SAN JOSÉ STATE UNIVERSITY

Department of Computer Engineering

Network Architecture and Protocol

(CMPE208)

Hypertext Transfer Protocol

Group Lab

Group 5
Charit Upadhyay
Devika Jadhav
Pradeep Patil

Table of Contents

INTRODUCTION

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. The figure below shows the basic architecture of a client-server model using HTTP Protocol.

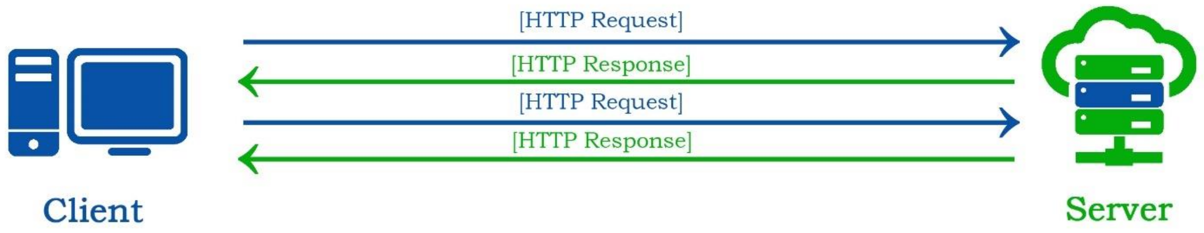


Image Reference: created using Adobe photoshop

HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the client and an application running on a pc or a computer hosting a website may be the server. The client submits an **HTTP** request message to the server. The server, which provides resources such as **HTML** files and other content, or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

The purpose of setting up this lab is to study the various concepts of **Hypertext Transfer Protocol** (**HTTP**). HTTP is an application layer protocol designed within the framework of the Internet protocol suite. Its definition presumes an underlying and reliable transport layer protocol , and Transmission Control Protocol (TCP) is commonly used. However, HTTP can be adapted to use unreliable protocols such as the User Datagram Protocol (UDP). With the help of following tools, we experiment and check the behavior of ICMP messages.

- **Oracle Virtual Box**: A software virtualization package that installs on an operating system as an application. VirtualBox allows additional operating systems to be installed on it, as a Guest OS, and run in a virtual environment.

- **Wireshark**: An open source network packet analyzer, which allows examining the network packet data at microscopic level.

- **Telnet:** **Telnet:** is a protocol used on the Internet or local area network to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. We will be using Telnet to send http requests.

HTTP OVERVIEW

HTTP (Hyper Text Transfer Protocol) is protocol allows to fetch resources like HTML/ Web pages. It provides the foundation for any type of data exchange on Web and a client-server protocol which means request are initiated by recipient usually the web browser. HTTP is an application layer protocol, usually used for communication between web browser and web server but can be used for other purposes as well.

HTTP generally follows the client-server model where clients sends the request to the server and waits until it receives the response from the server. It's a stateless protocol which means server doesn't keep any data between two requests. It is generally based on TCP/IP layer and work with any reliable transport layer so that it doesn't lose data packets silently unlike UDP.

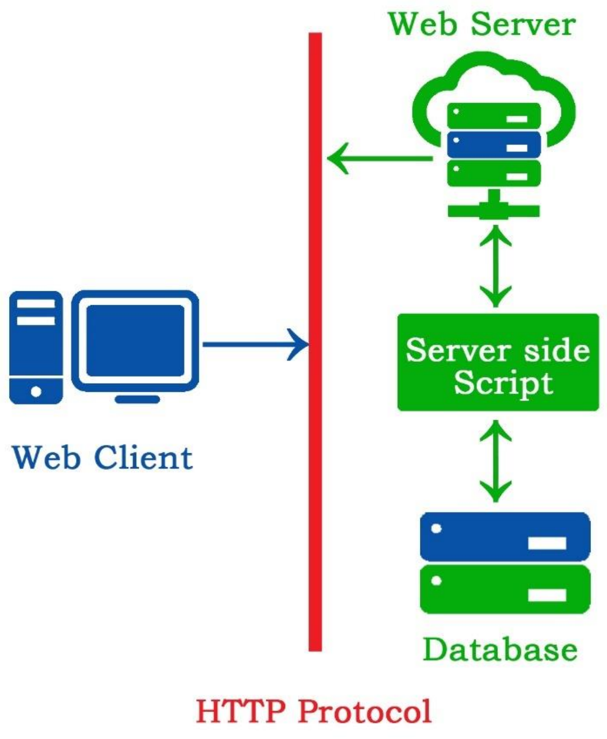


Image Reference: created using Adobe photoshop

HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the client and an application running on a computer hosting a website may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

HTTP is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time. Web browsers cache previously accessed web resources and reuse them, when possible, to reduce network traffic. HTTP proxy servers at private network boundaries can facilitate communication for clients without a globally routable address, by relaying messages with external servers.

HTTP uses default port **TCP 80** and provide standardize way for communication for multiple systems.

HTTP REQUEST METHODS

HTTP defines methods (sometimes referred to as verbs, but nowhere in the specification does it mention verb, nor is OPTIONS or HEAD a verb) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server. The HTTP/1.0 specification[13] defined the **GET, POST** and **HEAD** methods and the HTTP/1.1 specification[14] added five new methods: **OPTIONS, PUT, DELETE, TRACE and CONNECT**. By being specified in these documents, their semantics are well-known and can be depended on. Any client can use any method and the server can be configured to support any combination of methods. If a method is unknown to an intermediate, it will be treated as an unsafe and non-idempotent method. There is no limit to the number of methods that can be defined, and this allows for future methods to be specified without breaking existing infrastructure. For example, WebDAV defined 7 new methods and RFC 5789 specified the PATCH method.

Method	RFC	DESCRIPTION
GET	RFC 7231	Retrieve a file (95% of requests)
HEAD	RFC 7231	Just get meta-data
POST	RFC 7231	Submitting a form to a server
PUT	RFC 7231	Store enclosed document as an URI
DELETE	RFC 7231	Remove named resource
CONNECT	RFC 7231	Used by proxies for tunneling
OPTIONS	RFC 7231	Request for server/proxy options
TRACE	RFC 7231	HTTP ‘echo’ for Debugging
PATCH	RFC 5789	Partial modifications to a resource

Table 1. List of HTTP Methods

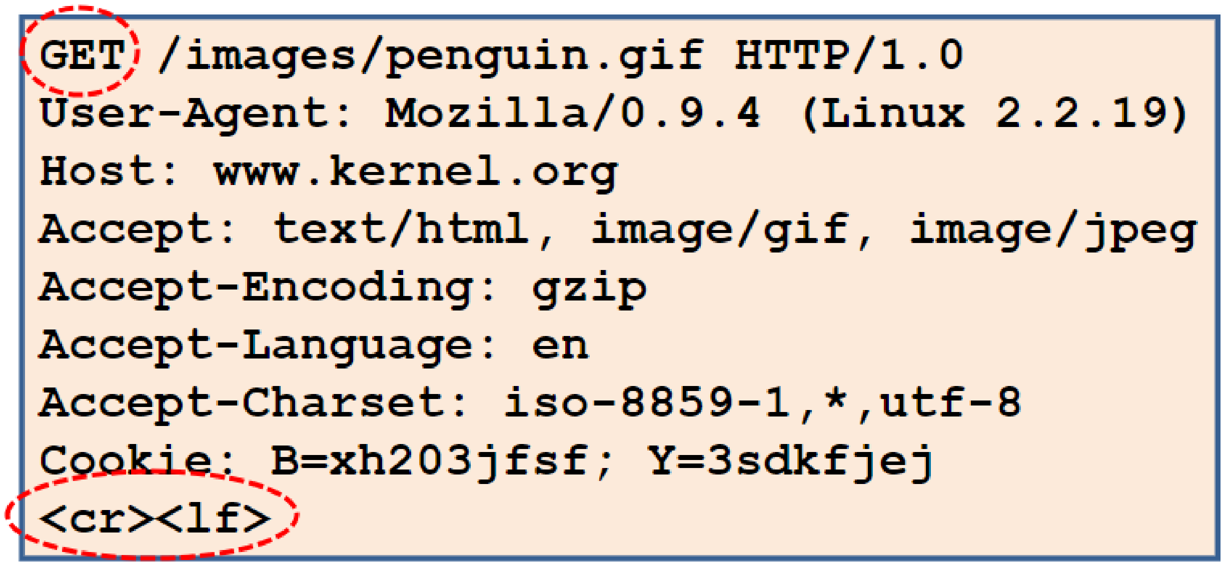
Some of the methods (for example, HEAD, GET, OPTIONS and TRACE) are, by convention, defined as safe, which means they are intended only for information retrieval and should not change the state of the server. In other words, they should not have side effects, beyond relatively harmless effects such as logging, web caching, the serving of banner advertisements or incrementing a web counter. Making arbitrary GET requests without regard to the context of the application's state should therefore be considered safe. However, this is not mandated by the standard, and it is explicitly acknowledged that it cannot be guaranteed.

By contrast, methods such as POST, PUT, DELETE and PATCH are intended for actions that may cause side effects either on the server, or external side effects such as financial transactions or transmission of email. Such methods are therefore not usually used by conforming web robots or web crawlers; some that do not conform tend to make requests without regard to context or consequences. Despite the prescribed safety of GET requests, in practice their handling by the server is not technically limited in any way. Therefore, careless or deliberate programming can cause non-trivial changes on the server. This is discouraged, because it can cause problems for web caching, search engines and other automated agents, which can make unintended changes on the server.

HTTP Message Formats

Request Format

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



```
GET /images/penguin.gif HTTP/1.0
User-Agent: Mozilla/0.9.4 (Linux 2.2.19)
Host: www.kernel.org
Accept: text/html, image/gif, image/jpeg
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: B=xh203jfsf; Y=3sdkfjej
<cr><lf>
```

https://sjsu.instructure.com/courses/

HTTP requests are messages sent by the client to initiate an action on the server. Their *start-line* contain three elements:

An HTTP method

A Keyword that describes the action to be performed. The *request target*, usually a URL, or the absolute path of the protocol, port, and domain are usually characterized by the request context. The format of this request target varies between different HTTP methods. It can be an absolute path, A complete URL or The authority component of a URL. The *HTTP version*, which defines the structure of the remaining message, acting as an indicator of the expected version to use for the response.

