzip
or
Accept-encoding: compress
or
Accept-encoding: deflate

Media Types

HTTP uses Internet Media Types in the **Content-Type** and **Accept** header fields in order to provide open and extensible data typing and type negotiation. All the Media-type values are registered with the Internet Assigned Number Authority (IANA). The general syntax to specify media type is as follows:
media-type = type "/" subtype *(";" parameter)
The type, subtype, and parameter attribute names are case--insensitive.
Example
Accept: image/gif

Language Tags

HTTP uses language tags within the **Accept-Language** and **Content-Language** fields. A language tag is

composed of one or more parts: a primary language tag and a possibly empty series of subtags:
language-tag = primary-tag *("-" subtag)
White spaces are not allowed within the tag and all tags are case- insensitive.
Example
Example tags include:
en, en-US, en-cockney, i-cherokee, x-pig-latin
where any two-letter primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-3166 country code.

Messages:

HTTP is based on the client-server architecture model and a stateless request/response protocol that operates by exchanging messages across a reliable TCP/IP connection.

An HTTP "client" is a program (Web browser or any other client) that establishes a connection to a server for the purpose of sending one or more HTTP request messages. An HTTP "server" is a program (generally a web server like Apache Web Server or Internet Information Services IIS, etc.) that accepts connections in order to serve HTTP requests by sending HTTP response messages.

HTTP makes use of the Uniform Resource Identifier (URI) to identify a given resource and to establish a connection. Once the connection is established, **HTTP messages** are passed in a format similar to that used by the Internet mail [RFC5322] and the Multipurpose Internet Mail Extensions (MIME) [RFC2045]. These messages include **requests** from client to server and **responses** from server to client which will have the following format:

HTTP-message = <Request> | <Response> ; HTTP/1.1 messages

HTTP requests and HTTP responses use a generic message format of RFC 822 for transferring the required data. This generic message format consists of the following four items.

A Start-line
Zero or more header fields followed by CRLF
An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
Optionally a message-body

In the following sections, we will explain each of the entities used in an HTTP message.

Message Start-Line

A start-line will have the following generic syntax:
start-line = Request-Line | Status-Line

Examples of start line in case of request and response:
GET /hello.htm HTTP/1.1 (This is Request-Line sent by the client)
HTTP/1.1 200 OK (This is Status-Line sent by the server)

Header Fields

HTTP header fields provide required information about the request or response, or about the object sent in the message body. There are four types of HTTP message headers:

- **General-header:** These header fields have general applicability for both request and response messages.
- **Request-header:** These header fields have applicability only for request messages.
- **Response-header:** These header fields have applicability only for response messages.
- **Entity-header:** These header fields define meta information about the entity-body or, if nobody is present, about the resource identified by the request.

All the above mentioned headers follow the same generic format and each of the header field consists of a name followed by a colon (**:**) and the field value as follows:

message-header = field-name ":" [field-value]

Following are the examples of various header fields:

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

Message Body

The message body part is optional for an HTTP message but if it is available, then it is used to carry the entity-body associated with the request or response. If entity body is associated, then usually **Content-Type** and **Content-Length** headers lines specify the nature of the body associated.
A message body is the one which carries the actual HTTP request data (including form data and uploaded, etc.) and HTTP response data from the server (including files, images, etc.). Shown below is the simple content of a message body:

```
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

HTTP Signaling

As discussed, HTTP is stateless protocol based on client-server architecture. It operates by exchanging

messages through a reliable TCP/IP connection. HTTP client can be considered as an application that establishes connection with server by sending one or more http request messages. And HTTP server is a program which accepts connection from a client serving requests through response messages. Let us observe client's request and server's response messages in detail as follows.

HTTP Request Messages

A client sends a HTTP request initially to server for a response. The request message includes follow structure or parameters.

- A Request-line
- Zero or more header (General | Request | Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF)
- indicating the end of the header fields
- Optionally a message-body

Request Line

Request line starts with a method name and then followed by URI request and protocol version. Each element is separated by space character. The following is the format for the request line

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

Request Method

The request method field specifies a method to be performed on the resource mentioned in URI. The method field is case sensitive and should always be mentioned in uppercase.

Request URI

The request URI is a field used for resource identifier. The following is the format for specifying Request URI

