PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Rodrigo Andrés Henríquez Bustamante

Descripción del protocolo HTTP y análisis de sus transferencias mediante metalenguajes

Informe Proyecto de Título de Ingeniero Electrónico

Escuela de Ingeniería Eléctrica

DESCRIPCION DEL PROTOCOLO HTTP Y ANALISIS DE SUS TRANSFERENCIAS MEDIANTE METALENGUAJES

Rodrigo Andrés Henríquez Bustamante

Informe Final para optar al título de Ingeniero Electrónico,

aprobada por la comisión de la

Escuela de Ingeniería Eléctrica de la

Pontificia Universidad Católica de Valparaíso

conformada por

Sr. David Velasco López

Profesor Guía

Sr. Francisco Alonso Villalobos

Segundo Revisor

Sr. Jorge Mendoza Baeza

Secretario Académico

Valparaíso, 06 de junio de 2017

Resumen

Este informe trata sobre el protocolo HTTP, el cual destaca en el área de la tecnología de la información y las comunicaciones, un protocolo de comunicaciones consiste en un conjunto de reglas que permiten que dos o más entidades se comuniquen entre si para transmitir información. Son las reglas que definen la sintaxis, la semántica y el momento de la comunicación entre las entidades involucradas.

Con este proyecto se da a entender como opera el protocolo HTTP y sus respectivas aplicaciones en el mundo moderno, y queda en evidencia que es un protocolo de vital importancia ya que por medio de él se abren todas las páginas de Internet. No obstante esto involucra un proceso de por medio, y en este informe se describe ese proceso.

Para poder aplicar este protocolo del proyecto al mundo real se desarrollará una experiencia práctica basada en los procesos de comunicación entre usuario y servidor que son lo que da vida al protocolo HTTP al abrir cualquier página web, utilizando también herramientas de metalenguajes para describir las opciones presentes en las transferencias de información entre usuario y servidor, para posteriormente crear dicha comunicación mostrando todo el contenido de ella.

Palabras claves: Protocolo HTTP, página web, comunicación cliente-servidor, metalenguajes.

Abstract

This report deals with the HTTP protocol, which stands out in the area of information and communication technology, a communications protocol consists of a set of rules that allow two or more entities to communicate with each other to transmit information. They are the rules that define the syntax, the semantics and the moment of communication between the involved entities.

With this project it is understood how the HTTP protocol and its respective applications operate in the modern world, and it is evident that it is a protocol of vital importance, since through it all the Internet pages are opened. However this involves a process through, and this report describes that process..

To be able to apply this protocol of the project to the real world will be developed a practical experience based on the processes of communication between user and server that are what gives life to the HTTP protocol when opening any web page, also using tools of metalanguages to describe the present options In the transfers of information between user and server, to later create said communication showing all the content of it..

Keywords: HTTP protocol, web page, client-server communication, metalanguages.

Índice general

INDICE DE FIGURAS

INDICE DE TABLAS

GLOSARIO DE TERMINOS

HTTP: Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertexto).

URL: Localizador Uniforme de Recursos.

HTML: Hypertext Markup Language (Lenguaje de marcas de hipertexto).

MIME: Multipurpose Internet Mail Extensions (Extensiones multipropósito de correos de internet).

RFC: Request For Comments (Petición de comentarios).

TCP: Transmission Control Protocol (Protocolo de control de transmisión).

IP: Internet Protocol (Protocolo de internet).

W3C: World Wide Web Consortium (Consorcio WWW).

IETF: Internet Engineering Task Force (Grupo de Trabajo de Ingeniería de Internet).

ASCII: American Standard Code for Information Interchange (Código Estándar Estadounidense para el Intercambio de Información).

XST: Cross-Site Tracing (Rastreo de sitios cruzados).

Introducción

Cada vez que las personas se conectan a Internet, comienzan a buscar información de determinados temas en diversas páginas web, no obstante se desconoce como el computador lleva a cabo dicho proceso y la gente piensa que mágicamente se abren las páginas web en Internet. Este proceso de conectarse a Internet y abrir diferentes páginas web involucra un gran proceso interno del computador, ya que se llevan a cabo grandes transferencias de paquetes de información mediante un proceso de comunicación entre el usuario y el computador, un proceso que se conoce como comunicación cliente-servidor, el cliente pasa a ser el usuario o sea la persona que ocupa el computador y se conecta a Internet. El servidor es toda una estructura de programación interna que posee el computador con la cual procesa la información que recibe del usuario y le entrega la información ya sea mostrándole la página web solicitada o también mostrándole que no es posible conectarse a dicha página. Este proceso de comunicación cliente-servidor al conectarse a Internet se lleva a cabo mediante el denominado protocolo HTTP (Protocolo de transferencia de Hipertexto), el cual pertenece a una amplia gama de protocolos de redes de computadores, en los cuales mediante determinadas características y normas se llevan a cabo determinados procesos de comunicación entre cliente y servidor. En lo que respecta al protocolo HTTP en este informe se detallará como opera el protocolo HTTP, sus características y funcionamiento, y cuáles son las normas que rigen este protocolo con sus distintas versiones, además del estudio de metalenguajes, los cuales servirán para analizar la estructura sintáctica de cada una de las etapas de comunicación entre el cliente y el servidor llevadas a cabo mediante el protocolo HTTP, en lo que se conoce como transacciones HTTP, es decir, transferencias de paquetes de datos durante la comunicación entre el cliente y el servidor al abrir una página web. En estas transferencias de información entre el usuario y el servidor, el usuario le pide al servidor que entregue determinada información de una página web, y el servidor responde mostrando la página. No obstante dichas transferencias tienen un contenido propio de información, y en lo que respecta al usuario esto se conoce como métodos de petición del usuario al servidor, y en lo que respecta al servidor se conoce como códigos de respuesta del servidor al cliente (usuario), además de que ya sea si la transferencia de información es de petición del usuario al servidor, o de respuesta del servidor al cliente, también tiene en su contenido información adicional sobre más características de dicha transferencia como lo es el tipo de contenido del documento, el largo del contenido, el idioma permitido, el tipo de

servidor que se utiliza. Además cada tipo de transferencia que se realice, de acuerdo a las características que posea tiene su propia documentación, y esta documentación también describe los campos y la estructura interna de cada tipo de información relacionada con las características de la transferencia, y en esa descripción de la estructura de la transferencia es que se ocupará lo que se define como metalenguaje, que no es más que otra forma de describir un determinado tipo de lenguaje, es decir, herramientas que permitirán describir determinado tipo de lenguajes.

Habiendo nombrado todas estas características relacionadas con el protocolo HTTP, es que se tiene como base en este proyecto el estudio y descripción del protocolo HTTP. Se pretende realizar una documentación del funcionamiento del protocolo HTTP y realizar un determinada aplicación al mundo real, es así como se utilizarán determinadas herramientas que permitan analizar con mayor profundidad el funcionamiento y aplicación del protocolo HTTP para crear un servidor HTTP o servidor web, y poder aplicar las herramientas y conceptos adquiridos durante el análisis del protocolo para conseguir el objetivo esperado, esto se hará con los equipos que se encuentran en el laboratorio de Sistemas Digitales C de la Escuela de Ingeniería Eléctrica de la Universidad Católica de Valparaíso .

Para terminar lo que respecta a este informe, es que también se realizará un análisis económico del proyecto, estudiando las distintas variables económicas involucradas en el proyecto con sus respectivos costos para poder realizar una inversión inicial del proyecto, la cual después pueda dar frutos de cara al futuro, lo que permita desarrollar el proyecto en años de servicio en una empresa y determinar si el proyecto de cara al futuro presentará ganancias o pérdidas, lo cual desde luego será el factor clave que permitirá saber si es que el proyecto es o no rentable de cara al futuro para implantarlo en una empresa, de acuerdo a los ingresos y costos que tenga involucrado este proyecto.

1 Descripción General

1.1 Descripción del Proyecto

En este proyecto se pretende estudiar en profundidad las características y el funcionamiento del protocolo HTTP, sus distintas aplicaciones al mundo real y como se pueden describir cada una de las etapas de su operación en base a herramientas de análisis como esquemas de planificación, normas de su descripción de etapas de su funcionamiento, y programas para el análisis de su operación y funcionamiento.

1.2 Objetivos

Este proyecto tiene definidos determinados objetivos:

1.2.1 Objetivo General

Estudio del protocolo HTTP, de sus respectivas características y funcionamiento, basándose en las transacciones de información entre cliente y servidor mediante la conexión que se lleva a cabo con este protocolo, para posteriormente crear un servidor de página con sistema operativo propietario.

1.2.2 Objetivos Específicos

- Descripción de las transferencias y transacciones que ocurren mediante la aplicación del protocolo HTTP.

- Estudio de metalenguajes y su aplicación al protocolo HTTP.

- Establecer esquemas de comunicación entre cliente y servidor, que permitan analizar más profundamente la transferencia de paquetes de datos presentes en el protocolo.

- Utilizar el SNIFFER, como una herramienta de análisis de las transacciones de informaciones entre cliente y servidor presentes en el protocolo HTTP.

- Crear un servidor de página (servidor HTTP) en sistema operativo propietario, utilizando las herramientas mencionadas en el análisis del protocolo.

1.3 Estado del Arte

Para la recopilación de información y de antecedentes de otros proyectos que puedan servir de base para este proyecto se utilizará como base fundamental de cara a crear el servidor HTTP, el proyecto del alumno Matías Castillo Felmer, el cual se encargó de hacer un proyecto basado en crear un servidor DHCP, el cual servirá de cara a crear el servidor HTTP, ya que con esto se tendrán asignadas las direcciones IP de referencia para el usuario por el servidor DHCP del sistema operativo propietario, y con esto se tendrán las referencias para poder establecer distintos parámetros para el proceso de comunicación cliente-servidor en el cual se basa el protocolo HTTP, de acuerdo a determinadas etapas de petición por parte del usuario al servidor, y un correspondiente código de respuesta del servidor al cliente indicando como se llevó a cabo la solicitud si fue o no exitosa, y que tipo de error hay en la comunicación. Otro proyecto que también sirve de base es el del alumno Rodrigo Gutiérrez Rojas, el cual se encargó de crear un servidor de tipo SIP, el cual resultó importante para crear el servidor HTTP, ya que con este proyecto quedaron establecidos ciertos tipos de variables con su respectivo formato, es decir, si el tipo de variable que se quiere declarar en la programación para crear el servidor HTTP es de tipo bit, byte o también de tipo string, la cual consiste en cadenas de caracteres y de letras que se permiten utilizar en sistema operativo propietario, ya que fueron establecidas con el servidor SIP. Teniendo como base aquellos proyectos anteriores para desarrollar este proyecto, es que se pretende crear un servidor HTTP en sistema operativo propietario, el cual servirá también como base importante para desarrollar otros proyectos a futuro en que también tengan involucrado el protocolo HTTP, o simplemente requieran del servidor HTTP desarrollado en este proyecto para poder llevar a cabo otro tipo de proyectos.

2 Protocolo HTTP

2.1 Introducción al Protocolo HTTP

HTTP es una sigla que significa HyperText Transfer Protocol, o Protocolo de Transferencia de Hipertexto. Es un protocolo de transferencia de hipertexto que se usa en la web. Este protocolo fue desarrollado por las instituciones internacionales W3C y IETF y se usa en todo tipo de transacciones (transferencias) por medio de Internet.