Headers Section

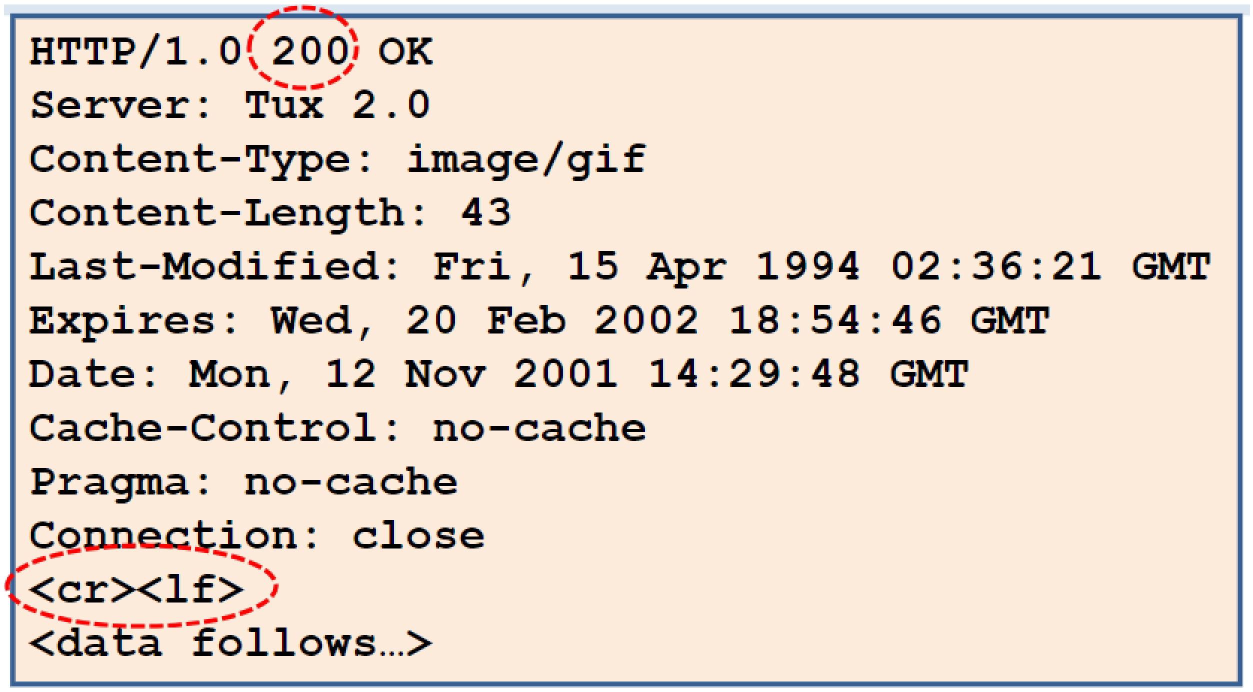
HTTP headers from a request follow the same basic structure of an HTTP header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consist of one single line, which can be quite long. There are numerous request headers available. They can be divided in several groups : General headers, Request headers, Entity headers.

Body Section

The final part of the request is its body. Not all requests have one. Some requests send data to the server in order to update it: as often the case with POST requests (containing HTML form data).Bodies can be broadly divided into two categories: **Single-resource bodies**, consisting of one single file, defined by the two headers: Content-Type and Content-Length. **Multiple-resource bodies**, consisting of a multipart body, each containing a different bit of information. This is typically associated with HTML Forms.

Request Format

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



```
HTTP/1.0 200 OK
Server: Tux 2.0
Content-Type: image/gif
Content-Length: 43
Last-Modified: Fri, 15 Apr 1994 02:36:21 GMT
Expires: Wed, 20 Feb 2002 18:54:46 GMT
Date: Mon, 12 Nov 2001 14:29:48 GMT
Cache-Control: no-cache
Pragma: no-cache
Connection: close
<cr><lf>
<data follows…>
```

https://sjsu.instructure.com/courses/

A typical Response message consist of the 3 parts Status line, Header and Body which are explained in detail below

Status line

The start line of an HTTP response, called the *status line*, contains the following information: The *protocol version*, usually HTTP/1.1. A *status code*, indicating success or failure of the request. Common status codes are 200, 404, or 302 A *status text*. A brief, purely informational, textual description of the status code to help a human understand the HTTP message.

Headers Section

HTTP headers for responses follow the same structure as any other header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the type of the header. The whole header, including its value, presents as a single line. There are numerous response headers available. These can be divided into several groups: *General headers, Response headers.*, *Entity headers.*

Body Section

The last part of a response is the body. Not all responses have one: responses with a status code, like 201 or 204, usually don't. Bodies can be broadly divided into three categories: Single-resource bodies, a file of known length, defined by the two headers: Content-Type and Content-Length. Single-resource bodies, consisting of a single file of unknown length, Transfer-Encoding set to chunked. Multiple-resource bodies, consisting of a multipart body, each containing a different section of information.

HTTP Status Codes

Status Code Series	Description
1XX	Informational Message
2XX	Success Message
3XX	Redirection Message
4XX	Error message related to client
5XX	Error message related to the server

Informational Series – 1xx
These are the informational codes send by the server indicating that the request is received from the client successfully and the same is under processing at the server end. This is a provisional response from the server normally contains only the status line and optional headers and is terminated with an empty line.
100 – continue.
101 - Switching Protocols

Success Series – 2xx
These are the success codes send by the server indicating that the request is received and processed successfully.
200 – OK . 201 Created. **206** - Partial Content

Redirection – 3xx
These are the status codes for redirection. When a user agent (a web browser or a crawler) requesting URL1 is redirected to another resource URL2 then 2xx codes are returned as a response. These codes are not seen in the browser window since browsers are auto redirected to another URL.
301 – Moved Permanently. **304 -** Modified

Client Errors – 4xx
These are the errors from the client side which the server could not resolve. Simple and well known example is a "404 – Page Not Found" error displayed in the browser window when an unavailable URL is requested by the browser.
400 – Bad Request. **403** – Forbidden. **404** Not Found

Server Errors Series – 5xx
When a web server can't fulfill a valid request from a client it sends a 5xx error code in the response. An example is "504 – Gateway Timeout" error where the web server1 is acting as a proxy to get a response from another web server2 but failed to receive a timely response.
500 – Internal server Error. **503 -** service unavailable. **505** – Version not supported

Benefits of HTTP

Identification
HTML was intended to be quick and lightweight. Speed of delivery is enabled by creating a notification of file type in the header of the data being transferred, known as MIME type. This enables the receiving application to quickly open the incoming file without having to ask the sender what application should be used to read or view the contents of the file.

Specialization
A Web page contains mixed elements such as text and images. Each element requires a different amount of resources to store and download. HTTP enables multiple connections to download separate elements concurrently, thus speeding up transmission. Each element is assigned its own particular file type and therefore can be handled faster and more efficiently by the receiving computer.

Addressing
The addressing scheme used by HTTP was also a revolutionary advancement. When computers had to be addressed using an IP address consisting of a series of numbers, the public found it difficult to engage with the Internet. Mapping IP addresses to easily recognizable names made the World Wide Web commercially viable.

Flexibility
With file type notification preceding data transmission, the receiving application has the option of quickly downloading extensions or plug-ins if additional capabilities are needed to display the data. These add-ons include Flash players and PDF document readers.

Security
HTTP 1.0 downloads each file over an independent connection and then closes the connection. This reduces the risk of interception during transmission, as the connection does not persist beyond the transfer of a single element of a Web page. Hypertext Transfer Protocol Secure (HTTPS) encrypts the HTTP exchange to add further security.

Ease of Programming
HTTP is coded in plain text and therefore is easier to follow and implement than protocols that make use of codes that require lookups. Data is formatted in lines of text and not as strings of variables or fields.

Search Capabilities
Although HTTP is a simple messaging protocol, it includes the ability to search a database with a single request. This allows the protocol to be used to carry out SQL searches and return results conveniently formatted in an HTML document.

Persistent Connections
One minor drawback of HTTP is the need to create multiple connections in order to transmit a typical Web page, which causes an administrative overhead. HTTP 1.1 has the ability to maintain an open connection for several requests. In addition, the concept of "pipelining" was added, enabling many requests to be sent to the receiving computer before the first request is served. These two measures speed up the response time for delivering a Web page.

HTTP v/s Security

This section is meant to inform application developers, information providers, and users of the security limitations in HTTP/1.1 as described by this document. The discussion does not include definitive solutions to the problems revealed, though it does make some suggestions for reducing security risks.

Personal Information
HTTP clients are often privy to large amounts of personal information (e.g. the user's name, location, mail address, passwords, encryption keys, etc.), and SHOULD be very careful to prevent unintentional leakage of this information via the HTTP protocol to other sources.

Abuse of Server Log Information
A server is in the position to save personal data about a user's requests which might identify their reading patterns or subjects of interest. This information is clearly confidential in nature and its handling can be constrained by law in certain countries. People using the HTTP protocol to provide data are responsible for ensuring that such material is not distributed without the permission of any individuals that are identifiable by the published results.

Transfer of Sensitive Information
Like any generic data transfer protocol, HTTP cannot regulate the content of the data that is transferred, nor is there any a priori method of determining the sensitivity of any particular piece of information within the context of any given request. Therefore, applications SHOULD supply as much control over this information as possible to the provider of that information. Four header fields are worth special mention in this context: Server, Via, Referer and From.

Encoding Sensitive Information in URI's
Because the source of a link might be private information or might reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referer field is sent. For example, a browser client could have a toggle switch for browsing openly/anonymously, which would respectively enable/disable the sending of Referer and From information. Clients SHOULD NOT include a Referrer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol.

Attacks Based On File and Path Names
Implementations of HTTP origin servers SHOULD be careful to restrict the documents returned by HTTP requests to be only those that were intended by the server administrators. If an HTTP server translates HTTP URIs directly into file system calls, the server MUST take special care not to serve files that were not intended to be delivered to HTTP clients. 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. Clients need to be cautious in assuming the continuing validity of an IP number/DNS name association.

In particular, HTTP clients SHOULD rely on their name resolver for confirmation of an IP number/DNS name association, rather than caching the result of previous host name lookups. Many platforms already can cache host name lookups locally when appropriate, and they SHOULD be configured to do so. It is proper for these lookups to be cached, however, only when the TTL (Time To Live) information reported by the name server makes it likely that the cached information will remain useful.

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 responses that are generated under control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority.

Content-Disposition Issues
RFC 1806 [35], from which the often implemented Content-Disposition header in HTTP is derived, has a number of very serious security considerations. Content-Disposition is not part of the HTTP standard, but since it is widely implemented, we are documenting its use and risks for implementors.

Authentication Credentials and Idle Clients
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. This is a significant defect that requires further extensions to HTTP. Circumstances under which credential caching can interfere with the application's security model include but are not limited to:

This is currently under separate study. There are a number of work- arounds to parts of this problem, and we encourage the use of password protection in screen savers, idle time-outs, and other methods which mitigate the security problems inherent in this problem. In particular, user agents which cache credentials are encouraged to provide a readily accessible mechanism for discarding cached credentials under user control.