```
Request-URI = "*" | absoluteURI | abs_path | authority
```

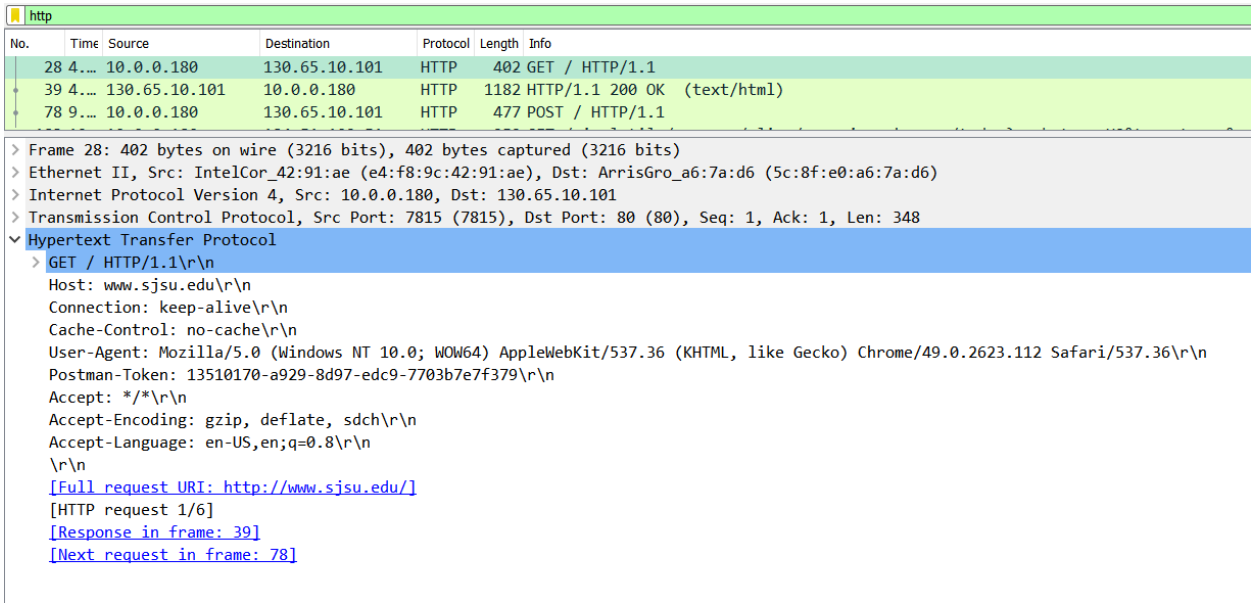
Only asterisk symbol is used when the request corresponds to server but not to a particular resource.

If a proxy is used to retrieve a resource, the concept of absoluteURI can be used. When absoluteURI, proxy is requested a service is provided from the cache available.

The most common option used is to identify resource directly from origin server. In this case, the client requests connection directly to the origin server of the resource. An absolute path should be specified in requesting a service. If none is present in the as part of URI, then it should be redirected to root folder of the server.

Request Header Fields

To pass any additional information about the request, then the client adds few header fields along the request



Above is the capture for HTTP request message in Wireshark. It is a request message for SJSU website. We can observe following points.

HTTP - Version : 1.1

Method: Get

Following points are observed from header fields in the above packet.

- It uses persistent connection so keep-alive is utilized.
- Cache is set to no-cache so, a cache must not be used as response to satisfy a subsequent request without successful revalidation with the origin server.
- Appropriate user agent is specified.
- Accept-Encoding restricts the content-coding to few that are acceptable in the response.

HTTP Response Messages

After receiving a request from a client, the server starts responding response messages. The following are few parameters included in a response messages.

- A Status-line
- Zero or more header (General |Response |Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF)
- indicating the end of the header fields
- Optionally a message-body

Status Line

The status line includes version of HTTP protocol involved and a status code along its textual description. There are several status codes described for this protocol. Each field in the status line are separated using a space character.

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

A server supporting version 1.1 responds version as follows HTTP/1.1

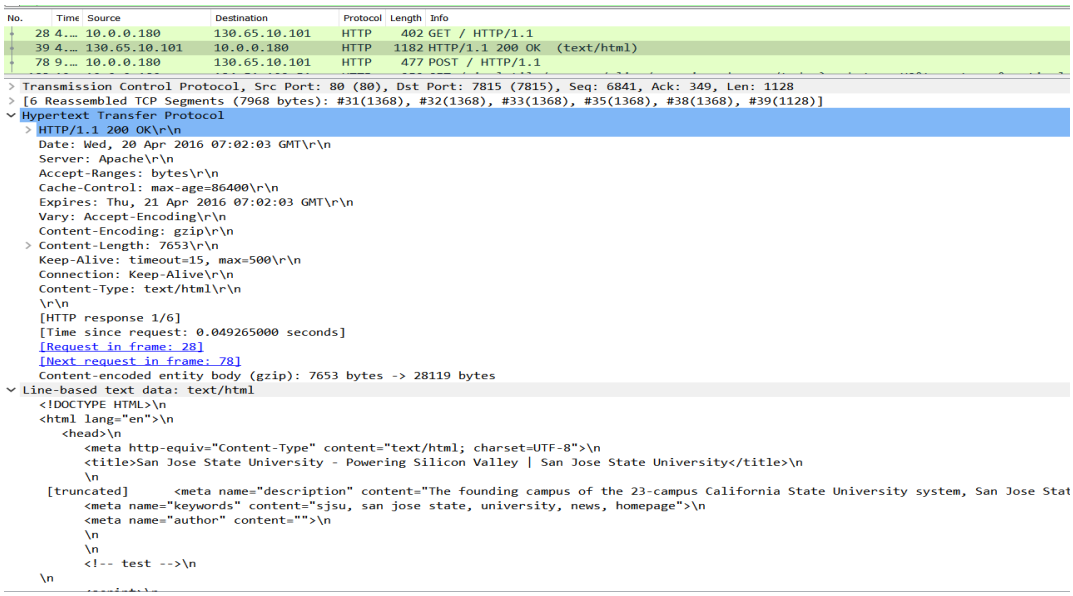
Response Header fields

The server passes additional information about response in the header fields if it cannot accommodate in the status line. These header fields include information such as parameters required by client in future access. There is also a provision to include new header fields when designing custom server and client. The following shows or describes a sample for header field format.