El protocolo HTTP facilita el proceso de comunicación en que se ven envueltos los distintos softwares web, tanto clientes, como servidores y proxis, para interactuar entre sí. [1]

Este protocolo opera en base a un proceso de comunicación cliente-servidor de acuerdo a petición y respuesta. A menudo las peticiones tienen que ver con archivos, ejecución de un programa, consulta a una base de datos, traducción y otras funciones. Toda la información que opera en la web mediante este protocolo es identificada mediante la URL o dirección.

Como este protocolo opera en base a transacciones de información entre el cliente y el servidor, la transacción básica del protocolo HTTP se compone de un encabezado, seguido por una línea en blanco y luego un dato. Este encabezado define la acción requerida por el servidor. [1]

Desde su creación, el protocolo HTTP ha pasado por distintas versiones, como lo son 0.9, 1.0, 1.1, 1.2 y 2.0. [1]

Este protocolo trabaja en base a códigos de respuesta de tres dígitos, que comunican si la conexión fue rechazada, si se realizó con éxito, si ha sido redirigida hacia otro URL, si existe un error por parte del cliente, o bien, un error por parte del servidor.

Hoy en día, muchas de las direcciones de URL requieren la inclusión del protocolo "http://" para su correcto funcionamiento. Este protocolo es usualmente seguido del clásico código "www" y luego por la dirección específica del sitio web que se desea visitar.

2.2 Introducción a HTML

HTML, sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la creación de páginas web. Define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del World Wide Web Consortium (W3C) o Consorcio WWW, la cual es una organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo que hace referencia a su escritura e interpretación. Se considera el lenguaje web más importante siendo fundamental su creación para el desarrollo y avance de la World Wide Web (WWW). Es el lenguaje estándar que se ha establecido para visualizar las páginas web y es el lenguaje que todos los navegadores actuales han adoptado. [10]

Cuando se accede a una página web es posible observar su correspondiente código HTML como código fuente de la página, haciendo click en el lado derecho del mouse, como se aprecia en la figura 2.1:

Fig. 2.1 Visualización de página web con opciones. (Fuente: www.latercera.com)

Una vez que se accede al código fuente de la página se abrirá una nueva ventana, correspondiente al código HTML de la página web visitada, lo cual se aprecia en la figura 2.2:

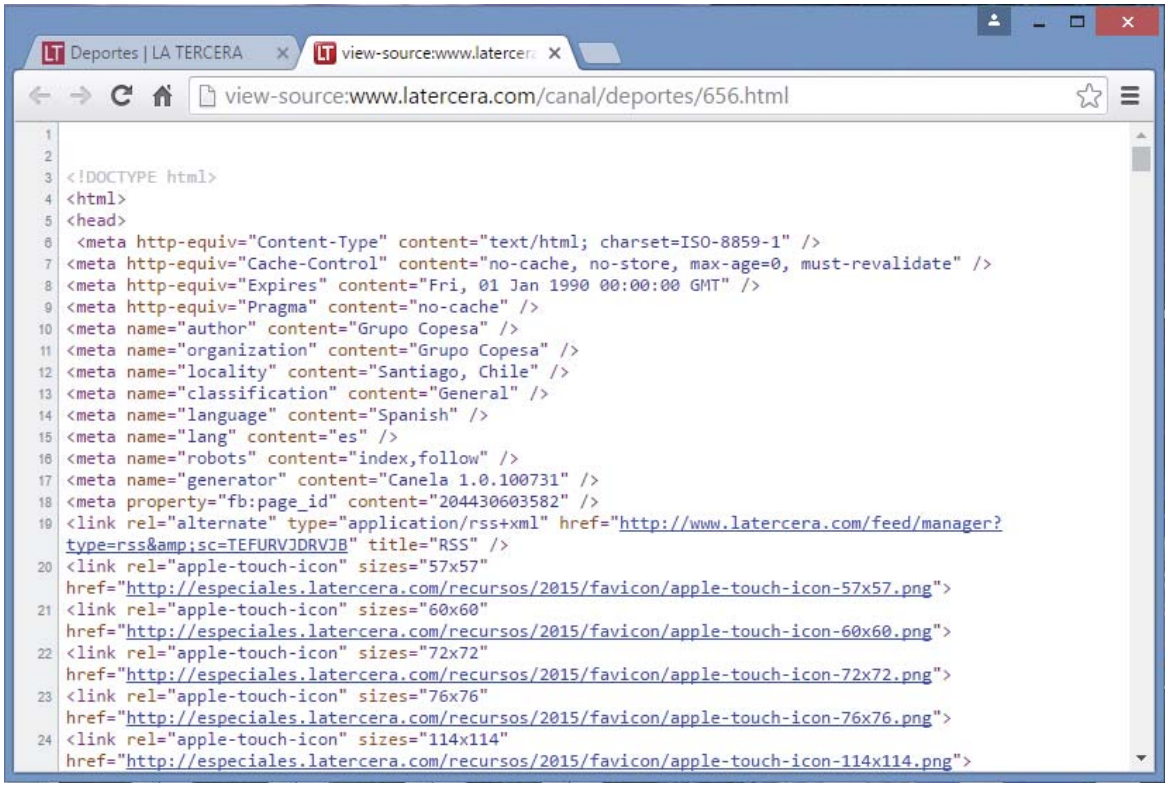


Fig. 2.2 Código HTML de la página web anterior. (Fuente: view-source:http://www.latercera.com/)

El lenguaje HTML basa su funcionamiento en la diferenciación. Para añadir un elemento externo a la página (imagen, vídeo, script, entre otros.), éste no se coloca directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De esta manera, la página web contiene solamente texto mientras que se centra en el navegador web, el que interpreta el código, la tarea de unir todos los elementos y visualizar la página final. Al ser un lenguaje estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma por cualquier navegador web actualizado.

Sin embargo, a lo largo de sus diferentes versiones, se han incorporado y eliminado diversas características, con el fin de hacerlo más eficiente y simplificar el desarrollo de páginas web que sean compatibles con distintos navegadores y plataformas (PC de escritorio, notebooks, teléfonos inteligentes, tablets).

2.2.1 Características principales de HTML

El lenguaje HTML puede ser creado y editado con cualquier editor de texto básico, como puede ser Gedit en GNU/Linux, el Bloc de notas de Windows, o cualquier otro editor que admita texto sin formato como GNU Emacs, Microsoft Wordpad, TextPad, Vim, Notepad++, entre otros.

HTML utiliza etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determina la forma en la que debe aparecer en su navegador el texto, así como también las imágenes y los demás elementos, en la pantalla del ordenador.

Toda etiqueta se identifica porque está encerrada entre los signos menor que y mayor que (<>), y algunas tienen atributos que pueden tomar algún valor. En general las etiquetas se aplicarán de dos formas especiales:

- Se abren y se cierran, negrita, que en su navegador web se vería como negrita.

- No pueden abrirse y cerrarse, <hr />, que en su navegador web se vería como una línea horizontal.

- Otras que pueden abrirse y cerrarse, como por ejemplo <p>.

- Los comandos básicos o elementales son:

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Ejemplo1</title>
    </head>
    <body>
        <p>Párrafo de ejemplo</p>
    </body>
</html>
```

2.2.2 Tipos de códigos HTML

En la figura 2.3 se observa la estructura de código HTML;

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Example<title>
5          <link rel="stylesheet" href="sty
6      </head>
7      <body>
8          <h1>
9              <a href="/">Header</a>
10         </h1>
11         <nav>
12             <a href="one/">One</a>
13             <a href="two/">Two</a>
14             <a href="three/">Three</a>
15         </nav>
```

Fig. 2.3 Estructura básica de código HTML. (Fuente: https://es.wikipedia.org/wiki/HTML)

DOCTYPE: Define el tipo de documento

<html>: Define el comienzo del documento HTML, le indica al navegador web que lo que viene a continuación debe ser interpretado como código HTML.

<script>: Incrusta un script en una web, o llama a uno mediante src="url del script".

<head>: Define la cabecera del documento HTML, esta cabecera suele contener información sobre el documento que no se muestra directamente al usuario como, por ejemplo, el título de la ventana del navegador. Dentro de la cabecera <head> es posible encontrar:

- <title>: Define el título de la página.
- <link>: Para vincular el sitio a hojas de estilo o iconos. Por ejemplo: <link rel="stylesheet" href="/style.css" type="text/css">.
- <style>: Para colocar el estilo interno de la página; ya sea usando CSS u otros lenguajes similares.
- <meta>: Para metadatos como la autoría o la licencia, incluso para indicar parámetros http (mediante http-equiv="") cuando no se pueden modificar por no estar disponible la configuración o por dificultades con server-side scripting.

<body>: Define el contenido principal o cuerpo del documento. Esta es la parte del documento html que se muestra en el navegador, dentro de esta etiqueta pueden definirse propiedades comunes a toda la página, como color de fondo y márgenes. Dentro del cuerpo <body> es posible encontrar numerosas etiquetas, como por ejemplo:

- <h1> a <h6>: Encabezados o títulos del documento con distinta relevancia.
- <table>: Define una tabla.
- <tr>: Fila de una tabla.
- <td>: Celda de una tabla (debe estar dentro de una fila).
- <a>: Hipervínculo o enlace, dentro o fuera del sitio web. Debe definirse el parámetro de pasada por medio del atributo href.
- : División de la página. Se recomienda, junto con css, en vez de <table> cuando se desea alinear contenido.
- <img>: Imagen. Requiere del atributo src, que indica la ruta en la que se encuentra la imagen. Por ejemplo: .
- : Etiquetas para listas.
- <b>: Texto en negrita (etiqueta no recomendada. Se recomienda usar la etiqueta).
- <i>: Texto en cursiva (etiqueta no recomendada. Se recomienda usar la etiqueta).
- <s>: Texto tachado (etiqueta no recomendada. Se recomienda usar la etiqueta <del>).
- <u>: Antes texto subrayado. A partir de la versión HTML 5 define porciones de texto diferenciadas o destacadas del resto, para indicar correcciones por ejemplo.

La mayoría de los comandos deben cerrarse como se abren, pero con una barra («/») tal como se muestra en los siguientes ejemplos:

<table><tr><td>Contenido de una celda</td></tr></table>.

<script>Código de un script integrado en la página</script>

2.3 Versiones de HTTP

El protocolo HTTP ha pasado por diversas versiones, muchas de las cuales son compatibles con las anteriores. La normativa RFC 2145 describe el uso de los números de versión del protocolo HTTP. El cliente le dice al servidor al principio de la petición la versión que usa, y el servidor usa la misma o una anterior en su respuesta. [1]

2.3.1 HTTP/0.9

Esta versión ya está obsoleta. Soporta sólo un método de petición, GET, además no especifica el número de versión del protocolo HTTP en las transferencias de datos y no soporta cabeceras. Como esta versión no soporta el método de petición POST, el cliente no puede enviarle mucha información al servidor.

2.3.2 HTTP/1.0

Esta versión especifica su versión del protocolo en las comunicaciones, y todavía se usa ampliamente, sobre todo en servidores proxy.

2.3.3 HTTP/1.1

Es la versión actual y la más utilizada, las conexiones persistentes están activadas por defecto y funcionan bien con los proxies. También permite al cliente enviar múltiples peticiones a la vez por la misma conexión.

2.3.4 HTTP/1.2

Con la nueva versión 1.2, HTTP consigue un apoyo mucho más fuerte para las jerarquías de recursos y obtiene un mejor soporte para interfaces de menú de texto, que están bien adaptados a entornos como los clientes móviles de computación. HTTP/1.2 incluye requisitos más estrictos que HTTP/1.1 con el fin de garantizar una aplicación fiable de sus características.