Proxies and Caching
By their very nature, HTTP proxies are men-in-the-middle, and represent an opportunity for man-in-the-middle attacks. Compromise of the systems on which the proxies run can result in serious security and privacy problems. Proxies have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers. A compromised proxy, or a proxy implemented or configured without regard to security and privacy considerations, might be used in the commission of a wide range of potential attacks.

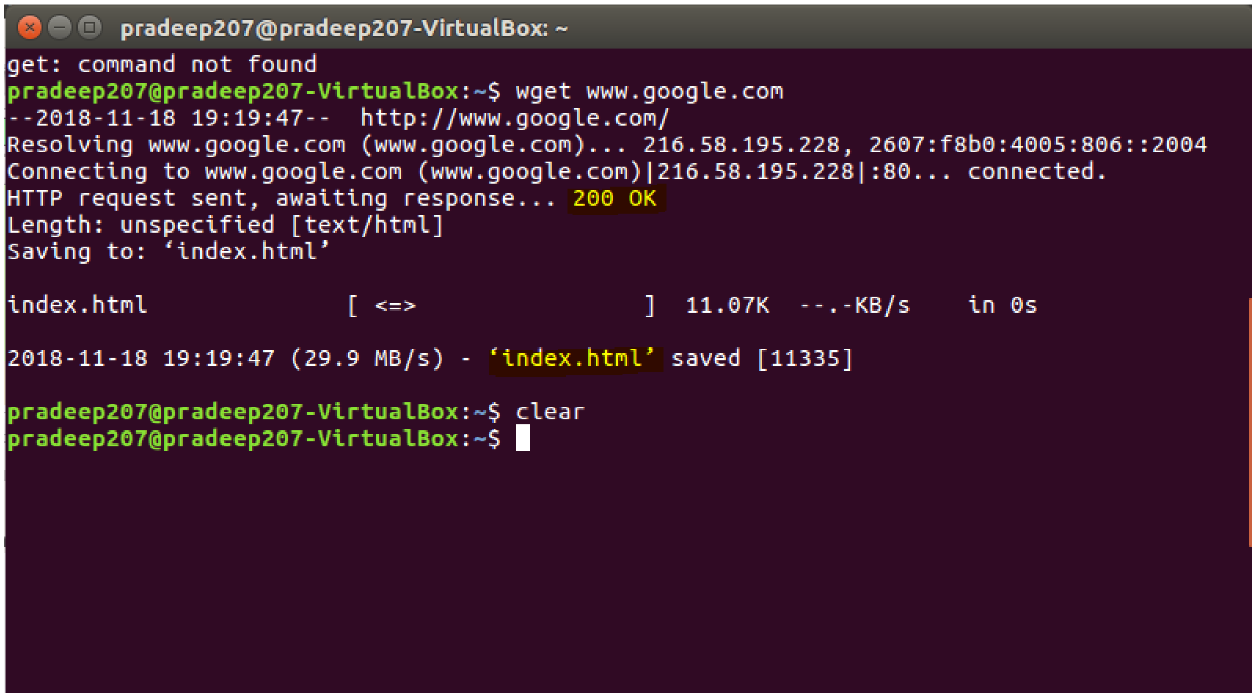
Proxy operators should protect the systems on which proxies run as they would protect any system that contains or transports sensitive information. In particular, log information gathered at proxies often contains highly sensitive personal information, and/or information about organizations. Log information should be carefully guarded, and appropriate guidelines for use developed and followed.

HTTP Lab and Observations

The behavior of various HTTP messages are observed using Telnet and Linux commands on a Linux based OS on a Virtual Machine. The packets that are sent and received are captured and studied using Wireshark. Following are the various HTTP Methods, Requests and their respective response details.

HTTP GET Method

We used the command WGET www.google.com in terminal to send a http request to get the google home page or the index.html file. The response indicates the status line as **200 OK** and the response body is saved as **index.html** .Following is the screenshot of response message



pradeep207@pradeep207-VirtualBox: ~

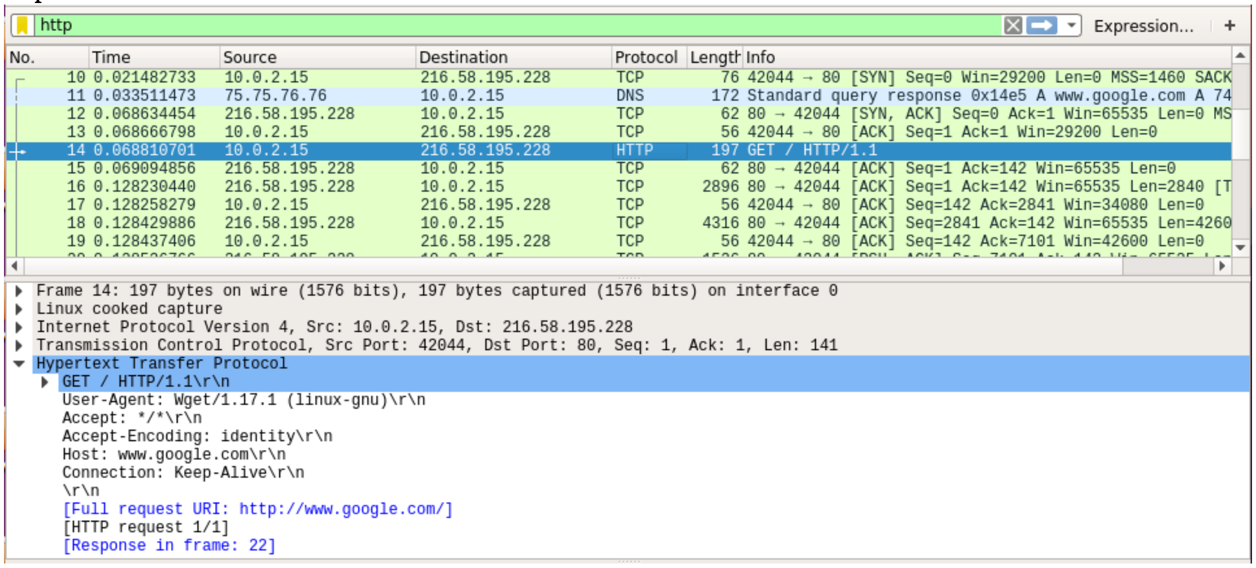
```
get: command not found
pradeep207@pradeep207-VirtualBox:~$ wget www.google.com
--2018-11-18 19:19:47--  http://www.google.com/
Resolving www.google.com (www.google.com)... 216.58.195.228, 2607:f8b0:4005:806::2004
Connecting to www.google.com (www.google.com)|216.58.195.228|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: ‘index.html’

index.html              [ <=>                ]  11.07K  --.-KB/s    in 0s

2018-11-18 19:19:47 (29.9 MB/s) - ‘index.html’ saved [11335]

pradeep207@pradeep207-VirtualBox:~$ clear
pradeep207@pradeep207-VirtualBox:~$ 
```

Wireshark capture for the GET request and Response.

Capture 1 shows the request message in detail. The request message contains request line, header and the message body. This request message is sent to the google server after an DNS query and a 3 way handshake. Once the server receives the request message, the server processes the request and sends the response message.

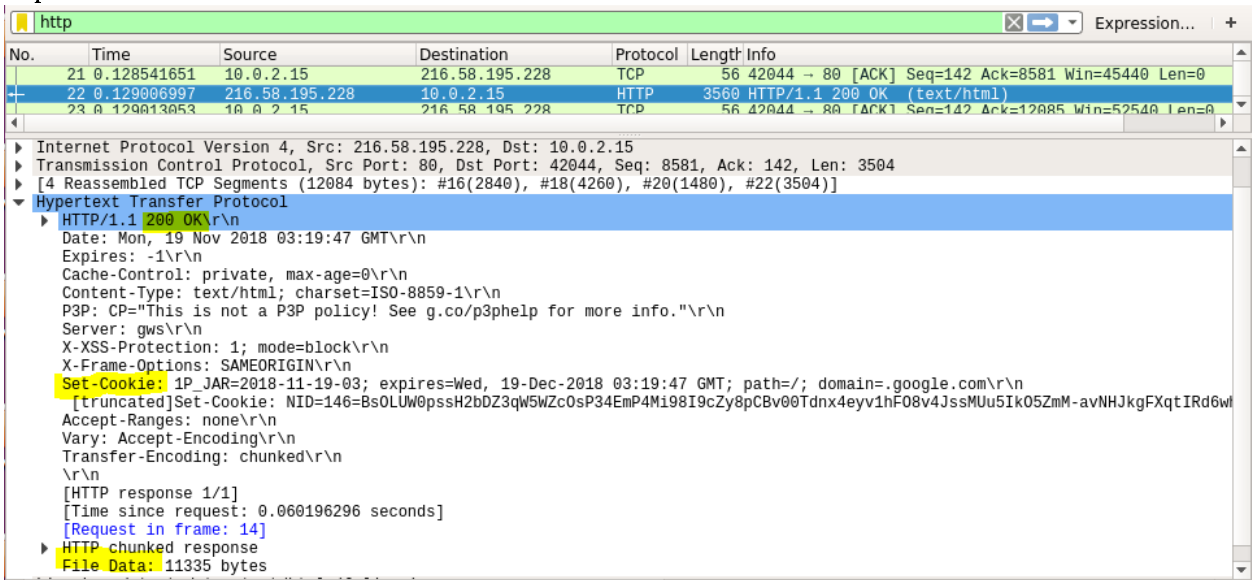
Capture 2 shows the HTTP response message from the google server. This response message contains the response status line response headers and the response message body. The message body contains the index.html which is the html page of google home page.