```
response-header = Accept-Ranges
        | Age
        | ETag
        | Location
        | Proxy-Authenticate
        | Retry-After
        | Server
        | Vary
        | WWW-Authenticate
```

Some of the headers are described as follows

- Age is considered as time in seconds since the message was created on the server.
- ETag is used to check any modifications are involved.
- Location is used when any redirection is involved due to new URL.
- Server identifies the server generating response message.



Above is the capture for HTTP response message in Wireshark. It is a response message for SJSU website. We can observe following points.

HTTP - Version : 1.1

Status: 200 (OK)

Following points are observed from header fields in the above packet.

- The server accepts byte range requests.
- Cache – control specifies that client is willing to accept a response whose age is not greater than 86400 seconds.
- Expires specify cookie expire time.
- Content encoding used is gzip.

HTTP Methods

Http provides several methods to implement desired action on identified resource. The resource can considered as a data file or any executable file residing on a server. The following are few common methods provided by HTTP/1.1. New custom methods may also be declared based on requirement. The method names are case sensitive and are used in uppercase.

Safe and Idempotent Methods

- **Safe Methods**
  These methods can be categorized in to methods where users should be in aware of their actions which may lead to unexpected results.
  GET and HEAD methods have significance only on retrieval of data and so they are considered as safe.
  In case of several other methods, such as POST, DELETE and PUT users should be made aware because of the fact that there is a possibility of unsafe action.
- **Idempotent Methods**

There is a possibility of side effects being generated due to identical requests. Methods such as GET, HEAD, PUT and DELETE share this property. Methods such as OTIONS and TRACE will generate no side effects.
There is a possibility that a sequence is considered to be non-idempotent if its result depends on a value that is later modified in the same sequence. However, by definition a sequence is considered as idempotent if they generate no side effects.

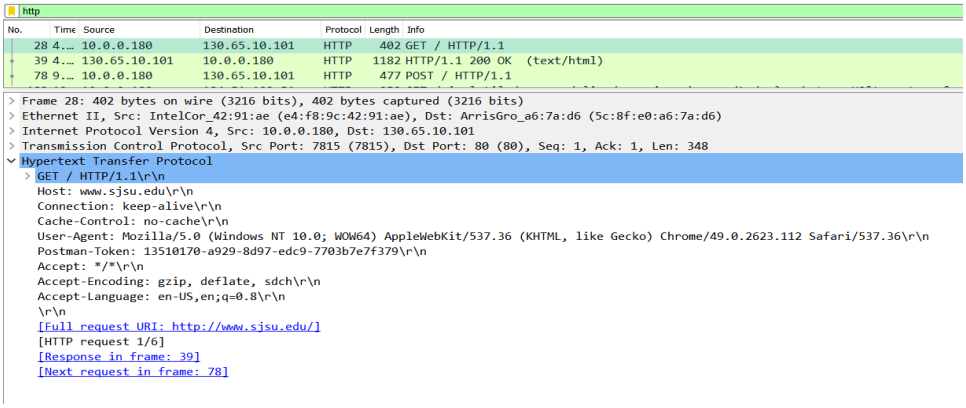
GET Method

A GET method retrieves data from web server by specifying parameters in the URL portion of the request. It retrieves directly any information found at location specified by URI. The get methods is categorized into two types. Conditional GET, normal GET.

If a header includes specifications such as If- Modified, If-Match, If-Range, then the entity of data is transferred based of matching conditions. The conditional GET method is very much useful in reducing unnecessary network usage. For example, only partial data is retrieved based on condition specified in range header field.

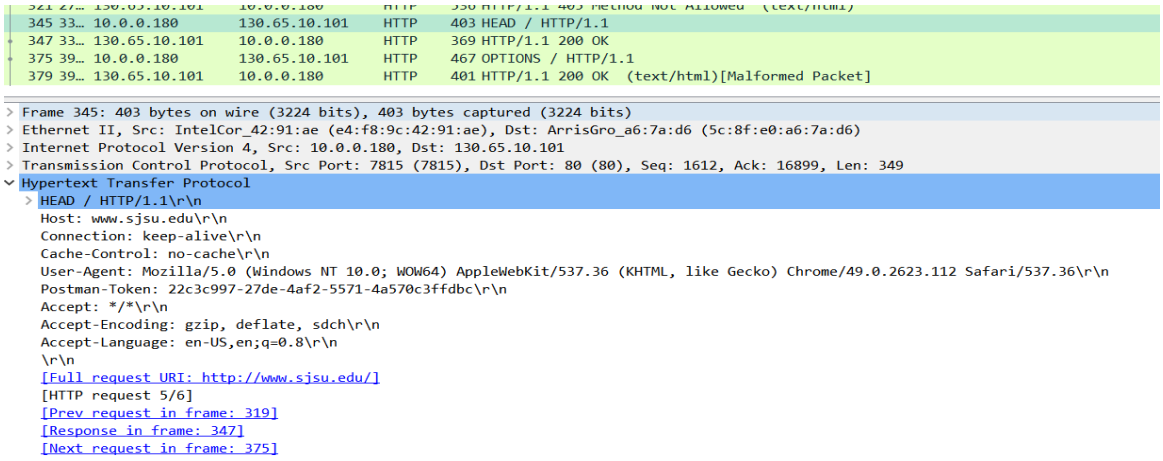
The response to the GET request can be cached if it suits a criteria.

Following is a Wireshark capture for GET method while requesting

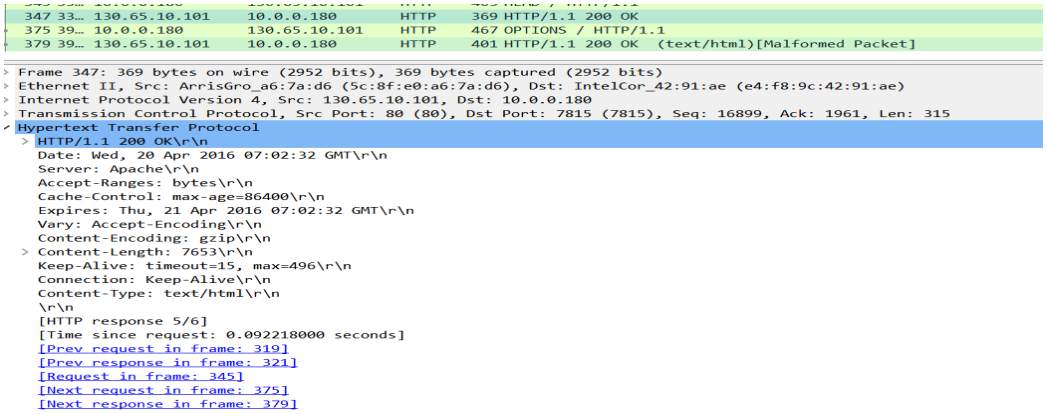


HEAD Method

This method is used to receive metadata of the resource requested on the server. This method is identical to GET method except that it doesn't revert with body of the message form the resource. This method is often used to test hyperlinks for their validity. The response of the HEAD method can be cached. If there is any updated in new data that is received from this request, the old cached data can be updated with the newly retrieved parameters. Following is a Wireshark capture for HEAD method while requesting



The request line clearly shows that it is using HEAD method. So for head method following is the response retrieved. The Response contains only metadata.



POST Method

The POST method is used when there is data that should be transferred to server. For example, a file update can be done based data received from the request, etc.

POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

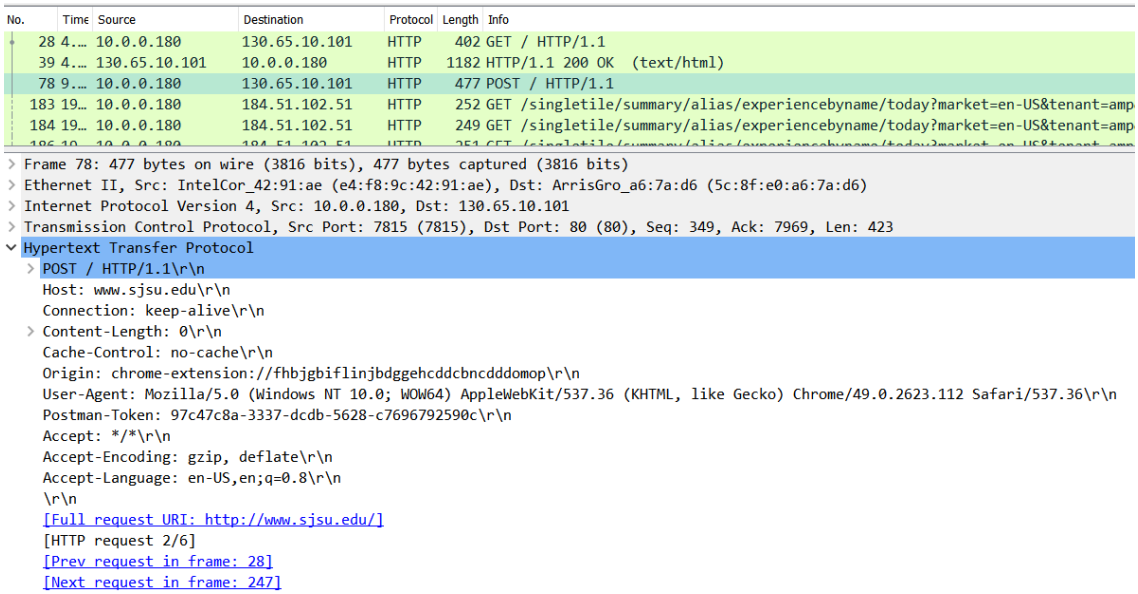
The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database.

The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status, depending on whether or not the response includes an entity that describes the result.

If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header.