2.3.5 HTTP/2.0

Esta es la última versión que se ha creado, establecida en mayo de 2015, y no modifica la semántica de aplicación de HTTP, todas las características básicas continúan sin modificaciones. Sus mejoras se enfocan en cómo se reúnen y se agrupan los datos para el posterior transporte de éstos. Entre las nuevas funciones que tiene está que añade el uso de una única conexión, la compresión de cabeceras o el servicio 'server push'.

2.4 Características y Funcionamiento

Este es un protocolo que está orientado a transferencias (transacciones) de datos, y sigue el esquema de petición-respuesta entre un cliente y un servidor. El cliente realiza una petición enviando un mensaje, con determinado formato al servidor, y el servidor, al cual se le suele llamar un servidor web, le envía un mensaje de respuesta. Como ejemplos de cliente están los navegadores web y los spider. En la figura 2.4 se observa un esquema de funcionamiento mediante HTTP.

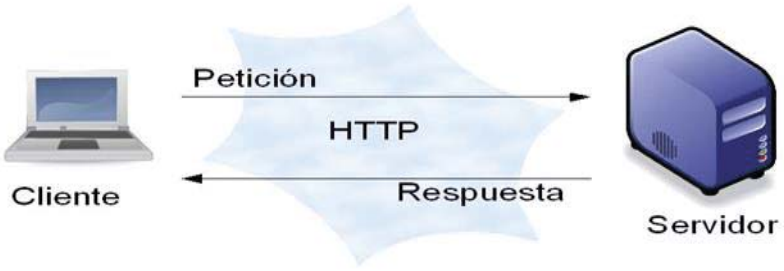


Fig. 2.4 Esquema de comunicación mediante el protocolo HTTP. (Fuente: roble.pntic.mec.es)

Desde el punto de vista de las comunicaciones, el protocolo está sustentado en los servicios de conexión TCP/IP. Un servidor escucha en un puerto de comunicaciones TCP, que por defecto es el puerto 80, y espera las solicitudes de conexión de los clientes web. Una vez que la conexión se ha establecido, el protocolo TCP se encarga de mantener la comunicación y garantizar que haya un intercambio de datos que esté libre de errores.

El protocolo HTTP se basa en etapas de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, el cual contiene el estado de la operación y su resultado. Todas las operaciones pueden adjuntar un objeto o recurso sobre el que actúan, cada objeto web es conocido por su URL.

Los recursos u objetos que actúan como entrada o salida de un comando HTTP están clasificados por su descripción MIME, que es una especificación relacionada al intercambio a través de Internet de todo tipo de archivos (texto, audio, video) de forma transparente para el usuario. De esta forma, el protocolo puede intercambiar cualquier tipo de dato, sin preocuparse de su contenido. La transferencia se realiza en modo binario, es decir, byte a byte, y la identificación MIME permitirá que el receptor reciba adecuadamente los datos.

Entre las principales características del protocolo HTTP se tiene que:

- Toda la comunicación entre los clientes y servidores se realiza a partir de caractéres de 8 bits. De esta forma, se puede transmitir cualquier tipo de documento, ya sea texto, binario, etc., sin modificar su formato original.

- Permite la transferencia de objetos multimedia. El contenido de cada objeto intercambiado está identificado por su descripción MIME.

- Existen tres métodos básicos de petición (hay más, pero por lo general no se utilizan) que un cliente puede utilizar para dialogar con el servidor:

  - GET, para recoger un objeto.

- POST, para enviar información al servidor.

- HEAD, para solicitar las características de un objeto (por ejemplo, la fecha de modificación de un documento HTML).

- Cada transacción HTTP implica una conexión con el servidor, que es liberada al término de la misma, es decir, en una operación se puede recoger un único objeto. En la actualidad se ha mejorado este procedimiento, permitiendo que una misma conexión se mantenga activa durante un determinado período de tiempo, de forma que sea utilizada en sucesivas transacciones. Este mecanismo, denominado HTTP Keep Alive, es empleado por la mayoría de los clientes y servidores modernos.

- No mantiene estado, es decir, cada petición de un cliente a un servidor no es influida por las transacciones anteriores. Para el servidor cada petición es una operación totalmente independiente del resto.

- Cada objeto al que se le aplican los métodos de petición del protocolo está identificado mediante la información del final de la URL.

Como este protocolo funciona en base a petición y respuesta entre cliente y servidor, cada vez que un cliente realiza una petición a un servidor, se ejecutan los siguientes pasos:

1.- Un usuario accede a una URL, seleccionando un enlace de un documento HTML o introduciéndola directamente en el campo Dirección del cliente Web.

2.- El cliente web decodifica la URL, separando sus diferentes partes. Así identifica el protocolo de acceso, la dirección DNS o IP del servidor, el posible puerto opcional, que es por defecto el 80) y el objeto requerido del servidor.

3.- Se abre una conexión TCP/IP con el servidor, llamando al puerto TCP correspondiente.

4.- Se realiza la petición. Para ello, se envía el método de petición necesario, la dirección del objeto requerido (el contenido de la URL que sigue a la dirección del servidor), la versión del protocolo HTTP empleada y un conjunto variable de información, que incluye datos como las capacidades del navegador o datos que son opcionales para el servidor,

5.- El servidor devuelve la respuesta al cliente, la cual consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.

6.- Se cierra la conexión TCP. Si esto no ocurre, se utiliza el modo HTTP Keep Alive, y este proceso se repetirá en cada acceso al servidor HTTP.

La comunicación con el servidor HTTP se establece a través de mensajes formados por líneas de texto, cada una de las cuales contiene los diferentes comandos y opciones del protocolo. Sólo existen dos tipos de mensajes, uno para realizar peticiones y otro para devolver la respuesta correspondiente.

2.5 Comunicación entre el cliente (usuario) y el servidor

La comunicación entre el cliente y el servidor se lleva a cabo en dos etapas, lo cual se observa en la figura 2.5: [2]

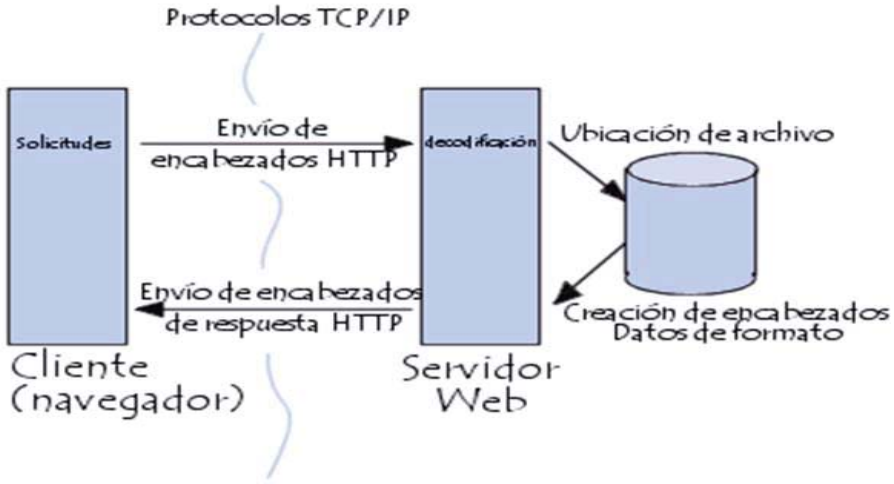


Fig. 2.5 Esquema de funcionamiento de comunicación cliente-servidor.
(Fuente: http://es.ccm.net/contents/264-el-protocolo-http)

- El cliente (navegador) realiza una solicitud HTTP.
- El servidor procesa la solicitud y después envía una respuesta HTTP.

2.5.1 Solicitud HTTP

Es un conjunto de líneas que el navegador envía al servidor, las cuales incluyen:

- Una línea de solicitud; la cual es una línea que especifica el tipo de documento pedido, el método que se usará y la versión del protocolo utilizada. Esta línea está formada por tres elementos que deben estar separados por un espacio:

  - El método de petición.

  - La dirección URL.

  - La versión del protocolo utilizada por el cliente.

- Los campos del encabezado de solicitud; que son un conjunto de líneas opcionales que aportan información adicional sobre la solicitud y/o el cliente (navegador, sistema operativo), y cada una de estas líneas está formada por un nombre que describe el tipo de encabezado, seguido de dos puntos (:) y el valor del encabezado.

- El cuerpo de la solicitud; que es un conjunto de líneas opcionales que deben estar separadas de las líneas anteriores por una línea en blanco y, por ejemplo, permiten que se envíen datos por un método de petición POST durante la transmisión de datos al servidor utilizando un formulario. También muestra el código HTML utilizado en la transacción HTTP.

2.5.2 Respuesta HTTP

Una respuesta HTTP es un conjunto de líneas que el servidor envía al cliente, las cuales incluyen:

- Una línea de estado; la cual es una línea que especifica la versión del protocolo utilizada y el estado de la solicitud que está en proceso a través de un texto explicativo y un código. Esta línea está compuesta por tres elementos que deben estar separados por un espacio:

  - La versión del protocolo utilizada.

  - El código de estado.

  - El significado del código.

- Los campos del encabezado de respuesta; que son un conjunto de líneas opcionales que permiten aportar información adicional sobre la respuesta y/o el servidor, y cada una de estas líneas está compuesta por un nombre que caracteriza el tipo de encabezado, seguido por dos puntos (:) y por el valor del encabezado.

- El cuerpo de la respuesta, el cual contiene el documento solicitado.

2.5.3 Estructura de comunicación cliente-servidor

Para obtener un recurso con el URL http://www.example.com/index.html, se realizan las siguientes etapas:

1.- Se abre una conexión al host www.example.com, puerto 80, que es el puerto por defecto para HTTP.

2.- Se envía un mensaje en el estilo siguiente:

GET /index.html HTTP/1.1

Host: www.example.com

User-Agent: nombre-cliente

[Línea en blanco]

La respuesta del servidor está formada por encabezados, seguidos del recurso solicitado, en el caso de una página web:

HTTP/1.1 200 OK

Date: Fri, 31 Dec 2003 23:59:59 GMT

Content-Type: text/html

Content-Length: 1221

<html>

<body>

<h1>Página principal de tuHost</h1>

(Contenido)

.