Capture 1



```
http

No.  Time          Source          Destination     Protocol Length Info
  10 0.021482733   10.0.2.15       216.58.195.228  TCP        76 42044 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK
  11 0.033511473   75.75.76.76     10.0.2.15       DNS       172 Standard query response 0x14e5 A www.google.com A 74
  12 0.068634454   216.58.195.228  10.0.2.15       TCP        62 80 → 42044 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MS
  13 0.068666798   10.0.2.15       216.58.195.228  TCP        56 42044 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
  14 0.068810701   10.0.2.15       216.58.195.228  HTTP      197 GET / HTTP/1.1
  15 0.069094856   216.58.195.228  10.0.2.15       TCP        62 80 → 42044 [ACK] Seq=1 Ack=142 Win=65535 Len=0
  16 0.128230440   216.58.195.228  10.0.2.15       TCP      2896 80 → 42044 [ACK] Seq=1 Ack=142 Win=65535 Len=2840 [T
  17 0.128258279   10.0.2.15       216.58.195.228  TCP        56 42044 → 80 [ACK] Seq=142 Ack=2841 Win=34080 Len=0
  18 0.128429886   216.58.195.228  10.0.2.15       TCP      4316 80 → 42044 [ACK] Seq=2841 Ack=142 Win=65535 Len=4260
  19 0.128437406   10.0.2.15       216.58.195.228  TCP        56 42044 → 80 [ACK] Seq=142 Ack=7101 Win=42600 Len=0

▸ Frame 14: 197 bytes on wire (1576 bits), 197 bytes captured (1576 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 216.58.195.228
▸ Transmission Control Protocol, Src Port: 42044, Dst Port: 80, Seq: 1, Ack: 1, Len: 141
▾ Hypertext Transfer Protocol
  ▸ GET / HTTP/1.1\r\n
    User-Agent: Wget/1.17.1 (linux-gnu)\r\n
    Accept: */*\r\n
    Accept-Encoding: identity\r\n
    Host: www.google.com\r\n
    Connection: Keep-Alive\r\n
    \r\n
    [Full request URI: http://www.google.com/]
    [HTTP request 1/1]
    [Response in frame: 22]
```

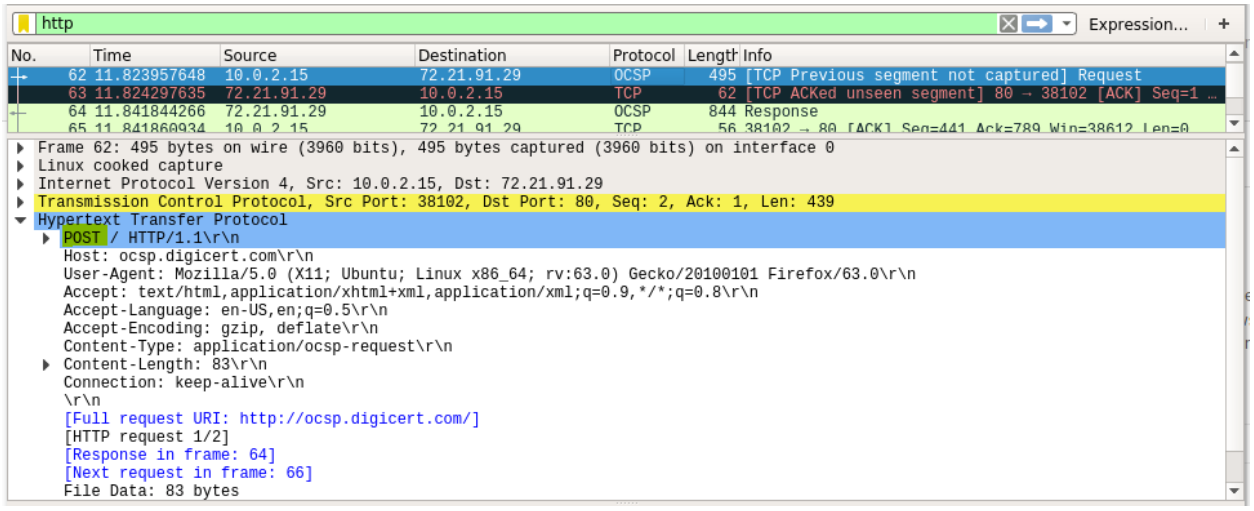
Capture 2



```
http

No.  Time          Source          Destination     Protocol Length Info
  21 0.128541651   10.0.2.15       216.58.195.228  TCP        56 42044 → 80 [ACK] Seq=142 Ack=8581 Win=45440 Len=0
  22 0.129006997   216.58.195.228  10.0.2.15       HTTP     3560 HTTP/1.1 200 OK  (text/html)
  23 0.129013053   10.0.2.15       216.58.195.228  TCP        56 42044 → 80 [ACK] Seq=142 Ack=12085 Win=52540 Len=0

▸ Internet Protocol Version 4, Src: 216.58.195.228, Dst: 10.0.2.15
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 42044, Seq: 8581, Ack: 142, Len: 3504
▸ [4 Reassembled TCP Segments (12084 bytes): #16(2840), #18(4260), #20(1480), #22(3504)]
▾ Hypertext Transfer Protocol
  ▸ HTTP/1.1 200 OK\r\n
    Date: Mon, 19 Nov 2018 03:19:47 GMT\r\n
    Expires: -1\r\n
    Cache-Control: private, max-age=0\r\n
    Content-Type: text/html; charset=ISO-8859-1\r\n
    P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."\r\n
    Server: gws\r\n
    X-XSS-Protection: 1; mode=block\r\n
    X-Frame-Options: SAMEORIGIN\r\n
    Set-Cookie: 1P_JAR=2018-11-19-03; expires=Wed, 19-Dec-2018 03:19:47 GMT; path=/; domain=.google.com\r\n
     [truncated]Set-Cookie: NID=146=BsOLUW0pssH2bDZ3qW5WZcOsP34EmP4Mi98I9cZy8pCBv00Tdnx4eyv1hFO8v4JssMUu5IkO5ZmM-avNHJkgFXqtIRd6w
    Accept-Ranges: none\r\n
    Vary: Accept-Encoding\r\n
    Transfer-Encoding: chunked\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.060196296 seconds]
    [Request in frame: 14]
  ▸ HTTP chunked response
    File Data: 11335 bytes
```

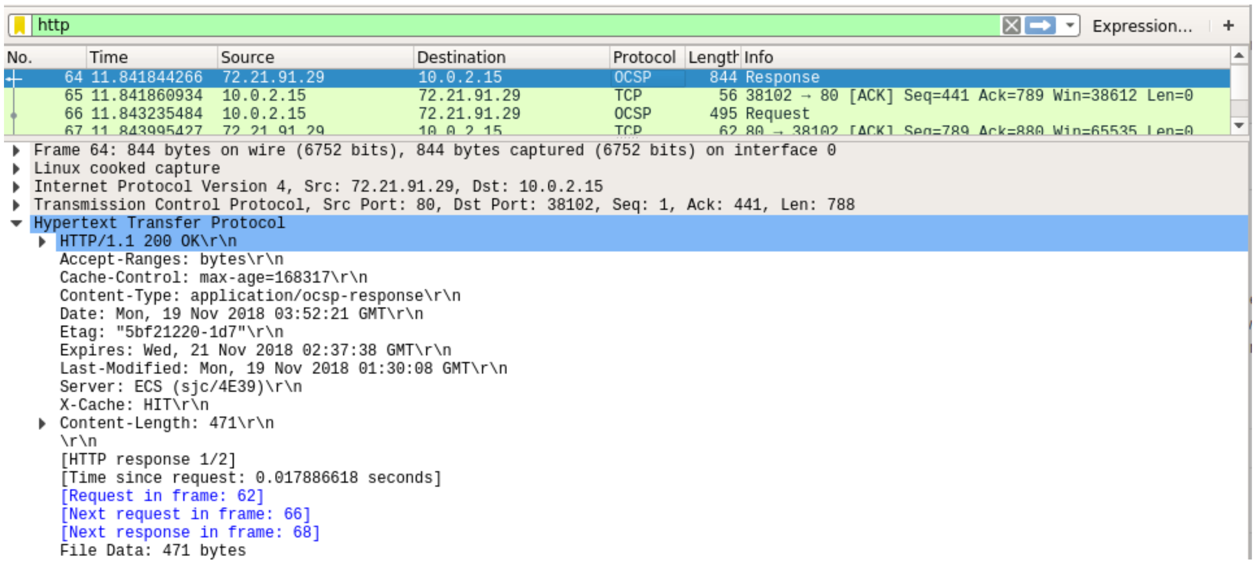
HTTP GET Method

Following capture shows the HTTP Post request made on a page using web browser. The Request message is highlighted in the screenshot below. Similar to GET request, POST request contains request line followed by headers and then the request message body.



```
http                                                                 Expression...  +
No.    Time          Source        Destination   Protocol Length Info
 62 11.823957648  10.0.2.15     72.21.91.29    OCSP      495 [TCP Previous segment not captured] Request
 63 11.824297635  72.21.91.29   10.0.2.15      TCP        62 [TCP ACKed unseen segment] 80 → 38102 [ACK] Seq=1 …
 64 11.841844266  72.21.91.29   10.0.2.15      OCSP      844 Response
 65 11.841860934  10.0.2.15     72.21.91.29    TCP        56 38102 → 80 [ACK] Seq=441 Ack=789 Win=38612 Len=0
▸ Frame 62: 495 bytes on wire (3960 bits), 495 bytes captured (3960 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 72.21.91.29
▸ Transmission Control Protocol, Src Port: 38102, Dst Port: 80, Seq: 2, Ack: 1, Len: 439
▾ Hypertext Transfer Protocol
  ▸ POST / HTTP/1.1\r\n
    Host: ocsp.digicert.com\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:63.0) Gecko/20100101 Firefox/63.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Content-Type: application/ocsp-request\r\n
  ▸ Content-Length: 83\r\n
    Connection: keep-alive\r\n
    \r\n
    [Full request URI: http://ocsp.digicert.com/]
    [HTTP request 1/2]
    [Response in frame: 64]
    [Next request in frame: 66]
    File Data: 83 bytes
```