The response of this method can be cached only if response includes any appropriate Cache-Control fields.

Following is a Wireshark capture for POST method from client



PUT Method

The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server.

If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response.

If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem.

Responses to this method are not cacheable.

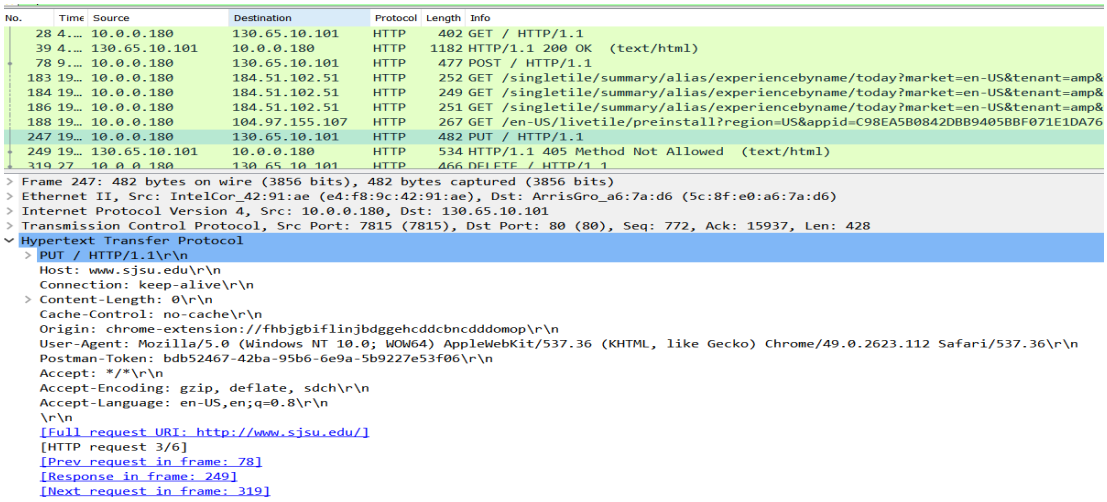
The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI. The URI in a POST request identifies the resource that will handle the enclosed entity. That resource might be a data-accepting process, a gateway to some other protocol,

or a separate entity that accepts annotations. In contrast, the URI in a PUT request identifies the entity enclosed with the request -- the user agent knows what URI is intended and the server MUST NOT attempt to apply the request to some other resource. If the server desires that the request be applied to a different URI, it MUST send a 301 (Moved Permanently) response; the user agent MAY then make its own decision regarding whether or not to redirect the request.

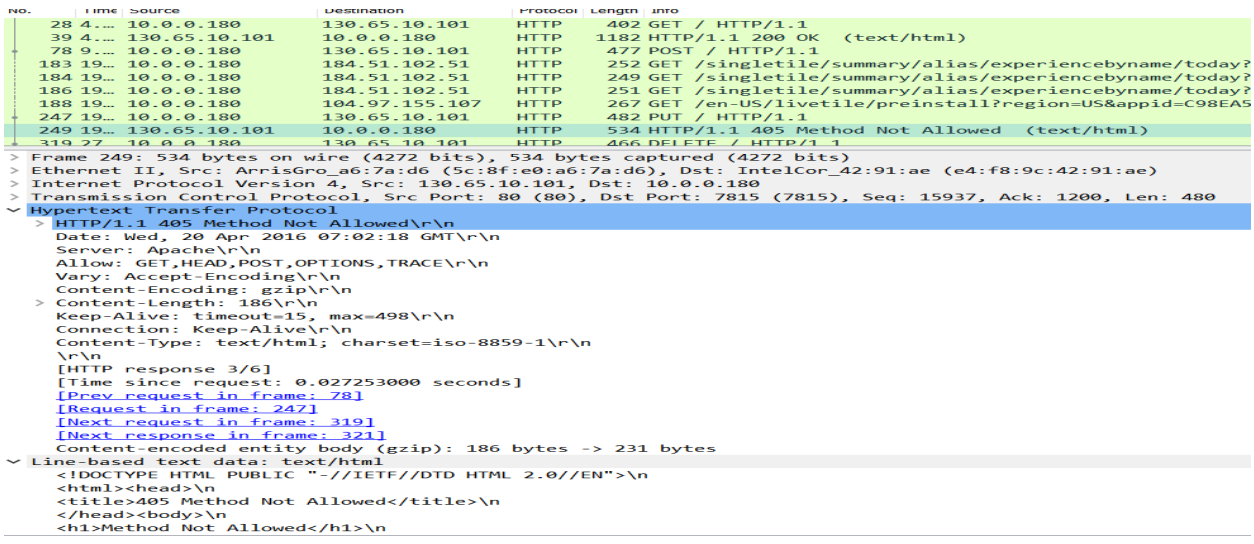
A single resource MAY be identified by many different URIs. For example, an article might have a URI for identifying "the current version" which is separate from the URI identifying each particular version. In this case, a PUT request on a general URI might result in several other URIs being defined by the origin server.

HTTP/1.1 does not define how a PUT method affects the state of an origin server.

Following is a Wireshark capture for PUT method while requesting



The response to PUT method for above request is as follows



It shows that the PUT method is not allowed on requested resource.

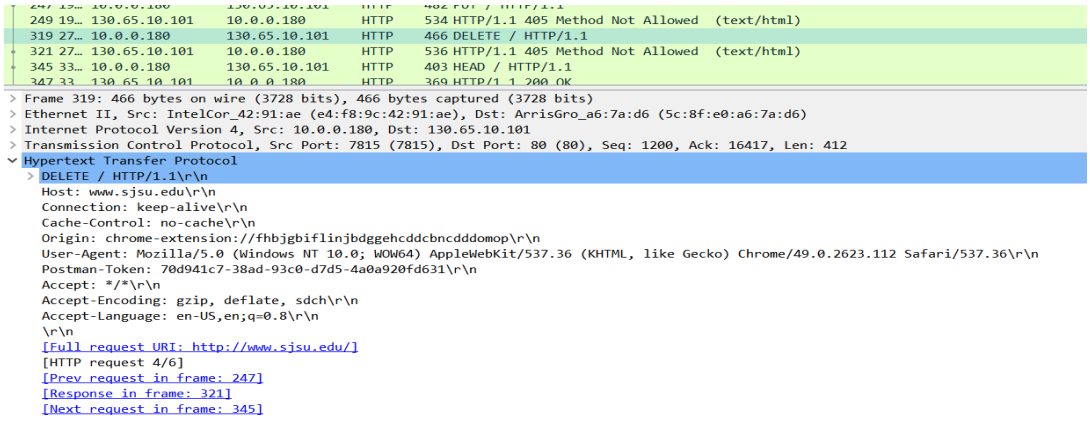
DELETE Method

This method is used to delete resource pointed by the URI. To provide security, this method can be overridden on the server. The action is not guaranteed in implementation though the response for such request is successful. However, success message is received only upon deletion or the resource is reallocated to some inaccessible location.

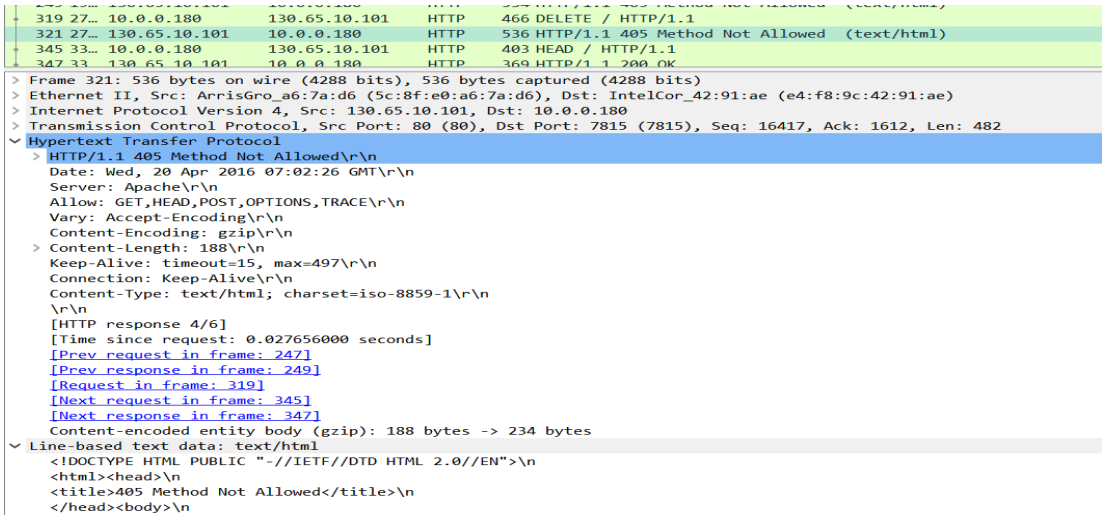
The response should be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include an entity.

Responses to this method are not cacheable.

Following is a Wireshark capture for DELETE method while requesting



The following is the response received for DELETE request observed in above capture.



TRACE Method

The TRACE method is used to echo requests. As a response the contents of an HTTP Request are sent back to the requester. Such method is used for debugging purpose at the time of development.

The final recipient should send response with status of 200(OK). A TRACE request will not include any entity.

Max-Forwards header field allows client in limiting length of the request chain, which is useful to test a chain of proxies involving an infinite loop.

If request is valid, the response should contain entire request message in the entity-body, with a Content-Type of "message/http".

Responses to this method is not cacheable.

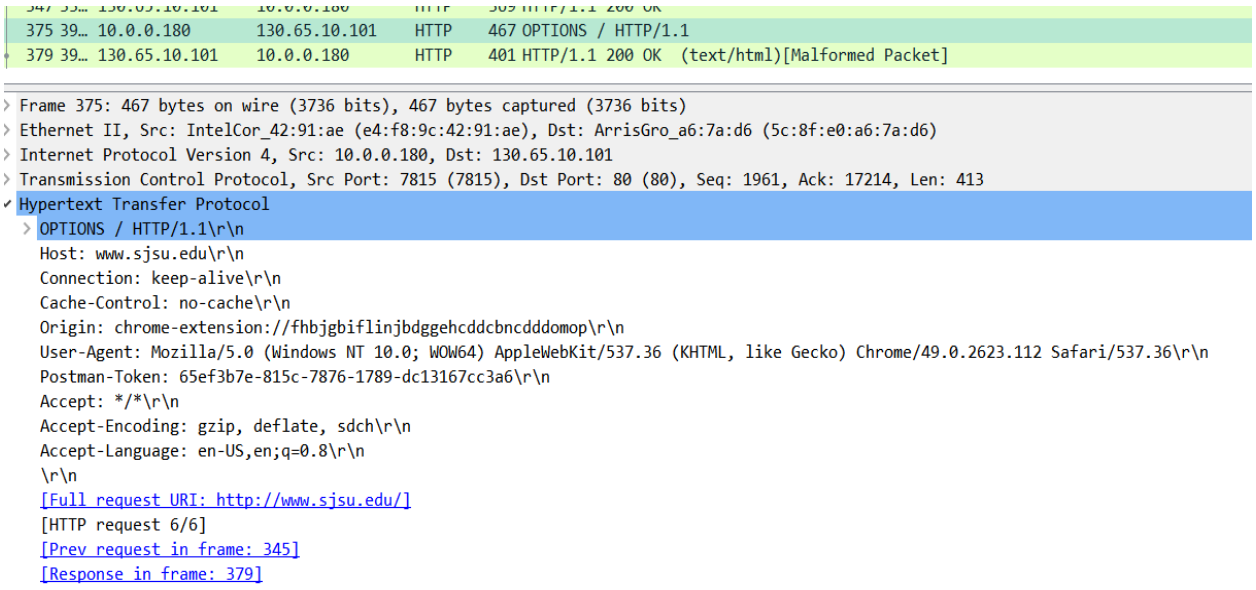
CONNECT Method

The CONNECT method is used by the client to establish proper connection with the server over HTTP.

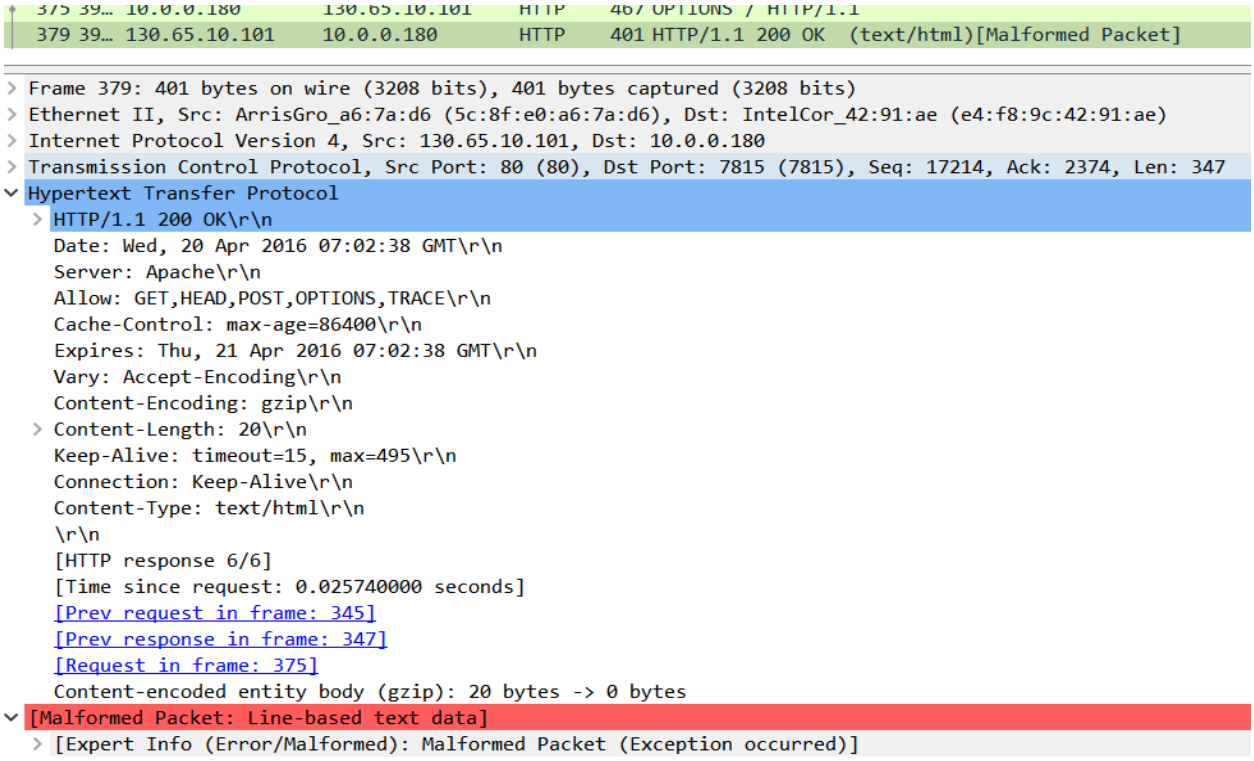
OPTIONS Method

The OPTIONS method is used by the client to find out the HTTP methods and other options supported by a web server. The client can specify a URL for the OPTIONS method, or an asterisk (*) to refer to the entire server.

Following is a Wireshark capture for OPTIONS method while requesting



The following is the response received for OPTIONS request observed in above capture. It also shows the methods available on resource with the Allow header.



HTTP Status Codes

The status code is included in response to any request from the client. It is a 3 digit number. The first digit is used to categorize the response. So based on first digit the response is categorized in 5 types.

- 1xx: Informational: It specifies that the request has been received and the process is continuing.
- 2xx: Success: It notifies that the action was successfully received, understood, and accepted.
- 3xx: Redirection: It means further action must be taken in order to complete the request.