</body>

</html>

La primera parte corresponde a la petición del usuario al servidor, y la segunda a la respuesta del servidor al usuario, al final se observa una parte entre paréntesis angulares la cual corresponde al cuerpo del mensaje. Para tener un ejemplo concreto de comunicación cliente-servidor, se tiene la siguiente figura 2.6:

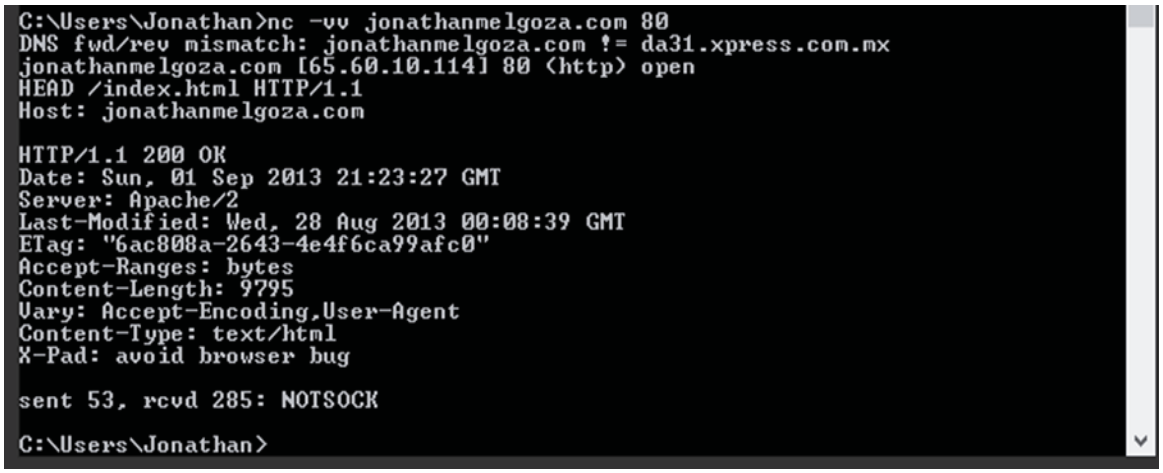


Fig. 2.6 Ejemplo de diálogo HTTP entre cliente y servidor. (Fuente: https://jonathanmelgoza.com/blog/como-funciona-el-protocolo-http-manual/)

2.6 Métodos de Petición

El protocolo HTTP define una serie de métodos de petición que pueden utilizarse y tiene flexibilidad para ir añadiendo nuevos métodos y nuevas funciones. El número de métodos de petición ha ido creciendo según avanzan las versiones de HTTP, y cada método indica la acción que desea que se efectúe sobre el recurso identificado. Lo que este recurso identificado representa depende de la aplicación del servidor. Por ejemplo el recurso puede corresponder a un archivo que se encuentre en el servidor, y cuando el servidor recibe una petición de URL, la cabecera de la misma contiene un código que solicita al servidor que realice una de las tareas determinadas. Estas tareas son los denominados métodos de petición.

El servidor soporta los siguientes métodos: [11]

GET

El método más común en la navegación web. Si se habilita el método GET, el servidor devolverá los datos identificados por el URL, devuelve un código de respuesta y las cabeceras asociadas. Si los datos identificados por el URL se refieren a un programa ejecutable, el servidor devolverá la salida del programa. Incluye el documento solicitado (habitualmente una página) en el cuerpo del mensaje.

HEAD

Idéntico al anterior, pero con la excepción de que no devuelve el documento en el cuerpo de la respuesta. Se utiliza para extraer información sobre el documento solicitado o comprobar si existe sin necesidad de enviar y recibir el documento como tal.

POST

Este método está pensado para publicar la información contenida en el cuerpo de la petición, en el recurso donde se envía esa petición. La información que se publica y la forma de hacerlo depende completamente del servidor y del recurso. Hoy, el uso que se le da a este método es el de traspaso de parámetros de cliente a servidor (en muchas ocasiones para ficheros). La respuesta por parte del servidor es la misma que para una petición GET.

PUT

Es el método inverso a GET, permite escribir datos en el servidor, la diferencia con el método POST, es que POST se trabaja sobre un recurso que manejará la información a su manera y el método PUT trabajará sobre un recurso final, es decir, se modificará el recurso en el que se trabaja. La petición contiene datos y un URL. El servidor almacena el recurso identificado en el URL, si el recurso ya existe, PUT lo sustituye, mientras que si el recurso no existe, PUT lo crea. Como PUT normalmente permite que los clientes añadan o sustituyan información en el servidor, debe utilizar configuraciones de protección para definir quién puede utilizar este método y para qué archivos.

DELETE

Sirve para eliminar un recurso especificado en el URL, aunque pocas veces será permitido por un servidor web. Si se habilita éste método, el servidor suprimirá el objeto identificado por el URL. Una vez que se haya suprimido el objeto, el URL no será válido. Como DELETE normalmente permite que los clientes supriman información del servidor, debe utilizar configuraciones de protección para definir quién puede utilizar este método y qué archivos pueden suprimirse.

TRACE

Este método solicita al servidor que en la respuesta coloque todos los datos que reciba en el mensaje de petición, permite al cliente ver qué es lo que se recibe al otro extremo de la cadena de petición. Entonces, el cliente puede utilizar los datos para comprobación o para información de diagnóstico ya que el cliente puede ver lo que llega al servidor y de esta forma ver lo que añaden al mensaje los servidores intermedios. El tipo de contenido de la respuesta es: Message/http

OPTIONS

Devuelve los métodos HTTP que el servidor soporta para un URL específico. Esto puede ser utilizado para comprobar la funcionalidad de un servidor web mediante petición en lugar de un recurso específico. Si se habilita éste método, la petición devolverá información referente a las opciones de comunicaciones en la cadena de respuesta identificada por el URL. Este método permite al cliente determinar cuáles son las opciones y los requisitos asociados con un objeto, o bien cuáles son las capacidades de un servidor. No es necesaria ninguna acción sobre el objeto ni su recuperación. La información de los métodos permitidos por el servidor se observa en la cabecera ALLOW.

CONNECT

Se utiliza para saber si se tiene acceso a un host, no necesariamente la petición llega al servidor, este método se utiliza principalmente para saber si un proxy nos da acceso a un host bajo condiciones especiales, como el caso por ejemplo de "corrientes" de datos bidireccionales encriptados (como lo requiere SSL). Si se habilita el método CONNECT, el servidor podrá establecer una sesión de túnel SSL entre un cliente (por ejemplo, Netscape Navigator) y un servidor remoto a través de un servidor proxy. Las sesiones entre el cliente y el proxy y entre el proxy y el servidor remoto serán seguras. El proxy no puede acceder a los datos enviados al cliente. El servidor proxy puede ser un servidor base o un servidor seguro.

2.7 La seguridad de los métodos HTTP

Hay ciertos métodos de petición HTTP que se utilizan muy poco ya que involucran ciertos riesgos en la comunicación entre el cliente y el servidor, como lo son PUT y DELETE, pero también hay otros métodos, como lo son: [4]

2.7.1 OPTIONS

A primera vista, dos de estos métodos pueden resultar críticos en el entorno web donde se encuentren habilitados. Tanto PUT como DELETE permiten interactuar directamente con recursos legítimos del sistema. Estos dos métodos pueden servir por ejemplo para eliminar archivos o activos de una compañía. Encontrarse este tipo de configuración en servidores en producción no es demasiado común, pero aún hoy no es extraño localizarlos. Para demostrarlo, basta con hacer consultas avanzadas sobre los motores de búsqueda, o basarse en las respuestas a las peticiones OPTIONS. Para ilustrar esto con ejemplos se tienen las figuras 2.7 y 2.8:

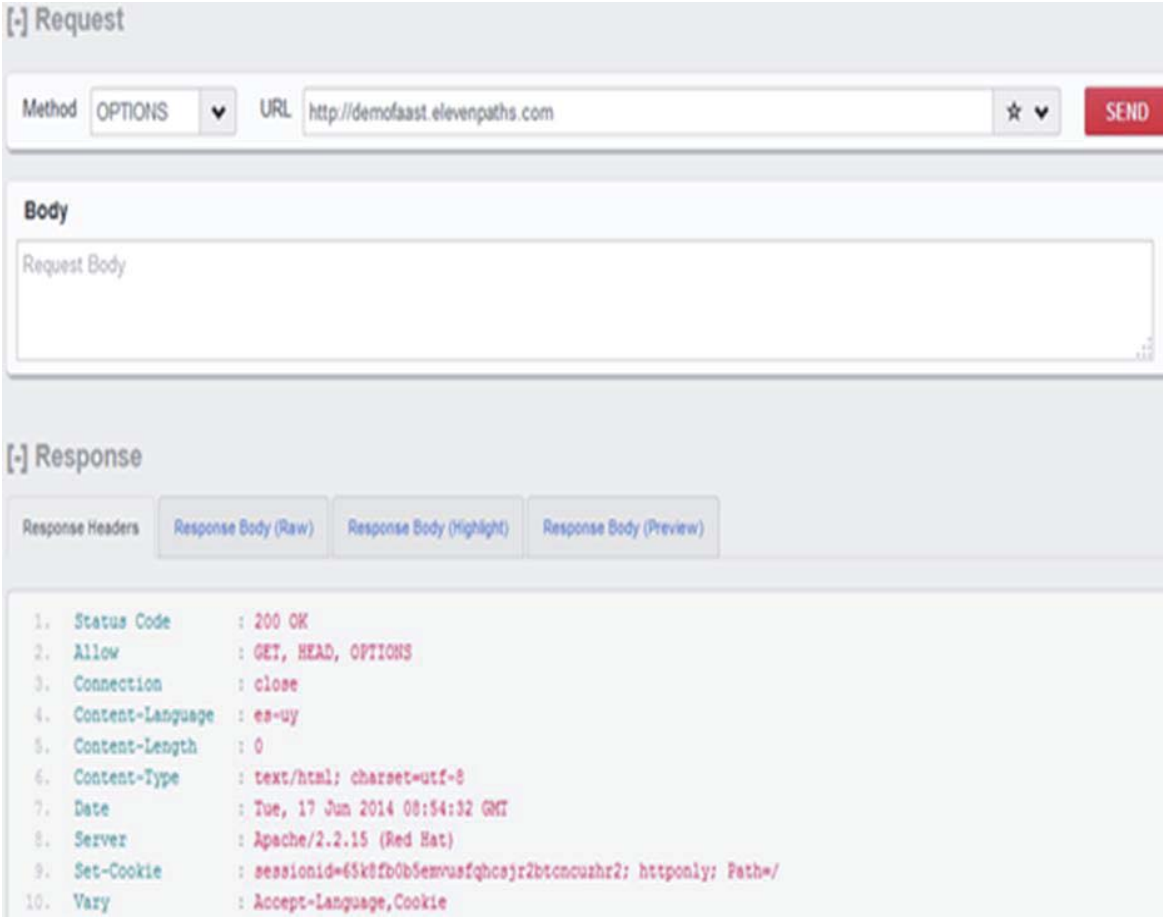


Fig. 2.7 Ejemplo de petición y respuesta OPTIONS usando el complemento RestClient de Firefox. (Fuente: http://blog.elevenpaths.com/2014/06/la-seguridad-en-los-metodos-http.html)

Este método devuelve información del servidor, pero esta información no tiene que ser completa, ni veraz. En ocasiones se mostrará una cabecera ALLOW con métodos de petición HTTP permitidos, lo que no quiere decir que estos métodos estén implementados. Es decir, a pesar de que el método de petición OPTIONS indique que se admite un método concreto, al utilizar este método es muy posible que retorne un código de error, como por ejemplo no implementado (501), o no encontrado (404).

```
OPTIONS / HTTP/1.1
Host: [redacted]
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 16 Jun 2014 12:18:24 GMT
Server: Apache/2.2.15 (Red Hat)
Vary: Accept-Language,Cookie
Content-Language: es-uy
Allow: GET, HEAD, OPTIONS
Set-Cookie: sessionid=k77x17w1lis4bsr46xsvem9ku8c0yna4; httponly; Path=/
Content-Length: 0
Connection: close
Content-Type: text/html; charset=utf-8
```

Fig. 2.8 Ejemplo de volcado del paquete de petición y respuesta con método OPTIONS. (Fuente: http://blog.elevenpaths.com/2014/06/la-seguridad-en-los-metodos-http.html)

En otras ocasiones también ocurrirá lo contrario, es decir, la cabecera ALLOW del método de petición OPTIONS no mostrará los métodos que realmente están permitidos e implementados, aunque si es posible que después se lleve a cabo. En resumen, no se deben descartar los métodos de petición que no estén reflejados en esa cabecera.