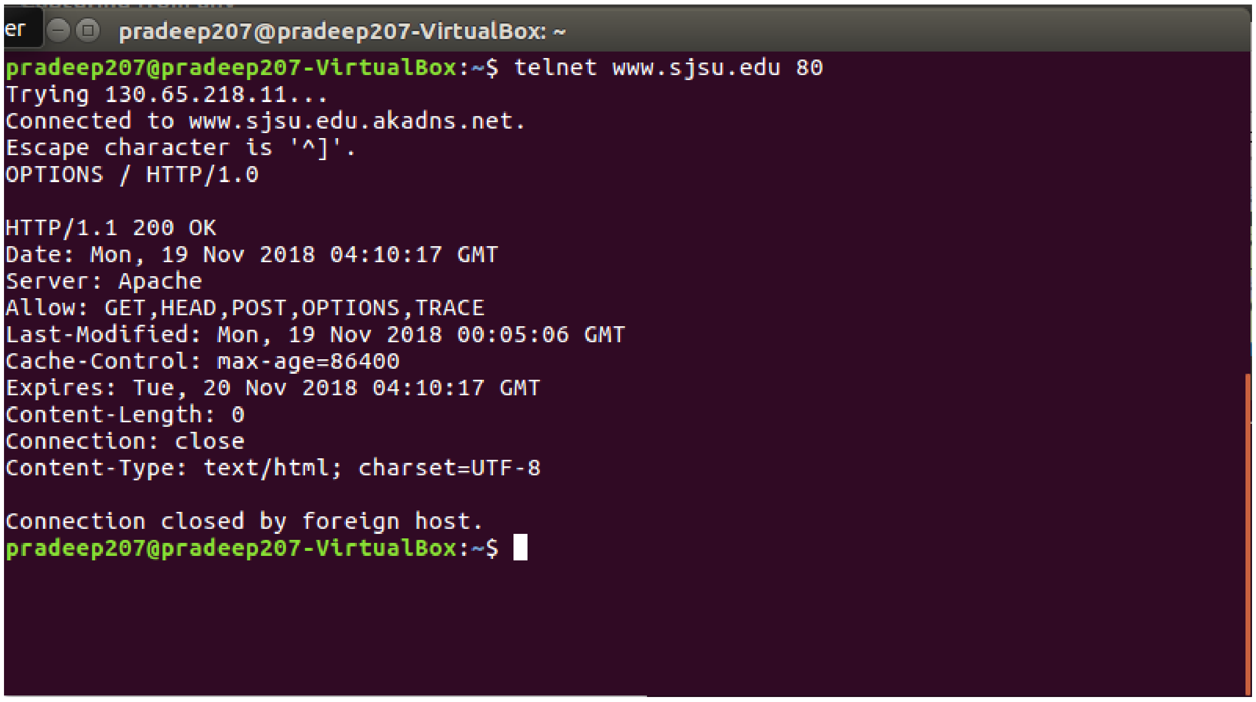
The following screenshot shows the response message for the POST request. As usual the message consists of request status line whisch says 200 OK. Indicating that the request was proper and the operation succeeded.



```
http                                                                 Expression...  +
No.    Time          Source        Destination   Protocol Length Info
 64 11.841844266  72.21.91.29   10.0.2.15      OCSP      844 Response
 65 11.841860934  10.0.2.15     72.21.91.29    TCP        56 38102 → 80 [ACK] Seq=441 Ack=789 Win=38612 Len=0
 66 11.843235484  10.0.2.15     72.21.91.29    OCSP      495 Request
 67 11.843995427  72.21.91.29   10.0.2.15      TCP        62 80 → 38102 [ACK] Seq=789 Ack=880 Win=65535 Len=0
▸ Frame 64: 844 bytes on wire (6752 bits), 844 bytes captured (6752 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 72.21.91.29, Dst: 10.0.2.15
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 38102, Seq: 1, Ack: 441, Len: 788
▾ Hypertext Transfer Protocol
  ▸ HTTP/1.1 200 OK\r\n
    Accept-Ranges: bytes\r\n
    Cache-Control: max-age=168317\r\n
    Content-Type: application/ocsp-response\r\n
    Date: Mon, 19 Nov 2018 03:52:21 GMT\r\n
    Etag: "5bf21220-1d7"\r\n
    Expires: Wed, 21 Nov 2018 02:37:38 GMT\r\n
    Last-Modified: Mon, 19 Nov 2018 01:30:08 GMT\r\n
    Server: ECS (sjc/4E39)\r\n
    X-Cache: HIT\r\n
  ▸ Content-Length: 471\r\n
    \r\n
    [HTTP response 1/2]
    [Time since request: 0.017886618 seconds]
    [Request in frame: 62]
    [Next request in frame: 66]
    [Next response in frame: 68]
    File Data: 471 bytes
```

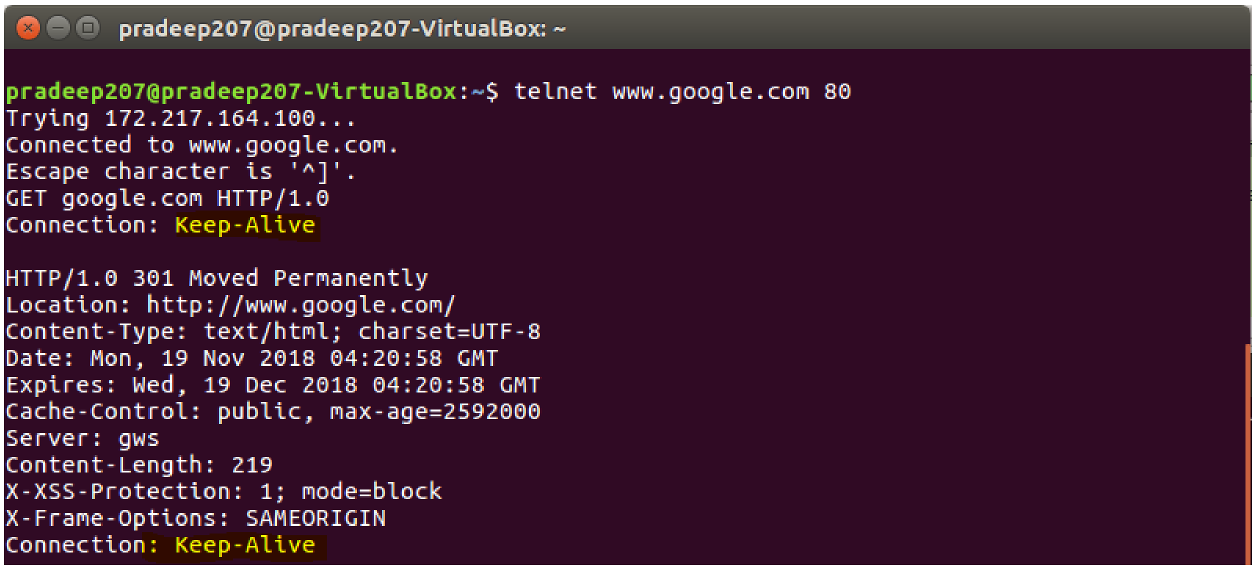
HTTP OPTIONS

Realization of http options is done with the help of telnet on a Linux machine.



```
pradeep207@pradeep207-VirtualBox:~$ telnet www.sjsu.edu 80
Trying 130.65.218.11...
Connected to www.sjsu.edu.akadns.net.
Escape character is '^]'.
OPTIONS / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 19 Nov 2018 04:10:17 GMT
Server: Apache
Allow: GET,HEAD,POST,OPTIONS,TRACE
Last-Modified: Mon, 19 Nov 2018 00:05:06 GMT
Cache-Control: max-age=86400
Expires: Tue, 20 Nov 2018 04:10:17 GMT
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8

Connection closed by foreign host.
pradeep207@pradeep207-VirtualBox:~$
```

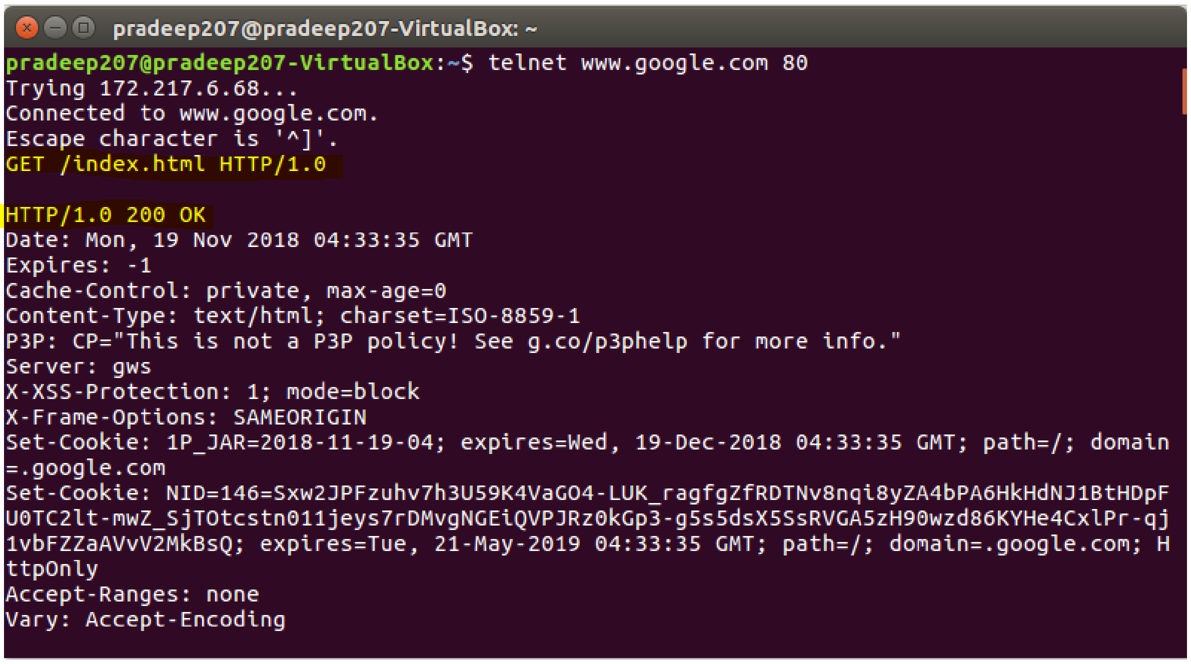
HTTP KEEP ALIVE



```
pradeep207@pradeep207-VirtualBox:~$ telnet www.google.com 80
Trying 172.217.164.100...
Connected to www.google.com.
Escape character is '^]'.
GET google.com HTTP/1.0
Connection: Keep-Alive

HTTP/1.0 301 Moved Permanently
Location: http://www.google.com/
Content-Type: text/html; charset=UTF-8
Date: Mon, 19 Nov 2018 04:20:58 GMT
Expires: Wed, 19 Dec 2018 04:20:58 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 219
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Connection: Keep-Alive
```

Realizations of HTTP Response Codes

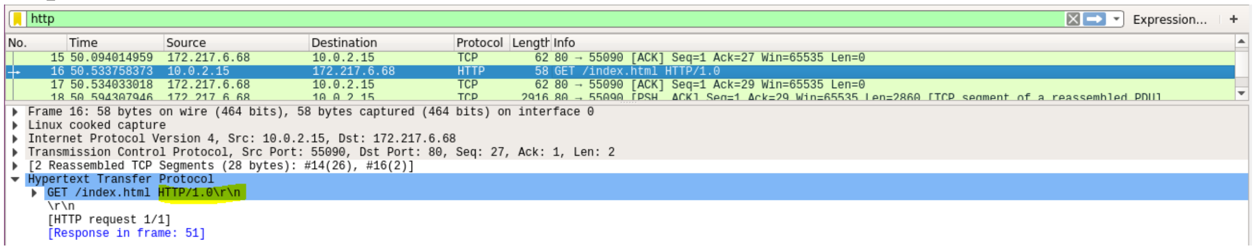
HTTP 200 OK



```
pradeep207@pradeep207-VirtualBox: ~
pradeep207@pradeep207-VirtualBox:~$ telnet www.google.com 80
Trying 172.217.6.68...
Connected to www.google.com.
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.0 200 OK
Date: Mon, 19 Nov 2018 04:33:35 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2018-11-19-04; expires=Wed, 19-Dec-2018 04:33:35 GMT; path=/; domain
=.google.com
Set-Cookie: NID=146=Sxw2JPFzuhv7h3U59K4VaGO4-LUK_ragfgZfRDTNv8nqi8yZA4bPA6HkHdNJ1BtHDpF
U0TC2lt-mwZ_SjTOtcstn011jeys7rDMvgNGEiQVPJRz0kGp3-g5s5dsX5SsRVGA5zH90wzd86KYHe4CxlPr-qj
1vbFZZaAVvV2MkBsQ; expires=Tue, 21-May-2019 04:33:35 GMT; path=/; domain=.google.com; H
ttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
```