- 4xx: Client Error: It denotes that the request contains incorrect syntax or cannot be fulfilled.
- 5xx: Server Error: It means the server failed to fulfill an apparently valid request.

Under each category there are several types of specific messages.

1xx: Information

Message	Description

100 Continue	Only a part of the request has been received by the server, but as long as it has not been rejected, the client should continue with the request.
101 Switching Protocols	The server switches protocol.

2xx: Successful

Message	Description
200 OK	The request is OK.
201 Created	The request is complete, and a new resource is created.
202 Accepted	The request is accepted for processing, but the processing is not complete.
203 Non-authoritative Information	The information in the entity header is from a local or third-party copy, not from the original server.
204 No Content	A status code and a header are given in the response, but there is no entity-body in the reply.
205 Reset Content	The browser should clear the form used for this transaction for additional input.
206 Partial Content	The server is returning partial data of the size requested. Used in response to a request specifying a Range header. The server must specify the range included in the response with the Content-Rangeheader.

3xx: Redirection

Message	Description
300 Multiple Choices	A link list. The user can select a link and go to that location. Maximum five addresses.
301 Moved Permanently	The requested page has moved to a new url.
302 Found	The requested page has moved temporarily to a new url.
303 See Other	The requested page can be found under a different url.
304 Not Modified	This is the response code to an If-Modified-Since or If-None-Matchheader, where the URL has not been modified since the specified date.

305 Use Proxy	The requested URL must be accessed through the proxy mentioned in the Location header.
306 Unused	This code was used in a previous version. It is no longer used, but the code is reserved.
307 Temporary Redirect	The requested page has moved temporarily to a new url.

4xx: Client Error

Message	Description
400 Bad Request	The server did not understand the request.
401 Unauthorized	The requested page needs a username and a password.
402 Payment Required	You cannot use this code yet.
403 Forbidden	Access is forbidden to the requested page.
404 Not Found	The server cannot find the requested page.
405 Method Not Allowed	The method specified in the request is not allowed.
406 Not Acceptable	The server can only generate a response that is not accepted by the client.
407 Proxy Authentication Required	You must authenticate with a proxy server before this request can be served.
408 Request Timeout	The request took longer than the server was prepared to wait.
409 Conflict	The request could not be completed because of a conflict.
410 Gone	The requested page is no longer available.
411 Length Required	The "Content-Length" is not defined. The server will not accept the request without it.
412 Precondition Failed	The pre-condition given in the request evaluated to false by the server.
413 Request Entity Too Large	The server will not accept the request, because the request entity is too large.
414 Request-url Too Long	The server will not accept the request, because the url is too long. Occurs when you convert a "post" request to a "get"

	request with a long query information.
415 Unsupported Media Type	The server will not accept the request, because the mediatype is not supported.
416 Requested Range Not acceptable	The requested byte range is not available and is out of bounds.
417 Expectation Failed	The expectation given in an Expect request-header field could not be met by this server.

5xx: Server Error

Message	Description
500 Internal Server Error	The request was not completed. The server met an unexpected condition.
501 Not Implemented	The request was not completed. The server did not support the functionality required.
502 Bad Gateway	The request was not completed. The server received an invalid response from the upstream server.
503 Service Unavailable	The request was not completed. The server is temporarily overloading or down.
504 Gateway Timeout	The gateway has timed out.
505 HTTP Version Not Supported	The server does not support the "http protocol" version.

HTTP Caching