2.7.2 TRACE y ataques XST

Además de los métodos PUT y DELETE, los cuales resultan muy peligrosos si no se tienen completamente controlados, también se encuentra el método de petición TRACE, que en principio se considera inofensivo. Un ejemplo se tiene en la figura 2.9:

```
TRACE / HTTP/1.1
Host: [redacted]
Cookie: name=value
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 16 Jun 2014 12:22:28 GMT
Server: Apache/2.2.15 (Red Hat)
Connection: close
Transfer-Encoding: chunked
Content-Type: message/http

60
TRACE / HTTP/1.1
Host: [redacted]
Cookie: name=value
Connection: Keep-Alive
```

Fig. 2.9 Ejemplo de volcado del paquete de envío y respuesta con método TRACE. (Fuente: http://blog.elevenpaths.com/2014/06/la-seguridad-en-los-metodos-http.html)

Como se observa en la figura, este método devuelve como cuerpo las cabeceras de la petición del cliente, incluyendo la cabecera Cookie que puede resultar crítica si existe una sesión establecida con el servidor. La combinación de este método HTTP con un fallo "Cross Site Scripting" en la aplicación web puede terminar en un robo de sesión de usuario, incluso si las Cookies han sido establecidas como HttpOnly. Este ataque es conocido como "Cross Site Tracing" o XST.

El ataque consiste en que cuando se inyecta código JavaScript y se daña el sistema, se realiza una petición con el método de petición TRACE al propio servidor y se envía el cuerpo de esta respuesta al propio atacante por la vía que sea más adecuada. El contenido devuelto incluirá la cabecera Cookie con sus respectivos valores, tenga el flag que tenga. Un ejemplo de código JavaScript que es posible inyectar se muestra en la figura 2.10, aunque en vez de enviar la respuesta a un servidor que sea propiedad del atacante, se muestra en un mensaje "alert" como prueba.

```
function sendTrace()
    {
        var xmlhttp = new XMLHttpRequest();
        var url = '/';

        xmlhttp.withCredentials = true;
        xmlhttp.open('TRACE', url, true);
        xmlhttp.onload = function (e)
        {
          if (xmlhttp.readyState === 4)
          {
            if (xmlhttp.status === 200)
                alert(xmlhttp.responseText);
          }
        }
        xmlhttp.send("");
    }
```

Fig. 2.10 Ejemplo de código JavaScript para inyectar y explotar un fallo tipo XST. (Fuente: http://blog.elevenpaths.com/2014/06/la-seguridad-en-los-metodos-http.html)

Aunque lo cierto es que hoy los navegadores más modernos ya bloquean este tipo de peticiones para evitar específicamente este ataque. Pero tampoco se debe olvidar que existen otras alternativas a JavaScript para realizar peticiones de tipo TRACE como Flash, Applets Java. Con ellos se puede obtener el mismo resultado evitando los bloqueos del navegador.

Desde el punto de vista del servidor, la manera más clara de evitar este tipo de problemas es reducir la superficie de ataque inhabilitando aquellos métodos que no sean útiles para el entorno, y controlar fuertemente los que sí se están utilizando. Faast, el servicio de pentesting persistente, monitoriza este tipo de configuraciones en los servidores web y analiza cualquier método de petición HTTP que no sea seguro y que pueda estar habilitado o implementado.

2.8 Cabeceras HTTP

Son datos que se envían en los métodos de petición y en los códigos de respuesta HTTP para entregar información esencial sobre la transacción en curso. Cada cabecera es especificada por un nombre de cabecera, seguido por un punto y coma, un espacio en blanco y el valor de dicha cabecera, seguido por un retorno de carro, seguido por un salto de línea. Se usa una línea en blanco para indicar el final de las cabeceras, y si no hay cabeceras la línea en blanco igual se debe colocar. [1]

Las cabeceras le dan una gran flexibilidad al protocolo HTTP permitiendo añadir nuevas funciones sin tener que cambiar la base. Por eso según han ido avanzando las versiones de HTTP, se han ido añadiendo más y más cabeceras que están permitidas.

Las cabeceras pueden tener datos que deben ser procesados por el cliente, por el servidor, o por los intermediarios. En el caso del cliente, en respuesta a una petición se puede indicar el tipo del contenido que tiene, mientras que para el servidor se encuentran los tipos de representaciones aceptables por el cliente del contenido que pide. Por último para el caso de intermediarios, está el caso de cómo gestionar el cacheo por parte de los proxys.

Dependiendo del tipo de mensaje en el que puede ir una cabecera es posible clasificarlas en cabeceras de petición, de respuesta y cabeceras que pueden ir en ambos casos.

Es posible clasificar las cabeceras según su función. Por ejemplo:

* Cabeceras que indican las capacidades aceptadas por el que envía el mensaje:

- Accept: Indica el contenido MIME aceptado.
- Accept-Charset: Indica el código de caracteres aceptado.
- Accept-Encoding: Indica el método de compresión aceptado.
- Accept-Language: Indica el idioma aceptado.
- User-Agent: Para describir al cliente.
- Server: Indica el tipo de servidor.
- Allow: Indica los métodos permitidos para el recurso.

* Cabeceras que describen el contenido:

- Content-Type: Indica el MIME del contenido.
- Content-Length: Longitud del mensaje.
- Content-Range: Cuando un mensaje parcial pertenece en un mensaje de cuerpo completo.

- Content-Encoding: El tipo de codificación utilizada en los datos.

- Content-Language: El lenguaje previsto para el contenido encerrado.

- Content-Location: Una ubicación alternativa para los datos devueltos.

* Cabeceras que hacen referencias a URIs:

- Location: Indica donde está el contenido.

- Referer: Indica el origen de la petición.

* Cabeceras que permiten ahorrar transmisiones:

- Date: Fecha de creación.

- If-Modified-Since: Permite a un 304 Not Modified ser devuelto si el contenido no se ha modificado.

- If-Unmodified-Since: Sólo envía la respuesta si la entidad no se ha modificado desde un tiempo específico.

- If-Match: Sólo lleva a cabo la acción si la entidad del cliente que suministra la información coincide con la misma entidad en el servidor.

- If-Range: Si la entidad no ha cambiado, se envía la parte que falta, de lo contrario, se envía toda la nueva entidad.

- Expires: Da la fecha/hora después de lo cual la respuesta se considerará caducada.

- Last-Modified: La última fecha de modificación para el objeto solicitado.

- Cache-Control: Le dice a todos los mecanismos de almacenamiento en caché del servidor al cliente si pueden almacenar en caché este objeto.

- Via: Informa al servidor de proxies a través de los cuales se envió la solicitud.

- Pragma: Aplicación de campos específicos que pueden tener diversos efectos en cualquier lugar a lo largo de la cadena de petición-respuesta.

- Etag: Identificador de una versión específica de un recurso.

- Age: La edad del objeto ha estado en un proxy caché en segundos.

- Retry-After: Si una entidad no está disponible temporalmente, esto indica al cliente que vuelva a intentarlo más tarde.

* Cabeceras para autentificación:

- Authorization: Determina la autenticación HTTP para la petición en curso.

- WW-Authenticate: Indica el esquema de autenticación que se debe utilizar para acceder a la entidad solicitada.

* Cabeceras para describir la comunicación:

- Host: Indica máquina destino del mensaje.

- Connection: Indica como establecer la conexión.

* Otras:

- Range: Para descargar sólo partes del recurso.

- Max-Forward: Límite de cabeceras añadidas en TRACE.

2.9 Códigos de respuesta HTTP

El código de respuesta es un número que indica que es lo que ha pasado con la petición, mientras que el resto del contenido de la respuesta dependerá del valor de este código. El sistema es flexible y la lista de códigos ha ido aumentando para así adaptarse a los cambios e identificar nuevas situaciones, por tanto cada código tiene un significado concreto, y el número de los códigos están ordenados de manera que se pueda identificar el tipo de respuesta que ha dado el servidor. La lista comenzará con la tabla 2-1, que indica códigos que empiezan con 1: [3]

Código	Mensaje	Descripción
1xx	Respuestas Informativas	Petición recibida, continuando proceso. Esta respuesta significa que el servidor ha recibido los encabezados de la petición, y que el cliente debería proceder a enviar el cuerpo de la misma.
100	Continue	El navegador puede continuar realizando su petición, se utiliza para indicar que la primera parte de la petición del navegador se ha recibido correctamente.
101	Switching Protocols	El servidor acepta el cambio de protocolo propuesto por el navegador (puede ser por ejemplo un cambio de HTTP/1.0 a HTTP/1.1).

102	Processing (WebDAV)	El servidor está procesando la petición del navegador pero todavía no ha terminado, esto evita que el navegador piense que la petición se ha perdido cuando no recibe ninguna respuesta.
103	Checkpoint	Se va a reanudar una petición POST o PUT que fue abortada previamente.

Tabla 2-1: Códigos Informativos

Para códigos que empiezan con 2, se tendrá la tabla 2-2:

Código	**Mensaje**	**Descripción**
2xx	Peticiones correctas	Esta clase de código de estado indica que la petición fue recibida correctamente, entendida y aceptada.
200	Ok	La petición del navegador se ha completado con éxito.
201	Created	La petición del navegador se ha completado con éxito y como resultado, se ha creado un nuevo recurso (la respuesta incluye la URI de ese recurso).
202	Accepted	La petición del navegador se ha aceptado y se está procesando, por lo que todavía no hay una respuesta, se utiliza por ejemplo cuando un proceso realiza una petición muy compleja a un servidor y no quiere mucho tiempo esperando la respuesta.
203	Non-Authoritative Information	La petición se ha completado con éxito, pero su contenido no se ha obtenido de la fuente originalmente solicitada sino de otro servidor.
204	No Content	La petición se ha completado con éxito pero su respuesta no tiene ningún contenido (la respuesta sí que puede incluir información en sus cabeceras HTTP).
205	Reset Content	La petición se ha completado con éxito, pero su respuesta no tiene contenidos, el navegador tiene que inicializar la página desde la que se realizó la petición. Este código es útil por ejemplo para páginas con formularios cuyo contenido debe borrarse después de que el usuario lo envíe.

206	Partial Content	La respuesta de esta petición sólo tiene parte de los contenidos, tal y como lo solicitó el propio navegador (se utiliza por ejemplo cuando se descarga un archivo muy grande en varias partes para acelerar la descarga).
207	Multi-Status (WebDAV)	La respuesta consiste en un archivo XML que contiene en su interior varias respuestas diferentes, el número depende de las peticiones realizadas previamente por el navegador.
208	Already Reported (WebDAV)	El listado de elementos DAV ya se notificó previamente, por lo que no se van a volver a listar.

Tabla 2-2: Códigos de aceptación

Para los códigos que empiezan con 3, se tendrá la tabla 2-3:

Código	**Mensaje**	**Descripción**
3xx	Redirecciones	El cliente tiene que tomar una acción adicional para completar la petición. Esta clase de código de estado indica que una acción consecutiva necesita efectuarse por el agente de usuario para completar la petición.
300	Multiple Choices	Existe más de una variante para el recurso solicitado por el navegador, por ejemplo si la petición se corresponde con más de un archivo.
301	Moved Permanently	El recurso solicitado por el navegador se encuentra en otro lugar y este cambio es permanente, el navegador es redirigido automáticamente a la nueva localización de ese recurso.
302	Moved Temporarily	El recurso solicitado por el navegador se encuentra en otro lugar, aunque sólo por tiempo limitado, mientras que el navegador es redirigido automáticamente a la nueva localización de ese recurso.
303	See Other	El recurso solicitado por el navegador se encuentra en otro lugar. El servidor no redirige automáticamente al navegador, pero le indica el nuevo URI en el que se puede obtener el recurso.
304	Not Modified	Cuando el navegador pregunta si un recurso ha cambiado desde la última vez que se solicitó, el servidor responde con este código cuando el recurso no ha cambiado.