Wireshark capture for 200 OK

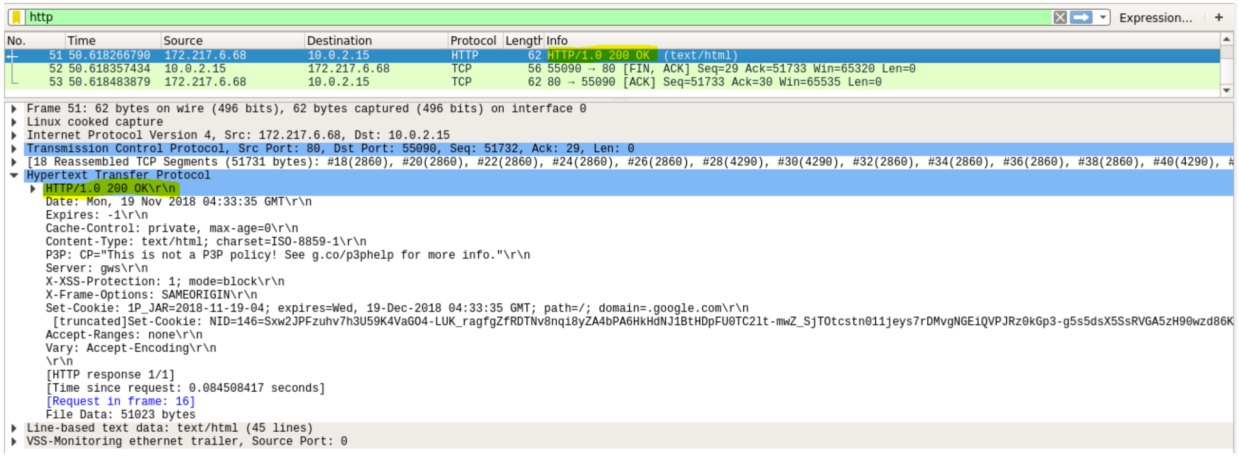
Request



```
No.  Time          Source        Destination   Protocol Length Info
  15 50.094014959  172.217.6.68  10.0.2.15     TCP        62 80 → 55090 [ACK] Seq=1 Ack=27 Win=65535 Len=0
  16 50.533758373  10.0.2.15     172.217.6.68  HTTP       58 GET /index.html HTTP/1.0
  17 50.534033018  172.217.6.68  10.0.2.15     TCP        62 80 → 55090 [ACK] Seq=1 Ack=29 Win=65535 Len=0
  18 50.594307946  172.217.6.68  10.0.2.15     TCP      2916 80 → 55090 [PSH, ACK] Seq=1 Ack=29 Win=65535 Len=2860 [TCP segment of a reassembled PDU]
```

```
▸ Frame 16: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 172.217.6.68
▸ Transmission Control Protocol, Src Port: 55090, Dst Port: 80, Seq: 27, Ack: 1, Len: 2
▸ [2 Reassembled TCP Segments (28 bytes): #14(26), #16(2)]
▾ Hypertext Transfer Protocol
  ▸ GET /index.html HTTP/1.0\r\n
    \r\n
    [HTTP request 1/1]
    [Response in frame: 51]
```

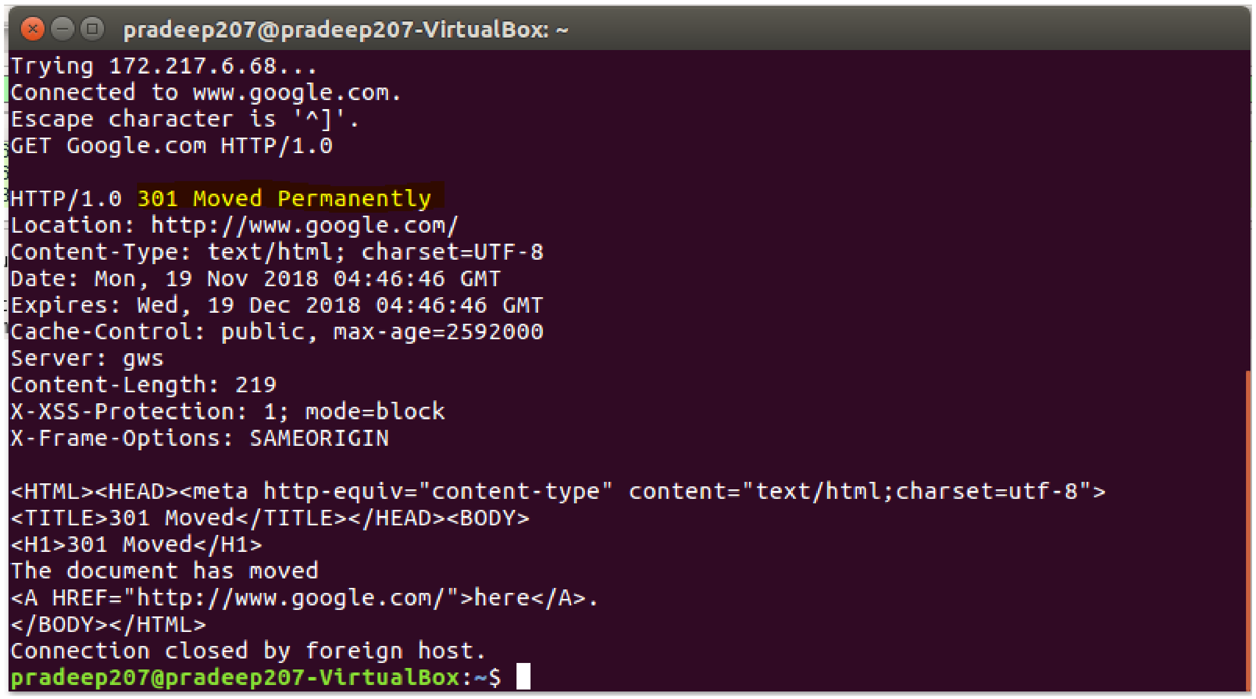
Response



```
No.  Time          Source        Destination   Protocol Length Info
  51 50.618266790  172.217.6.68  10.0.2.15     HTTP       62 HTTP/1.0 200 OK  (text/html)
  52 50.618357434  10.0.2.15     172.217.6.68  TCP        56 55090 → 80 [FIN, ACK] Seq=29 Ack=51733 Win=65320 Len=0
  53 50.618483879  172.217.6.68  10.0.2.15     TCP        62 80 → 55090 [ACK] Seq=51733 Ack=30 Win=65535 Len=0
```

```
▸ Frame 51: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 172.217.6.68, Dst: 10.0.2.15
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 55090, Seq: 51732, Ack: 29, Len: 0
▸ [18 Reassembled TCP Segments (51731 bytes): #18(2860), #20(2860), #22(2860), #24(2860), #26(2860), #28(4290), #30(4290), #32(2860), #34(2860), #36(2860), #38(2860), #40(4290), #
▾ Hypertext Transfer Protocol
  ▸ HTTP/1.0 200 OK\r\n
    Date: Mon, 19 Nov 2018 04:33:35 GMT\r\n
    Expires: -1\r\n
    Cache-Control: private, max-age=0\r\n
    Content-Type: text/html; charset=ISO-8859-1\r\n
    P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."\r\n
    Server: gws\r\n
    X-XSS-Protection: 1; mode=block\r\n
    X-Frame-Options: SAMEORIGIN\r\n
    Set-Cookie: 1P_JAR=2018-11-19-04; expires=Wed, 19-Dec-2018 04:33:35 GMT; path=/; domain=.google.com\r\n
     [truncated]Set-Cookie: NID=146=Sxw2JPFzuhv7h3U59K4VaGO4-LUK_ragfgZfRDTNv8nqi8yZA4bPA6HkHdNJ1BtHDpFU0TC2lt-mwZ_SjTOtcstn011jeys7rDMvgNGEiQVPJRz0kGp3-g5s5dsX5SsRVGA5zH90wzd86K
    Accept-Ranges: none\r\n
    Vary: Accept-Encoding\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.084508417 seconds]
    [Request in frame: 16]
    File Data: 51023 bytes
▸ Line-based text data: text/html (45 lines)
▸ VSS-Monitoring ethernet trailer, Source Port: 0
```

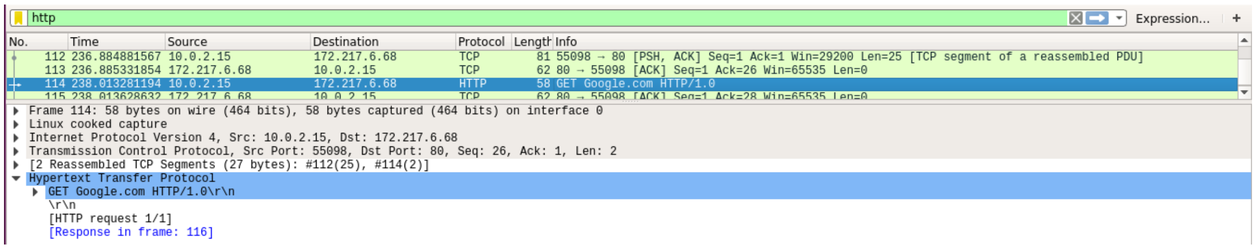
HTTP 301 Moved



```
Trying 172.217.6.68...
Connected to www.google.com.
Escape character is '^]'.
GET Google.com HTTP/1.0

HTTP/1.0 301 Moved Permanently
Location: http://www.google.com/
Content-Type: text/html; charset=UTF-8
Date: Mon, 19 Nov 2018 04:46:46 GMT
Expires: Wed, 19 Dec 2018 04:46:46 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 219
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
Connection closed by foreign host.
pradeep207@pradeep207-VirtualBox:~$
```