HTTP is typically used for distributed information systems, where performance can be improved by the use of response caches. The HTTP/1.1 protocol includes a number of elements intended to make caching work. The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases.

The basic cache mechanisms in HTTP/1.1 are implicit directives to caches where server-specifies expiration times and validators. We use the **Cache-Control** header for this purpose. 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

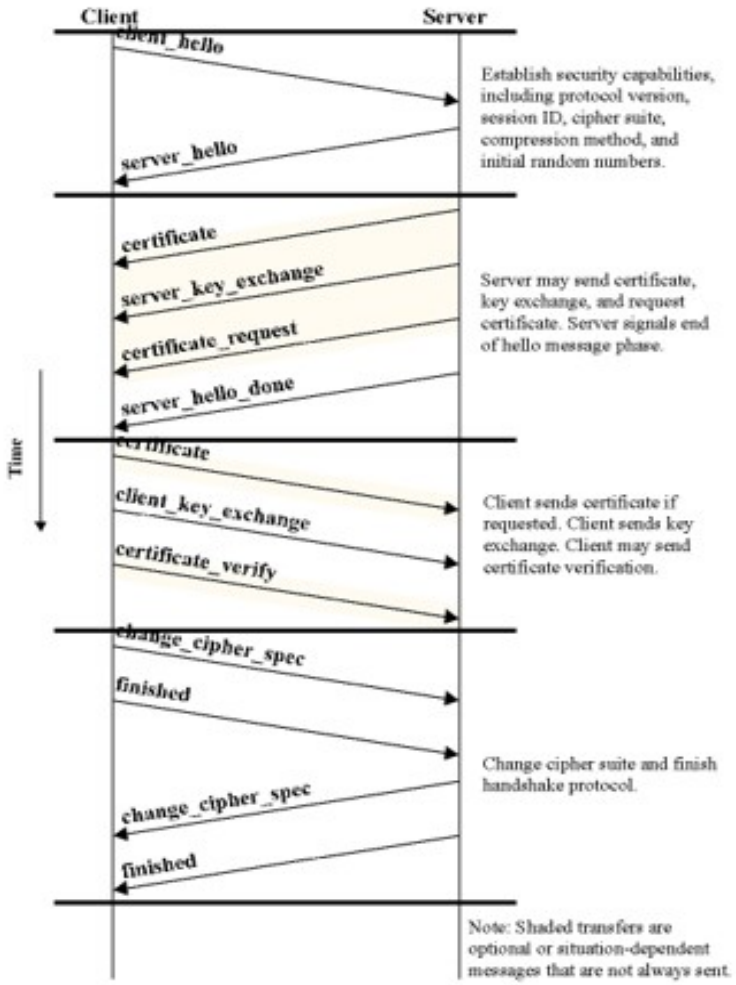
The following cache request directives can be used by the client in its HTTP request:

1	**no-cache**: A cache must not use the response to satisfy a subsequent request without successful revalidation with the origin server.
2	**no-store**: The cache should not store anything about the client request or server response.
3	**max-age = seconds**: Indicates that the client is willing to accept a response whose age is not greater than the specified time in seconds.
4	**max-stale [= seconds]**: Indicates that the client is willing to accept a response that has exceeded its expiration time. If seconds are given, it must not be expired by more than that time.
5	**min-fresh = seconds**: Indicates that the client is willing to accept a response whose freshness lifetime is not less than its current age plus the specified time in seconds.
6	**no-transform**: Does not convert the entity-body.
7	**only-if-cached**: Does not retrieve new data. The cache can send a document only if it is in the cache, and should not contact the origin-server to see if a newer copy exists.

The following cache response directives can be used by the server in its HTTP response:

1	**Public**: Indicates that the response may be cached by any cache.
2	**Private**: Indicates that all or part of the response message is intended for a single user and must not be cached by a shared cache.
3	**no-cache:** A cache must not use the response to satisfy a subsequent request without successful re-validation with the origin server.
4	**no-store**: The cache should not store anything about the client request or server response.
5	**no-transform**: Does not convert the entity-body.
6	**must-revalidate**: The cache must verify the status of stale documents before using it and expired ones should not be used.
7	**proxy-revalidate**: The proxy-revalidate directive has the same meaning as the must- revalidate directive, except that it does not apply to non-shared user agent caches.
8	**max-age = seconds**: Indicates that the client is willing to accept a response whose age is not greater than the specified time in seconds.
9	**s-maxage = seconds**: The maximum age specified by this directive overrides the maximum age specified by either the max-age directive or the Expires header. The s-maxage directive is always ignored by a private cache.

HTTP HANDSHAKE:



HTTP PAYLOAD FORMAT:

Request message:

The request message consists of the following:

- A request line(e.g., GET /images/logo.png HTTP/1.1, which requests a resource called /images/logo.png from the server)
- Request header fields (e.g., Accept-Language: en).
- An empty line.
- An optional message body.

The request line and other header fields must each end with <CR><LF> (that is, a carriage return character followed by a line feed character). The empty line must consist of only <CR><LF> and no other whitespace. In the HTTP/1.1 protocol, all header fields except Host are optional. A request line

containing only the path name is accepted by servers to maintain compatibility with HTTP clients before the HTTP/1.0 specification in RFC 1945.

Response message

The response message consists of the following:

- A status line which includes the status code and reason message (e.g., HTTP/1.1 200 OK, which indicates that the client's request succeeded).
- Response header fields (e.g., Content-Type: text/html).
- An empty line.
- An optional message body.

The status line and other header fields must all end with <CR><LF>. The empty line must consist of only <CR><LF> and no other whitespace. This strict requirement for <CR><LF> is relaxed somewhat within message bodies for consistent use of other system line breaks such as <CR> or <LF> alone.

The following example illustrates a typical message exchange for a GET request on the URI "http://www.example.com/hello.txt":

Client request:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

Server response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.
```