305	Use Proxy	El recurso solicitado por el navegador debe obtenerse a través del proxy, cuya dirección se indica en la cabecera Location de esta misma respuesta.
306	Switch Proxy	Este código se utilizaba en las versiones antiguas de HTTP, pero ya no se usa (aunque está reservado para usos futuros).
307	Temporary Redirect	El recurso solicitado por el navegador se puede obtener en otro lugar, pero sólo para esta petición. Las próximas peticiones pueden seguir utilizando la localización original del recurso.
308	Permanent Redirect	El recurso solicitado por el navegador se encuentra en otro lugar y este cambio es permanente. A diferencia del código 301, no se permite cambiar el método HTTP para la nueva petición.

Tabla 2-3: Códigos de redirección

Para los códigos que empiezan con 4, se tendrá la tabla 2-4:

Código	Mensaje	Descripción
4xx	Errores del cliente	La solicitud contiene una sintaxis incorrecta o que no puede procesarse.
400	Bad Request	El servidor no es capaz de entender la petición del navegador porque su sintaxis no es correcta.
401	Unauthorized	El recurso solicitado por el navegador requiere de autenticación. La respuesta incluye una cabecera de tipo WWW-Authenticate para que el navegador pueda iniciar el proceso de autenticación.
402	Payment Required	Este código está reservado para usos futuros.
403	Forbidden	La petición del navegador es correcta, pero el servidor no puede responder con el recurso solicitado porque se ha denegado el acceso.
404	Not Found	El servidor no puede encontrar el recurso solicitado por el navegador y no es posible determinar si esta ausencia es temporal o permanente.
405	Method Not Allowed	El navegador ha utilizado un método (GET, POST,…)

		no permitido por el servidor para obtener ese recurso.
406	Not Acceptable	El recurso solicitado tiene un formato que en teoría no es aceptable por el navegador, según los valores que ha indicado en la cabecera Accept de la petición.
407	Proxy Authentication Required	Es muy similar al código 401, pero en este caso, el navegador debe autenticarse primero con un proxy.
408	Request Timeout	El navegador ha tardado demasiado tiempo en realizar su petición, y el servidor ya no espera esa petición. No obstante, el navegador puede realizar nuevas peticiones cuando quiera.
409	Conflict	La petición del navegador no se ha podido completar porque se ha producido un conflicto con el recurso solicitado.
410	Gone	No es posible encontrar el recurso solicitado por el navegador y esta ausencia se considera permanente. Si existe alguna posibilidad de que el recurso vuelva a estar disponible, se debe utilizar el código 404.
411	Length Required	El servidor rechaza la petición del navegador porque no incluye la cabecera Content-Length adecuada.
412	Precondition Failed	El servidor no es capaz de cumplir con algunas de las condiciones impuestas por el navegador en su petición.
413	Request Entity Too Large	La petición del navegador es demasiado grande, y por ese motivo el servidor no la procesa.
414	Request-URI Too Long	El URI de la petición del navegador es demasiado grande, y por ese motivo el servidor no la procesa. Esta condición se produce en muy raras ocasiones y casi siempre porque el navegador envía como GET una petición que debería ser POST).
415	Unsupported Media Type	La petición del navegador tiene un formato que no entiende el servidor y por eso no se procesa.
416	Requested Range Not Satisfiable	El navegador ha solicitado una parte inexistente de un recurso. Este error se produce cuando el navegador descarga por partes un archivo muy grande, y calcula mal el tamaño de algún trozo.
417	Expectation Failed	La petición del navegador no se procesa porque el servidor no es capaz de cumplir con los requerimientos de la cabecera Expect de la petición
422	Unprocessable Entity	La petición del navegador tiene el formato correcto, pero sus contenidos tienen algún error semántico que

	(WebDAV)	impide al servidor responder.
423	Locked (WebDAV)	El recurso solicitado por el navegador no se puede entregar porque está bloqueado.
424	Failed Dependency (WebDAV)	La petición del navegador ha fallado debido al error de alguna petición anterior.
426	Upgrade Required	El navegador debe cambiar a un protocolo diferente para realizar las peticiones, como por ejemplo TLS/1.0.
428	Precondition Required	El servidor requiere que la petición del navegador sea condicional, este tipo de peticiones evitan los problemas producidos al modificar con PUT un recurso que ha sido modificado por otra parte.
429	Too Many Requests	El navegador ha realizado demasiadas peticiones en un determinado período de tiempo
431	Request Header Fileds Too Large	El servidor no puede procesar la petición porque una de las cabeceras de la petición es demasiado grande, este error también se produce cuando la suma del tamaño de todas las peticiones es demasiado grande.

Tabla 2-4: Códigos de error del cliente

Por último para los códigos que empiezan con 5, se tendrá la tabla 2-5:

Código	Mensaje	Descripción
5xx	Errores de servidor	El servidor falló al completar una solicitud aparentemente válida. Estos códigos indican casos en los cuales el servidor tiene registrado aún antes de servir la solicitud, que está errado o es incapaz de ejecutar la petición.
500	Internal Server Error	La solicitud del navegador no se ha podido completar porque se ha producido un error inesperado en el servidor.
501	Not Implemented	El servidor no soporta alguna función necesaria para responder a la solicitud del navegador, como por ejemplo el método utilizado para la petición.
502	Bad Gateway	El servidor está actuando de proxy o Gateway, y ha recibido una respuesta inválida del otro servidor, por lo que no puede responder adecuadamente a la

		petición del navegador.
503	Service Unavailable	El servidor no puede responder a la petición del navegador porque está congestionado o está realizando tareas de mantenimiento.
504	Gateway Timeout	El servidor está actuando de proxy o gateway y no ha recibido a tiempo una respuesta del otro servidor, por lo que no puede responder adecuadamente a la petición del navegador.
505	HTTP Version Not Supported	El servidor no soporta o no quiere soportar la versión del protocolo HTTP utilizada en la petición del navegador.
506	Variant Also Negotiates	El servidor ha detectado una referencia circular al procesar la parte de la negociación del contenido de la petición.
507	Insufficient Storage (WebDAV)	El servidor no puede crear o modificar el recurso solicitado porque no hay suficiente espacio de almacenamiento libre.
508	Loop Detected (WebDAV)	La petición no se puede procesar porque el servidor ha encontrado un bucle infinito al intentar procesarla.
510	Not Extended	La petición del navegador debe añadir más extensiones para que el servidor pueda procesarla.
511	Network Authentication Required	El navegador debe autenticarse para poder realizar peticiones.

Tabla 2-5: Códigos de error del servidor

3 Metalenguajes

En términos generales cualquier metalenguaje es el lenguaje o los símbolos utilizados en el lenguaje mismo que se está discutiendo o examinando. En la lógica y la lingüística un metalenguaje es un lenguaje utilizado para hacer declaraciones acerca de las declaraciones en otra lengua, es un lenguaje que se usa para hablar acerca de otro lenguaje. Las expresiones en un metalenguaje a menudo se distinguen de las de un lenguaje de objetos mediante el uso de cursivas, comillas, o la escritura en una línea separada.

3.1 El uso de metalenguajes

En diversas ocasiones se utiliza este recurso con el que, si no se es consciente, se pueden cometer errores de interpretación. En la gramática se distingue entre uso y mención.

Todo lenguaje tiene un objeto al que se dirige o refiere, es el "lenguaje-objeto". A su vez todo lenguaje que tenga por objeto un lenguaje es un "metalenguaje", que a su vez puede ser lenguaje objeto de otro metalenguaje de orden superior, y así sucesivamente.

Como ejemplo, se consideran las diferentes referencias de la frase siguiente: "Antonio dice que Luis dijo que María Luisa dijo que...". "Antonio dijo que ayer fue al cine". Tal afirmación no nos da información acerca de si Antonio fue o no fue ayer al cine. No tener en cuenta esa distinción que habla de la realidad del hecho: "Antonio dijo", y el lenguaje (metalenguaje) acerca de lo que dijo Antonio: "que ayer fue al cine", se presta a confusiones interpretativas. [5]

3.2 Papel en la informática

Las Computadoras actúan en base a programas, es decir, conjuntos de instrucciones en un lenguaje claro y sencillo. El desarrollo de un lenguaje de programación implica el uso de un metalenguaje. Backus-Naur Form, fue desarrollado en la década del 60´ por John Backus y Peter Naur, y es uno de los primeros metalenguajes utilizados en la informática. [5]

3.2.1 Notación de Backus-Naur

La notación de Backus-Naur, también conocida como Backus-Naur Form (BNF), Backus-Naur Formalism o Backus Normal Form, es un metalenguaje usado para expresar gramáticas libres de contexto, es decir, una manera formal de describir lenguajes formales.

El BNF se utiliza ampliamente como notación para las gramáticas de los lenguajes de programación de la computadora, de los sistemas de comando y de los protocolos de comunicación, así como una notación para representar partes de las gramáticas de la lengua natural. La mayoría de los libros de textos para la teoría o la semántica del lenguaje de programación documentan el lenguaje de programación en BNF.

Algunas variantes a BNF, tales como la Augmented Backus-Naur Form (ABNF) y la Extended Backus‑Naur Form (EBNF), tienen su propia documentación. [6]

3.2.1.1 Introducción

Una especificación de BNF es un sistema de reglas de derivación, escrito como:

<símbolo> ::= <expresión con símbolos>

Donde <símbolo> es un símbolo no terminal, y la expresión consiste en secuencias de símbolos o secuencias separadas por la barra vertical, '|', indicando una opción, el conjunto es una posible sustitución para el símbolo a la izquierda. Los símbolos que nunca aparecen en un lado izquierdo son terminales.

Como ejemplo de BNF, se tiene el caso para una dirección postal de los EE.UU:

<dirección postal> ::= <nombre> <dirección> <apartado postal>

<nombre> ::= <personal> <apellido> [<trato>] <EOL>

| <personal> <nombre>

<personal> ::= <primer nombre> | <inicial> "."

<dirección> ::= [<departamento>] <número de la casa> <nombre de la calle> <EOL>

<apartado postal> ::= <ciudad> "," <código estado> <código postal> <EOL>

Esto puede interpretarse como que:

Una dirección postal consiste en un nombre, seguido por una dirección, seguido por un apartado postal.

Un nombre consiste en una parte personal seguido por un apellido, seguido opcionalmente por una jerarquía o el trato que se la da a la persona (Jr., Sr.) y un salto de línea (end-of-line), o bien una parte personal seguido por un nombre.

Una parte "personal" consiste en un nombre o una inicial seguido por un punto.

Una dirección consiste de una especificación opcional del departamento, seguido de un número de casa, seguido por el nombre de la calle, seguido por un salto de línea (end-of-line).

Un apartado postal consiste de una ciudad, seguido por una coma, seguido por un código del estado, seguido por un código postal y este seguido por un salto de línea (end-of-line).