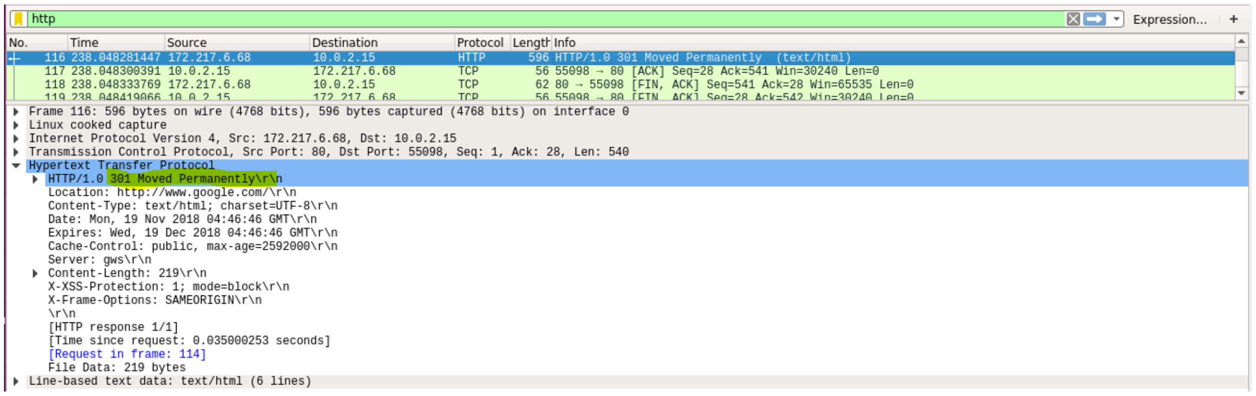
Wireshark capture of 301 Moved

Request



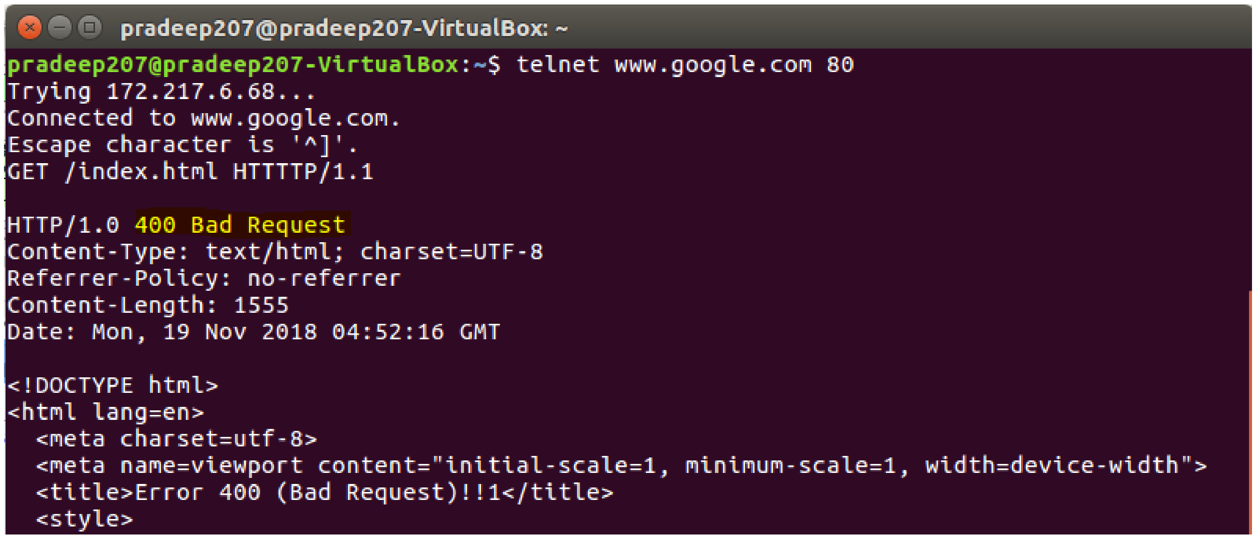
```
No.   Time           Source         Destination    Protocol Length Info
  112 236.884881567 10.0.2.15      172.217.6.68   TCP        81 55098 → 80 [PSH, ACK] Seq=1 Ack=1 Win=29200 Len=25 [TCP segment of a reassembled PDU]
  113 236.885331854 172.217.6.68   10.0.2.15      TCP        62 80 → 55098 [ACK] Seq=1 Ack=26 Win=65535 Len=0
  114 238.013281194 10.0.2.15      172.217.6.68   HTTP       58 GET Google.com HTTP/1.0
  115 238.013628632 172.217.6.68   10.0.2.15      TCP        62 80 → 55098 [ACK] Seq=1 Ack=28 Win=65535 Len=0
Frame 114: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 172.217.6.68
Transmission Control Protocol, Src Port: 55098, Dst Port: 80, Seq: 26, Ack: 1, Len: 2
[2 Reassembled TCP Segments (27 bytes): #112(25), #114(2)]
Hypertext Transfer Protocol
  GET Google.com HTTP/1.0\r\n
  \r\n
  [HTTP request 1/1]
  [Response in frame: 116]
```

Response



```
No.   Time           Source         Destination    Protocol Length Info
  116 238.048281447 172.217.6.68   10.0.2.15      HTTP      596 HTTP/1.0 301 Moved Permanently  (text/html)
  117 238.048300391 10.0.2.15      172.217.6.68   TCP        56 55098 → 80 [ACK] Seq=28 Ack=541 Win=30240 Len=0
  118 238.048333769 172.217.6.68   10.0.2.15      TCP        62 80 → 55098 [FIN, ACK] Seq=541 Ack=28 Win=65535 Len=0
  119 238.048419066 10.0.2.15      172.217.6.68   TCP        56 55098 → 80 [FIN, ACK] Seq=28 Ack=542 Win=30240 Len=0
Frame 116: 596 bytes on wire (4768 bits), 596 bytes captured (4768 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 172.217.6.68, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 80, Dst Port: 55098, Seq: 1, Ack: 28, Len: 540
Hypertext Transfer Protocol
  HTTP/1.0 301 Moved Permanently\r\n
  Location: http://www.google.com/\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  Date: Mon, 19 Nov 2018 04:46:46 GMT\r\n
  Expires: Wed, 19 Dec 2018 04:46:46 GMT\r\n
  Cache-Control: public, max-age=2592000\r\n
  Server: gws\r\n
  Content-Length: 219\r\n
  X-XSS-Protection: 1; mode=block\r\n
  X-Frame-Options: SAMEORIGIN\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.035000253 seconds]
  [Request in frame: 114]
  File Data: 219 bytes
Line-based text data: text/html (6 lines)
```

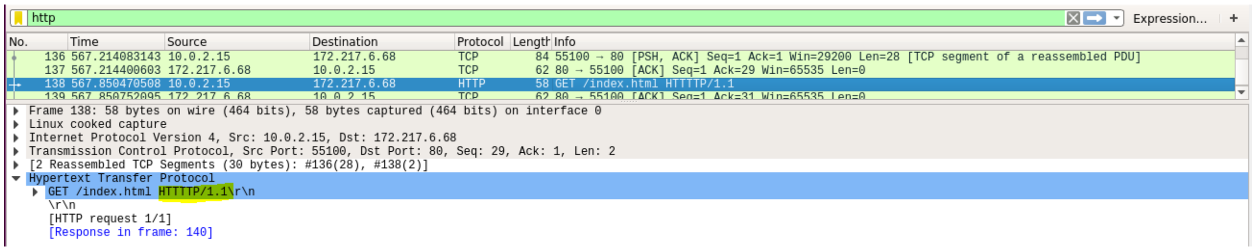
HTTP 400 Bad Request



```
pradeep207@pradeep207-VirtualBox: ~
pradeep207@pradeep207-VirtualBox:~$ telnet www.google.com 80
Trying 172.217.6.68...
Connected to www.google.com.
Escape character is '^]'.
GET /index.html HTTTTP/1.1

HTTP/1.0 400 Bad Request
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Content-Length: 1555
Date: Mon, 19 Nov 2018 04:52:16 GMT

<!DOCTYPE html>
<html lang=en>
  <meta charset=utf-8>
  <meta name=viewport content="initial-scale=1, minimum-scale=1, width=device-width">
  <title>Error 400 (Bad Request)!!1</title>
  <style>
```