HTML URL ENCODING

URL encoding converts characters into a format that can be transmitted over the Internet.

URL - Uniform Resource Locator

Web browsers request pages from web servers by using a URL. The URL is the address of a web page, like: http://www.w3schools.com.

URL Encoding

URLs can only be sent over the Internet using the ASCII character-set. Since URLs often contain characters outside the ASCII set, the URL has to be converted into a valid ASCII format. URL encoding

replaces unsafe ASCII characters with a "%" followed by two hexadecimal digits. URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign or with %20.

Ascii Encoding Reference

The browser will encode input, according to the character-set used in our page. The default character-set in HTML5 is UTF-8.

Character	From Windows-1252	From UTF-8
space	%20	%20
!	%21	%21
"	%22	%22
#	%23	%23
$	%24	%24
%	%25	%25
&	%26	%26
'	%27	%27
(	%28	%28
)	%29	%29
*	%2A	%2A
+	%2B	%2B
,	%2C	%2C
-	%2D	%2D

.	%2E	%2E
/	%2F	%2F
0	%30	%30
1	%31	%31
2	%32	%32
3	%33	%33
4	%34	%34
5	%35	%35
6	%36	%36
7	%37	%37
8	%38	%38
9	%39	%39
:	%3A	%3A
;	%3B	%3B
<	%3C	%3C
=	%3D	%3D
>	%3E	%3E
?	%3F	%3F
@	%40	%40
A	%41	%41
B	%42	%42
C	%43	%43
D	%44	%44
E	%45	%45
F	%46	%46
G	%47	%47
H	%48	%48
I	%49	%49
J	%4A	%4A
K	%4B	%4B

L	%4C	%4C
M	%4D	%4D
N	%4E	%4E
O	%4F	%4F
P	%50	%50
Q	%51	%51
R	%52	%52
S	%53	%53
T	%54	%54
U	%55	%55
V	%56	%56
W	%57	%57
X	%58	%58
Y	%59	%59
Z	%5A	%5A

[	%5B	%5B
\	%5C	%5C
]	%5D	%5D
^	%5E	%5E
_	%5F	%5F
`	%60	%60
a	%61	%61
b	%62	%62
c	%63	%63
d	%64	%64
e	%65	%65
f	%66	%66
g	%67	%67
h	%68	%68
i	%69	%69

j	%6A	%6A
k	%6B	%6B
l	%6C	%6C
m	%6D	%6D
n	%6E	%6E
o	%6F	%6F
p	%70	%70
q	%71	%71
r	%72	%72
s	%73	%73
t	%74	%74
u	%75	%75
v	%76	%76
w	%77	%77
x	%78	%78
y	%79	%79
z	%7A	%7A
{	%7B	%7B
|	%7C	%7C
}	%7D	%7D
~	%7E	%7E
	%7F	%7F
`	%80	%E2%82%AC
	%81	%81
‚	%82	%E2%80%9A
ƒ	%83	%C6%92
„	%84	%E2%80%9E
…	%85	%E2%80%A6
†	%86	%E2%80%A0
‡	%87	%E2%80%A1

ˆ	%88	%CB%86
‰	%89	%E2%80%B0
Š	%8A	%C5%A0
‹	%8B	%E2%80%B9
Œ	%8C	%C5%92
	%8D	%C5%8D
Ž	%8E	%C5%BD
	%8F	%8F
	%90	%C2%90
‘	%91	%E2%80%98
’	%92	%E2%80%99
“	%93	%E2%80%9C
”	%94	%E2%80%9D
•	%95	%E2%80%A2
–	%96	%E2%80%93
—	%97	%E2%80%94
˜	%98	%CB%9C
™	%99	%E2%84
š	%9A	%C5%A1
›	%9B	%E2%80
œ	%9C	%C5%93
	%9D	%9D
ž	%9E	%C5%BE
Ÿ	%9F	%C5%B8
	%A0	%C2%A0
¡	%A1	%C2%A1
¢	%A2	%C2%A2
£	%A3	%C2%A3
¤	%A4	%C2%A4
¥	%A5	%C2%A5