Como otro ejemplo bastante interesante, se tiene que la sintaxis de BNF se puede representar en BNF como sigue:

<syntax> ::= <rule> [<syntax>]

<rule> ::= <whitespace> "<" <rule-name> ">" <whitespace> "::="

<expression> <whitespace> <expression> ")" | "[" <expression>

"]") [<list-expression>]

<whitespace> ::= [" " <whitespace>]

<line-end> ::= [<whitespace>] <EOL> [<line-end>]

Esto asume que no hay Whitespace necesario para la interpretación apropiada de la regla. El <QUOTE> se presume para ser el carácter ", y el <EOL> para ser el fin de línea apropiado especificado. El <rule-name> y el <text> deben ser sustituidos con nombre/etiqueta o el texto literal de una regla declarada, respectivamente.

3.2.1.2 Variantes

Hay muchas variantes y extensiones de BNF, conteniendo algunos o todos los elementos de expresiones regulares como un "*" o "+". El Extended Backus-Naur Form (EBNF) es una variante común. La notación de los corchetes "[ ]" fue introducida algunos años más tarde en la definición de PL/I de la IBM pero ahora se reconoce universal. La ABNF es otra extensión usada comúnmente para describir protocolos del IETF. [6]

Muchas especificaciones de BNF disponibles en línea tienen como propósito ser legibles a simple vista y no son especificaciones formales, éstas incluyen con frecuencia algunas las siguientes reglas sintácticas y extensiones:

- Los elementos opcionales son presentados entre corchetes. Por ejemplo [<elemento-x>].
- Los elementos que se repiten son presentados entre paréntesis de llave o terminados con un asterisco. Por ejemplo <palabra> ::= <letra> {<letra>}.

- Los elementos que se repiten 1 o más veces son terminados con un '+'.
- Los terminales pueden aparecer en negrita y los no-terminales en texto normal en lugar de utilizar paréntesis angulares.
- Las alternativas opcionales son separadas por el símbolo '|'.
- Cuando se requiere agrupar varios elementos, se hace con paréntesis simples.

3.2.2 BNF Extendido (EBNF)

Extended Backus-Naur Form (EBNF) es un conjunto de extensiones a Backus-Naur Form, no obstante no todas esas extensiones son estrictamente un superconjunto, ya que algunas cambian la relación de regla de definición ::= a = , mientras que otras eliminan los paréntesis angulares de los símbolos no terminales.

Más importante que las diferencias sintácticas menores entre las formas de EBNF son las operaciones adicionales permitidas en las expansiones. [7]

3.2.1.1 Opción

En EBNF, los corchetes alrededor de una expansión, [de expansión], indican que esta expansión es opcional.

Por ejemplo, la regla: <Term> ::= ["-"] <factor de>

<factor de> permite ser negado.

3.2.2.2 Repetición

En EBNF, las llaves indican que la expresión se puede repetir cero o más veces.

Por ejemplo, la regla: <Args> ::= <arg> { "," <arg>}

define una lista de argumentos separados por una coma convencional.

3.2.3.3 Agrupamiento

Para indicar precedencia, las gramáticas EBNF pueden utilizar paréntesis ( ), para definir el orden de expansión.

Por ejemplo, la regla: <Expr> ::= <term> ("+" | "-") <expr>

define una forma de expresión que permite tanto la suma como la resta.

3.2.3.4 Ejemplo

Ejemplo con BNF:

Gramática para representar números con decimales

número ::= entPos | entPos '.' entPos

entPos ::= dígito | dígito entPos

dígito ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Ejemplo anterior con E-BNF:

número ::= digito + ('.' digito +) ?

dígito ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

3.2.3 Forma Ampliada de Backus-Naur

En ciencias de la computación, la forma ampliada de Backus-Naur (ABNF) es un metalenguaje basado en Backus-Naur Form (BNF), pero que consta de sus propias reglas de sintaxis y derivación. El motivo principal para ABNF es describir un sistema formal de un lenguaje para ser utilizado como un protocolo bidireccional de comunicaciones. Se define por la normativa RFC 5234, que a menudo sirve como el lenguaje de definición de protocolos de comunicación IETF. La normativa RFC 5234 reemplaza a RFC 4234, mientras que RFC 7405 lo actualiza, se especifica la adición de una sintaxis para especificar los literales de cadena entre mayúsculas y minúsculas. [8]

3.2.3.1 Introducción

Una especificación de ABNF es un conjunto de reglas de derivación, escrita como:

rule = definition ; comment CR LF

donde la regla es una combinación no terminal entre mayúsculas y minúsculas, la definición consiste en secuencias de símbolos que definen la regla, un comentario de documentación, y termina con un retorno de carro y avance de línea.

Los nombres de regla son sensibles a mayúsculas: <rulename>, <Rulename>, <RULENAME> y <rUlENamE> todos se refieren a la misma regla. Los nombres de reglas consisten en una letra seguida de letras, números y guiones.

Los corchetes angulares (" < "," > ") no son necesarios alrededor de nombres de reglas, como si lo son en BNF.

3.2.3.2 Valores Terminales

Los valores terminales son especificados por uno o más caracteres numéricos, y dichos caracteres numéricos pueden especificarse como el signo de porcentaje " % ", seguido por la base (b = binario, d = decimal, y x = hexadecimal), seguido por el valor, o la concatenación de los valores (indicados por " . "). Por ejemplo se especifica un retorno de carro por d13% en decimal o %x0D en hexadecimal. Un retorno de carro seguido de un avance de línea se puede especificar con la concatenación como % d13.10.

El texto literal se especifica mediante el uso de una cadena entre comillas ("). Estas cadenas son sensibles a mayúsculas y el juego de caracteres utilizado es ASCII. Por lo tanto la cadena "abc" coincidirá con "abc", "Abc "," aBc "," abC "," ABc "," AbC "," aBC "y" ABC ". Para una mezcla entre mayúsculas y minúsculas deben definirse los caracteres explícitos. Para que " abc " coincida con la definición será % d97.66.99.

3.2.3.3 Operadores

El espacio en blanco

El espacio en blanco se utiliza para separar los elementos de una definición, para que el espacio sea reconocido como un delimitador debe incluirse explícitamente. La referencia explícita a un solo carácter es WSP (whitespace (espacio en blanco lineal)), o LWSP para cero o más espacios en blanco con las nuevas líneas permitidas. La definición de LWSP en la norma RFC 5234 indica que se necesita al menos un caracter de espacio en blanco para formar un delimitador de dos campos.

Las definiciones quedan alineadas a la izquierda. Cuando se requieren múltiples líneas, las líneas de continuación se sangran por espacios en blanco.

Comentario

Se describe como: ; comentario

Un punto y coma (" ; ") inicia un comentario que continúa hasta el final de la línea.

Concatenación

Es del tipo: Regla1 Regla2

Una regla puede ser definida por la inclusión de una secuencia de nombres de reglas.

Para que coincida con la cadena "aba" podrían utilizarse las siguientes reglas:

1.- fu =% x61; a

2.- bar =% x62; b

3.- mumble = fu bar fu

Alternativa

Se describe como: Regla1 / Regla2

Una regla puede ser definida por una lista de reglas alternativas separadas por una barra inclinada (" / ").

Para aceptar la regla fu o la regla bar, se podría construir la siguiente regla:

fubar = fu / bar

Alternativas incrementales

Se describe como: Regla1 = / Regla2

Alternativas adicionales pueden ser agregadas a una regla mediante el uso de " = / ", entre el nombre de la regla y la definición.

La regla

El conjunto de reglas = alt 1 / alt2 / alt3 / alt4 / alt5 ; es equivalente a:

1.- conjunto de reglas = alt 1 / alt2

2.- conjunto de reglas = / alt3

3.- conjunto de reglas = / alt4 / alt5

Rango de valores

Se describe como: % C ## - ##

Una cantidad de valores numéricos se pueden especificar mediante el uso de un guión (" - ").

La regla

Se describe como: OCTAL = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7"

Es equivalente a: OCTAL = % x30-37

Grupo de secuencias

Se describe como: (Regla1 Regla2)

Los elementos pueden ser colocados en paréntesis, como reglas de grupo en una definición.

Para hacer coincidir "elem fubar snafu" o "elem tarfu snafu", podría construirse la siguiente regla:

group = elem (fubar / tarfu) snafu

Para hacer coincidir " elem fubar " o " tarfu snafu", podrían construirse las siguientes reglas:

1.- group = elem fubar / tarfu snafu

2.- group = (elem fubar) / (tarfu snafu)

Repetición Variable

Se describe como: n * nRule

Para indicar la repetición de un elemento en la forma <a> * <b>, se utilizan dos opciones. La opción <a>, la cual da el número mínimo de elementos que se incluye con el valor por defecto que es 0. La opción <b> indica el número máximo de elementos que se incluirán en el valor por defecto que es infinito.

Se utiliza * element para cero o más elementos, * 1element para cero o un elemento, 1 * element para uno o más elementos, y 2 * 3element para dos o tres elementos.

Repetición específica

Se describe como: nRule

Para indicar un número explícito de elementos se utiliza <a> element y es equivalente a <a> * <a> elemento. Se utiliza 2 DIGIT para obtener dos dígitos numéricos y 3 DIGIT para tres dígitos numéricos.

Secuencia Opcional

Se describe como: [Rule]

Para indicar un elemento opcional, las siguientes construcciones son equivalentes:

1.- [fubar snafu]

2.- * 1 (fubar snafu)

3.- 0 * 1 (fubar snafu)

Normas básicas (Las reglas básicas se definen en la norma ABNF)

Regla	Definición Formal	Significado
ALPHA	X41-5A% /% x61-7A	Letras mayúsculas y minúsculas ASCII (A-Z, a-z)
DIGIT	% X30-39	Dígitos decimales (0-9)
HEXDIG	DIGIT / "A" / "B" / "C" / "D" / "E" / "F"	Los dígitos hexadecimales (0-9, A-F)
DQUOTE	% X22	Doble Comilla
SP	% X20	Espacio
HTAB	% X09	tabulador horizontal
WSP	SP / HTAB	espacio y tabulador horizontal
LWSP	* (WSP / CRLF WSP)	espacio en blanco lineal (más allá del salto de línea)
VCHAR	% X21-7E	(impresión) caracteres visibles
CHAR	% X01-7F	cualquier carácter ASCII, excluyendo NUL
OCTETO	% X00-FF	8 bits de datos
CTL	X00-1F% /% t x7F	controles
CR	% X0D	retorno de carro
LF	X0A%	salto de línea
CRLF	CR LF	Salto de línea estándar de Internet
BIT	"0" / "1"	dígito binario

Tabla 3-1: Comandos de ABNF

Ejemplo:

En el ejemplo de dirección postal de la forma de Backus-Naur (BNF), para la forma aumentada de Backus-Naur (ABNF) se puede especificar de la siguiente manera:

Dirección Postal = nombre-parte calle parte-postal

nombre-Parte = * (parte-personal SP) último-nombre [sufijo SP] CRLF

nombre-Parte = / parte-personal CRLF

parte personal = primer-nombre / (inicial ".")

primer nombre = * ALPHA

inicial = ALPHA

último nombre = * ALPHA

sufijo = ("Jr." / "Sr." / 1 * ("I" / "V" / "X"))

calle = [apt SP] número-casa SP nombre-calle CRLF

apt = 1 * 4 DIGIT

número-casa = 1 * 8 (DIGIT / ALPHA)

nombre-calle = 1 * VCHAR

parte-postal = nombre-ciudad "," SP estado 1 * 2SP código-postal CRLF

nombre-ciudad = 1 * (ALPHA / SP)

estado = 2ALPHA

código-postal = 5DIGIT ["-" 4DIGIT]