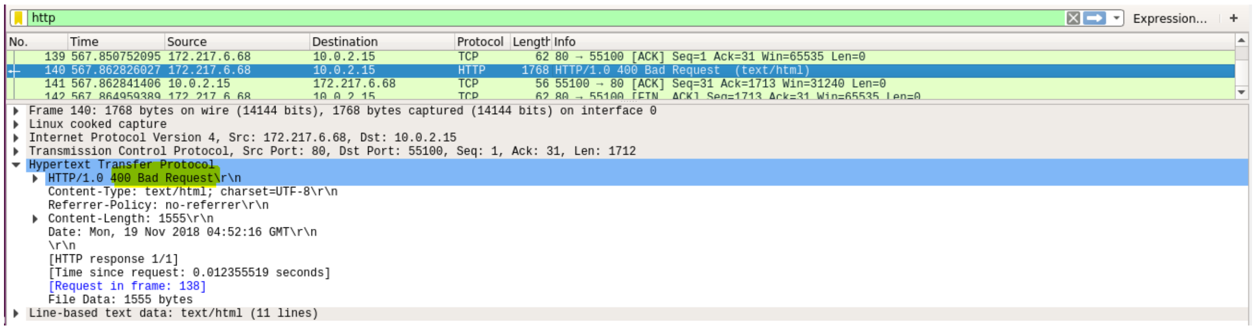
Wireshark captures

Request



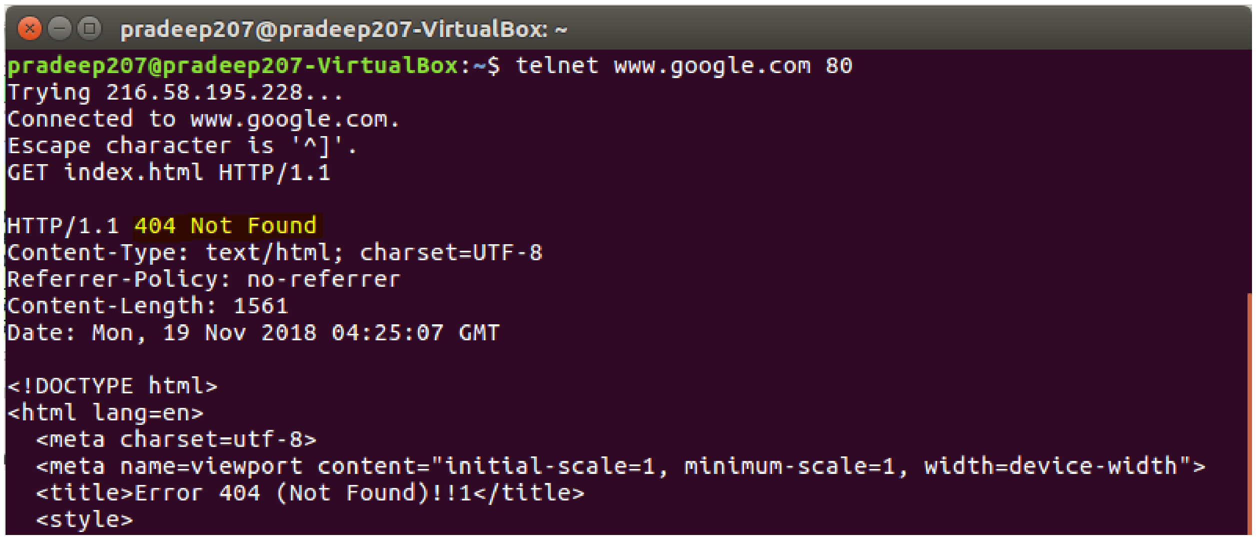
```
http
No.  Time           Source        Destination   Protocol Length Info
136 567.214083143 10.0.2.15     172.217.6.68   TCP      84 55100 → 80 [PSH, ACK] Seq=1 Ack=1 Win=29200 Len=28 [TCP segment of a reassembled PDU]
137 567.214400603 172.217.6.68  10.0.2.15      TCP      62 80 → 55100 [ACK] Seq=1 Ack=29 Win=65535 Len=0
138 567.850470508 10.0.2.15     172.217.6.68   HTTP     58 GET /index.html HTTTTP/1.1
139 567.850752095 172.217.6.68  10.0.2.15      TCP      62 80 → 55100 [ACK] Seq=1 Ack=31 Win=65535 Len=0

Frame 138: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 172.217.6.68
Transmission Control Protocol, Src Port: 55100, Dst Port: 80, Seq: 29, Ack: 1, Len: 2
[2 Reassembled TCP Segments (30 bytes): #136(28), #138(2)]
Hypertext Transfer Protocol
  GET /index.html HTTTTP/1.1\r\n
  \r\n
  [HTTP request 1/1]
  [Response in frame: 140]
```

Response



```
http
No.  Time           Source        Destination   Protocol Length Info
139 567.850752095 172.217.6.68  10.0.2.15      TCP      62 80 → 55100 [ACK] Seq=1 Ack=31 Win=65535 Len=0
140 567.862826027 172.217.6.68  10.0.2.15      HTTP   1768 HTTP/1.0 400 Bad Request  (text/html)
141 567.862841406 10.0.2.15     172.217.6.68   TCP      56 55100 → 80 [ACK] Seq=31 Ack=1713 Win=31240 Len=0
142 567.864959389 172.217.6.68  10.0.2.15      TCP      62 80 → 55100 [FIN, ACK] Seq=1713 Ack=31 Win=65535 Len=0

Frame 140: 1768 bytes on wire (14144 bits), 1768 bytes captured (14144 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 172.217.6.68, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 80, Dst Port: 55100, Seq: 1, Ack: 31, Len: 1712
Hypertext Transfer Protocol
  HTTP/1.0 400 Bad Request\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  Referrer-Policy: no-referrer\r\n
  Content-Length: 1555\r\n
  Date: Mon, 19 Nov 2018 04:52:16 GMT\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.012355519 seconds]
  [Request in frame: 138]
  File Data: 1555 bytes
Line-based text data: text/html (11 lines)
```

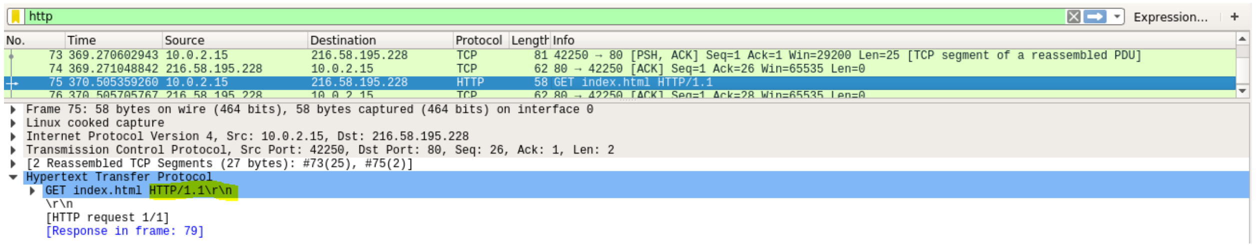
HTTP 404 Not Found



```
pradeep207@pradeep207-VirtualBox:~$ telnet www.google.com 80
Trying 216.58.195.228...
Connected to www.google.com.
Escape character is '^]'.
GET index.html HTTP/1.1

HTTP/1.1 404 Not Found
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Content-Length: 1561
Date: Mon, 19 Nov 2018 04:25:07 GMT

<!DOCTYPE html>
<html lang=en>
  <meta charset=utf-8>
  <meta name=viewport content="initial-scale=1, minimum-scale=1, width=device-width">
  <title>Error 404 (Not Found)!!1</title>
  <style>
```

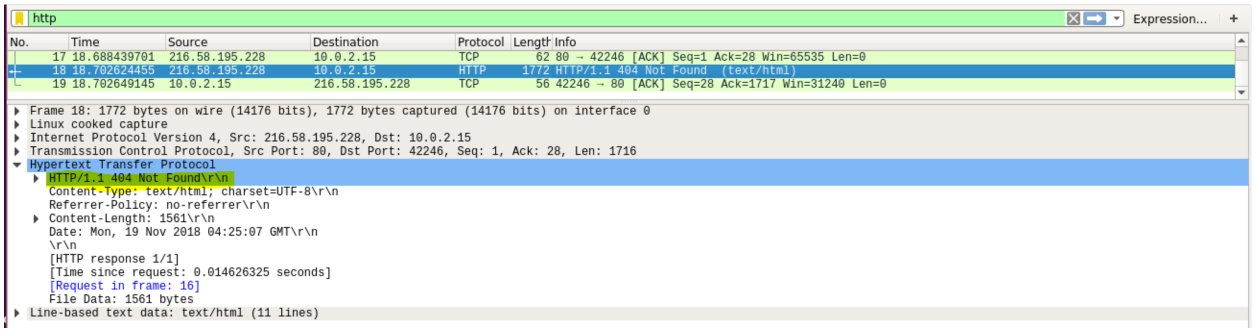
Wireshark capture for 404 not found

Request



```
No.  Time           Source          Destination     Protocol Length Info
  73 369.270602943 10.0.2.15       216.58.195.228  TCP       81 42250 → 80 [PSH, ACK] Seq=1 Ack=1 Win=29200 Len=25 [TCP segment of a reassembled PDU]
  74 369.271048842 216.58.195.228  10.0.2.15       TCP       62 80 → 42250 [ACK] Seq=1 Ack=26 Win=65535 Len=0
  75 370.505359260 10.0.2.15       216.58.195.228  HTTP      58 GET index.html HTTP/1.1
  76 370.505705767 216.58.195.228  10.0.2.15       TCP       62 80 → 42250 [ACK] Seq=1 Ack=28 Win=65535 Len=0

Frame 75: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 216.58.195.228
Transmission Control Protocol, Src Port: 42250, Dst Port: 80, Seq: 26, Ack: 1, Len: 2
[2 Reassembled TCP Segments (27 bytes): #73(25), #75(2)]
Hypertext Transfer Protocol
  GET index.html HTTP/1.1\r\n
  \r\n
  [HTTP request 1/1]
  [Response in frame: 79]
```

Response



```
No.  Time          Source          Destination     Protocol Length Info
  17 18.688439701  216.58.195.228  10.0.2.15       TCP       62 80 → 42246 [ACK] Seq=1 Ack=28 Win=65535 Len=0
  18 18.702624455  216.58.195.228  10.0.2.15       HTTP    1772 HTTP/1.1 404 Not Found  (text/html)
  19 18.702649145  10.0.2.15       216.58.195.228  TCP       56 42246 → 80 [ACK] Seq=28 Ack=1717 Win=31240 Len=0

Frame 18: 1772 bytes on wire (14176 bits), 1772 bytes captured (14176 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 216.58.195.228, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 80, Dst Port: 42246, Seq: 1, Ack: 28, Len: 1716
Hypertext Transfer Protocol
  HTTP/1.1 404 Not Found\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  Referrer-Policy: no-referrer\r\n
  Content-Length: 1561\r\n
  Date: Mon, 19 Nov 2018 04:25:07 GMT\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.014626325 seconds]
  [Request in frame: 16]
  File Data: 1561 bytes
Line-based text data: text/html (11 lines)
```

Conclusion

In this lab we obtained practical and detail understating of HTTP. Discussed the operation of HTTP, took a detailed look at all the HTTP Messages, Request/Response format and looked at some of the Advantages of using HTTP. We took a brief overview at HTTP security risks and their preventions. We also executed the HTTP LAB and recorded verious HTTP requests and responses.

Contributions

Charit Upadhyay

- Command Execution and Observation
- Troubleshooting network topology
- Wireshark observation
- Equal contribution and learning on all aspects

Devika Jadhav

- HTTP message format
- HTTP Architecture
- Documentation and report formatting
- Observations on Wireshark
- Equal contribution and learning on all aspects

Pradeep Patil

- Documentation and report formatting
- HTTP Introduction
- HTTP Architecture
- Screenshots and command troubleshooting
- Equal contribution and learning on all aspects

References / Links

- https://tools.ietf.org/html/rfc792
- https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf
- https://www.tutorialspoint.com/http/http_tutorial.pdf
- https://www.techwalla.com/articles/what-do-the-parts-of-a-web-address-mean
- https://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html
- http://searchnetworking.techtarget.com/definition/DHCP
- https://technet.microsoft.com/en-us/library/cc781008(v=ws.10).aspx

- https://technet.microsoft.com/en-us/library/cc780760(v=ws.10).aspx
- Benefits https://itstillworks.com/benefits-internet-control-message-protocol-6742787.html
- https://www.ipv6.com/general/icmpv6-tech-details-advantages/
- https://tools.ietf.org/html/rfc2131
- https://www.lifewire.com/what-is-dhcp-2625848
- https://www.thegeekstuff.com/2013/03/dhcp-basics/
- http://www.omnisecu.com/tcpip/dhcp-dynamic-host-configuration-protocol-how-dhcp-works.php
- https://www.cisco.com/c/en/us/support/docs/ip/dynamic-address-allocation-resolution/27470-100.html
- GNS: https://www.gns3.com/
- Wireshark: https://www.wireshark.org/
- ISC DHCP Server https://help.ubuntu.com/community/isc-dhcp-server/
- http://www.enterprisenetworkingplanet.com/netsp/article.php/3584166/Networking-101-Understanding-and-Using-ICMP.htm
- http://www.rhyshaden.com/icmp.htm
- Cisco Basic IOS commands: https://www.cisco.com/c/en/us/td/docs/ios/12_2/configfun/command/reference/ffun_r/frf001.html