¦	%A6	%C2%A6
§	%A7	%C2%A7
¨	%A8	%C2%A8
©	%A9	%C2%A9
ª	%AA	%C2%AA
«	%AB	%C2%AB
¬	%AC	%C2%AC
	%AD	%C2%AD
®	%AE	%C2%AE
¯	%AF	%C2%AF
°	%B0	%C2%B0
±	%B1	%C2%B1
²	%B2	%C2%B2
³	%B3	%C2%B3
´	%B4	%C2%B4
µ	%B5	%C2%B5
¶	%B6	%C2%B6
·	%B7	%C2%B7
¸	%B8	%C2%B8
¹	%B9	%C2%B9
º	%BA	%C2%BA
»	%BB	%C2%BB
¼	%BC	%C2%BC
½	%BD	%C2%BD
¾	%BE	%C2%BE
¿	%BF	%C2%BF
À	%C0	%C3%80
Á	%C1	%C3%81
Â	%C2	%C3%82
Ã	%C3	%C3%83

Ä	%C4	%C3%84
Å	%C5	%C3%85
Æ	%C6	%C3%86
Ç	%C7	%C3%87
È	%C8	%C3%88
É	%C9	%C3%89
Ê	%CA	%C3%8A
Ë	%CB	%C3%8B
Ì	%CC	%C3%8C
Í	%CD	%C3%8D
Î	%CE	%C3%8E
Ï	%CF	%C3%8F
Ð	%D0	%C3%90
Ñ	%D1	%C3%91
Ò	%D2	%C3%92
Ó	%D3	%C3%93
Ô	%D4	%C3%94
Õ	%D5	%C3%95
Ö	%D6	%C3%96
×	%D7	%C3%97
Ø	%D8	%C3%98
Ù	%D9	%C3%99
Ú	%DA	%C3%9A
Û	%DB	%C3%9B
Ü	%DC	%C3%9C
Ý	%DD	%C3%9D
Þ	%DE	%C3%9E
ß	%DF	%C3%9F
à	%E0	%C3%A0
á	%E1	%C3%A1

â	%E2	%C3%A2
ã	%E3	%C3%A3
ä	%E4	%C3%A4
å	%E5	%C3%A5
æ	%E6	%C3%A6
ç	%E7	%C3%A7
è	%E8	%C3%A8
é	%E9	%C3%A9
ê	%EA	%C3%AA
ë	%EB	%C3%AB
ì	%EC	%C3%AC
í	%ED	%C3%AD
î	%EE	%C3%AE
ï	%EF	%C3%AF
ð	%F0	%C3%B0
ñ	%F1	%C3%B1
ò	%F2	%C3%B2
ó	%F3	%C3%B3
ô	%F4	%C3%B4
õ	%F5	%C3%B5
ö	%F6	%C3%B6
÷	%F7	%C3%B7
ø	%F8	%C3%B8
ù	%F9	%C3%B9
ú	%FA	%C3%BA
û	%FB	%C3%BB
ü	%FC	%C3%BC
ý	%FD	%C3%BD
þ	%FE	%C3%BE
ÿ	%FF	%C3%BF

URL ENCODING REFERENCE

The ASCII control characters %00-%1F were originally designed to control hardware devices. Control characters have nothing to do inside a URL.

ASCII Character	Description	URL-encoding
NUL	null character	%00
SOH	start of header	%01
STX	start of text	%02
ETX	end of text	%03
EOT	end of transmission	%04
ENQ	enquiry	%05
ACK	acknowledge	%06
BEL	bell (ring)	%07
BS	backspace	%08
HT	horizontal tab	%09
LF	line feed	%0A
VT	vertical tab	%0B
FF	form feed	%0C
CR	carriage return	%0D

SO	shift out	%0E
SI	shift in	%0F
DLE	data link escape	%10
DC1	device control 1	%11
DC2	device control 2	%12
DC3	device control 3	%13
DC4	device control 4	%14
NAK	negative acknowledge	%15
SYN	synchronize	%16
ETB	end transmission block	%17
CAN	cancel	%18
EM	end of medium	%19
SUB	substitute	%1A
ESC	escape	%1B
FS	file separator	%1C
GS	group separator	%1D
RS	record separator	%1E
US	unit separator	%1F

HTTP SECURITY

HTTP is used for communications over the internet, so application developers, information providers, and users should be aware of the security limitations in HTTP/1.1.

Personal Information Leakage

HTTP clients are often privy to large amount of personal information such as the user's name, location, mail address, passwords, encryption keys, etc. So we should be very careful to prevent unintentional leakage of this information via the HTTP protocol to other sources.

- All the confidential information should be stored at the server in encrypted form.
- Revealing the specific software version of the server might allow the server machine to become more vulnerable to attacks against software that is known to contain security holes.
- Proxies that serve as a portal through a network firewall should take special precautions regarding the transfer of header information that identifies the hosts behind the firewall.
- The information sent in the 'From' field might conflict with the user's privacy interests or their site's security policy, and hence, it should not be transmitted without the user being able to disable, enable, and modify the contents of the field.
- Clients should not include a Referer header field in a (non-secure) HTTP request, if the referring page was transferred with a secure protocol.
- Authors of services that use the HTTP protocol should not use GET based forms for the submission of sensitive data, because it will cause the data to be encoded in the Request-URI.

File and Path Names Based Attack

The document should be restricted to the documents returned by HTTP requests to be only those that were intended by the server administrators. For example, UNIX, Microsoft Windows, and other operating systems use '..' as a path component to indicate a directory level above the current one. On such a system, an HTTP server MUST disallow any such construct in the Request-URI, if it would otherwise allow access to a resource outside those intended to be accessible via the HTTP server.

DNS Spoofing

Clients using HTTP rely heavily on the Domain Name Service, and are thus generally prone to security attacks based on the deliberate mis-association of IP addresses and DNS names. So clients need to be cautious in assuming the continuing validity of an IP number/DNS name association. If HTTP clients cache the results of host name lookups in order to achieve a performance improvement, they must observe the TTL information reported by the DNS. If HTTP clients do not observe this rule, they could be spoofed when a previously-accessed server's IP address changes.

Location Headers and Spoofing

If a single server supports multiple organizations that do not trust one another, then it MUST check the values of Location and Content Location headers in the responses that are generated under the control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority**.**

Authentication Credentials

Existing HTTP clients and user agents typically retain authentication information indefinitely. HTTP/1.1 does not provide a method for a server to direct clients to discard these cached credentials which is a big security risk. There are a number of work around to the parts of this problem, and so it is recommended to make the use of password protection in screen savers, idle time-outs, and other methods that mitigate the security problems inherent in this problem.

Proxies and Caching

HTTP proxies are men-in-the-middle, and represent an opportunity for man-in-the-middle attacks. Proxies have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers. Proxy operators should protect the systems on which proxies run, as they would protect any system that contains or transports sensitive information. Caching proxies provide additional potential vulnerabilities, since the contents of the cache represent an attractive target for malicious exploitation. Therefore, cache contents should be protected as sensitive information.

Learning Outcomes

HTTP is the probably the most common protocol used. whenever you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server.

When this request message reaches the server, the server can take either one of these actions:

- The server interprets the request received, maps the request into a *file* under the server's document directory, and returns the file requested to the client.
- The server interprets the request received, maps the request into a *program* kept in the server, executes the program, and returns the output of the program to the client.
- The request cannot be satisfied; the server returns an error message.

Also, We learnt that there are different status codes for delivering messages to the user varying from 200 for Ok and 404 for File Not Found. These are probably the most popular of the bunch.

Conclusion