4 Análisis de Tráfico HTTP

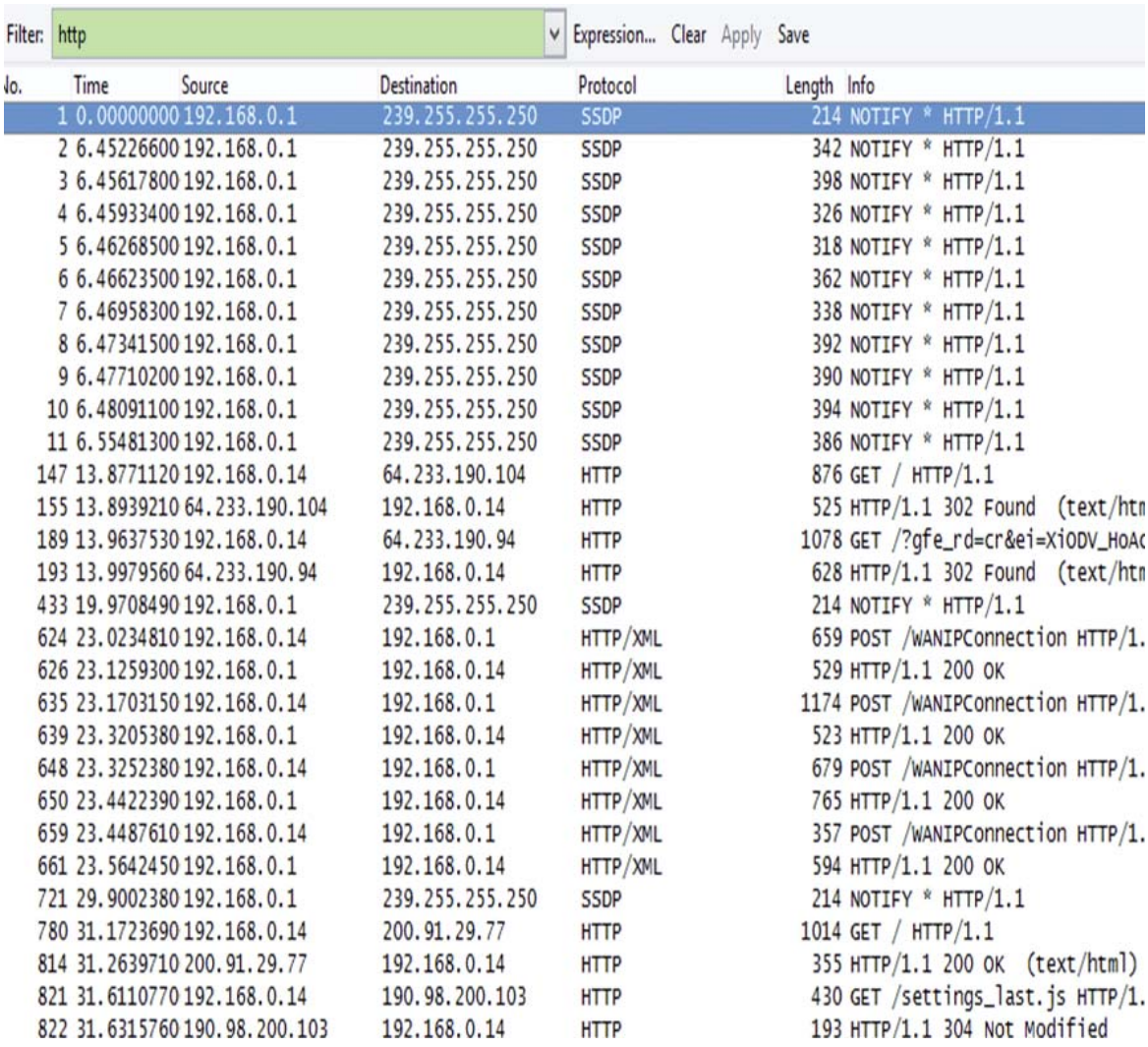
Para analizar el tráfico de paquetes de datos de la comunicación cliente-servidor, al abrir alguna página de Internet estando conectado a Wi-Fi, se tendrá como ejemplo la siguiente captura con Sniffer (figura 4.1):

Time	Source	Destination	Protocol	Length	Info
27 11.8797900	200.83.1.4	192.168.0.14	DNS	130	Standard query response 0x6b8c CNAME googlehosted.l.
28 11.8799480	192.168.0.14	200.83.1.4	DNS	75	Standard query 0x3d77 A ssl.gstatic.com
29 11.8919910	200.83.1.4	192.168.0.14	DNS	107	Standard query response 0x3d77 A 64.233.186.94 A 64.
30 11.9043690	192.168.0.14	200.83.1.4	DNS	75	Standard query 0x683a A www.gstatic.com
31 11.9166470	200.83.1.4	192.168.0.14	DNS	107	Standard query response 0x683a A 64.233.186.94 A 64.
32 12.2503460	fe80::8ddb:8e12:dd5	ff02::1:3	LLMNR	84	Standard query 0xc2f3 A wpad
33 12.2506360	192.168.0.14	224.0.0.252	LLMNR	64	Standard query 0xc2f3 A wpad
34 12.3839530	192.168.0.14	224.0.0.22	IGMPv3	54	Membership Report / Join group 224.0.0.251 for any so
35 12.5803430	192.168.0.14	192.168.0.255	NBNS	92	Name query NB WPAD<00>
36 12.7913470	192.168.0.14	200.83.1.4	DNS	76	Standard query 0xc24a A mtalk.google.com
37 12.7925730	192.168.0.14	64.233.190.104	TCP	66	62978→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 S
38 12.7931800	192.168.0.14	64.233.190.94	TCP	66	62979→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 S
39 12.7937030	192.168.0.14	64.233.190.94	TCP	66	62980→443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256
40 12.8051700	64.233.190.104	192.168.0.14	TCP	66	80→62978 [SYN, ACK] Seq=0 Ack=1 Win=42780 Len=0 MSS=1
41 12.8053650	192.168.0.14	64.233.190.104	TCP	54	62978→80 [ACK] Seq=1 Ack=1 Win=66048 Len=0
42 12.8065850	64.233.190.94	192.168.0.14	TCP	66	80→62979 [SYN, ACK] Seq=0 Ack=1 Win=42780 Len=0 MSS=1
43 12.8067620	192.168.0.14	64.233.190.94	TCP	54	62979→80 [ACK] Seq=1 Ack=1 Win=66048 Len=0
44 12.8070420	200.83.1.4	192.168.0.14	DNS	121	Standard query response 0xc24a CNAME mobile-gtalk.l.
45 12.8070440	64.233.190.94	192.168.0.14	TCP	66	443→62980 [SYN, ACK] Seq=0 Ack=1 Win=42780 Len=0 MSS=
46 12.8072650	192.168.0.14	64.233.190.94	TCP	54	62980→443 [ACK] Seq=1 Ack=1 Win=66048 Len=0
47 12.8079550	192.168.0.14	64.233.190.94	TLSv1.2	248	Client Hello

Fig. 4.1 Captura de tráfico de datos con conexión Wi-Fi mediante Sniffer. (Fuente: Sniffer (Wireshark))

Con lo cual se observa una cantidad enorme de transacciones relacionadas a otros protocolos y otras redes conectadas también a Wi-Fi, demasiada interferencia en el tráfico de información lo que hace prácticamente imposible poder analizar el tráfico relacionado a la página y al usuario.

Respecto a la captura anterior es posible utilizar un filtro HTTP para analizar el tráfico relacionado sólo al protocolo HTTP, como se observa en la figura 4.2:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	192.168.0.1	239.255.255.250	SSDP	214	NOTIFY * HTTP/1.1
2	6.45226600	192.168.0.1	239.255.255.250	SSDP	342	NOTIFY * HTTP/1.1
3	6.45617800	192.168.0.1	239.255.255.250	SSDP	398	NOTIFY * HTTP/1.1
4	6.45933400	192.168.0.1	239.255.255.250	SSDP	326	NOTIFY * HTTP/1.1
5	6.46268500	192.168.0.1	239.255.255.250	SSDP	318	NOTIFY * HTTP/1.1
6	6.46623500	192.168.0.1	239.255.255.250	SSDP	362	NOTIFY * HTTP/1.1
7	6.46958300	192.168.0.1	239.255.255.250	SSDP	338	NOTIFY * HTTP/1.1
8	6.47341500	192.168.0.1	239.255.255.250	SSDP	392	NOTIFY * HTTP/1.1
9	6.47710200	192.168.0.1	239.255.255.250	SSDP	390	NOTIFY * HTTP/1.1
10	6.48091100	192.168.0.1	239.255.255.250	SSDP	394	NOTIFY * HTTP/1.1
11	6.55481300	192.168.0.1	239.255.255.250	SSDP	386	NOTIFY * HTTP/1.1
147	13.8771120	192.168.0.14	64.233.190.104	HTTP	876	GET / HTTP/1.1
155	13.8939210	64.233.190.104	192.168.0.14	HTTP	525	HTTP/1.1 302 Found (text/htm
189	13.9637530	192.168.0.14	64.233.190.94	HTTP	1078	GET /?gfe_rd=cr&ei=XiODV_HOAC
193	13.9979560	64.233.190.94	192.168.0.14	HTTP	628	HTTP/1.1 302 Found (text/htm
433	19.9708490	192.168.0.1	239.255.255.250	SSDP	214	NOTIFY * HTTP/1.1
624	23.0234810	192.168.0.14	192.168.0.1	HTTP/XML	659	POST /WANIPConnection HTTP/1.
626	23.1259300	192.168.0.1	192.168.0.14	HTTP/XML	529	HTTP/1.1 200 OK
635	23.1703150	192.168.0.14	192.168.0.1	HTTP/XML	1174	POST /WANIPConnection HTTP/1.
639	23.3205380	192.168.0.1	192.168.0.14	HTTP/XML	523	HTTP/1.1 200 OK
648	23.3252380	192.168.0.14	192.168.0.1	HTTP/XML	679	POST /WANIPConnection HTTP/1.
650	23.4422390	192.168.0.1	192.168.0.14	HTTP/XML	765	HTTP/1.1 200 OK
659	23.4487610	192.168.0.14	192.168.0.1	HTTP/XML	357	POST /WANIPConnection HTTP/1.
661	23.5642450	192.168.0.1	192.168.0.14	HTTP/XML	594	HTTP/1.1 200 OK
721	29.9002380	192.168.0.1	239.255.255.250	SSDP	214	NOTIFY * HTTP/1.1
780	31.1723690	192.168.0.14	200.91.29.77	HTTP	1014	GET / HTTP/1.1
814	31.2639710	200.91.29.77	192.168.0.14	HTTP	355	HTTP/1.1 200 OK (text/html)
821	31.6110770	192.168.0.14	190.98.200.103	HTTP	430	GET /settings_last.js HTTP/1.
822	31.6315760	190.98.200.103	192.168.0.14	HTTP	193	HTTP/1.1 304 Not Modified

Fig. 4.2 Captura de tráfico de datos con filtro HTTP. (Fuente: Sniffer (Wireshark))

Ahora para rescatar la captura de alguna transacción HTTP con SNIFFER, se tendrá la figura 4.3:

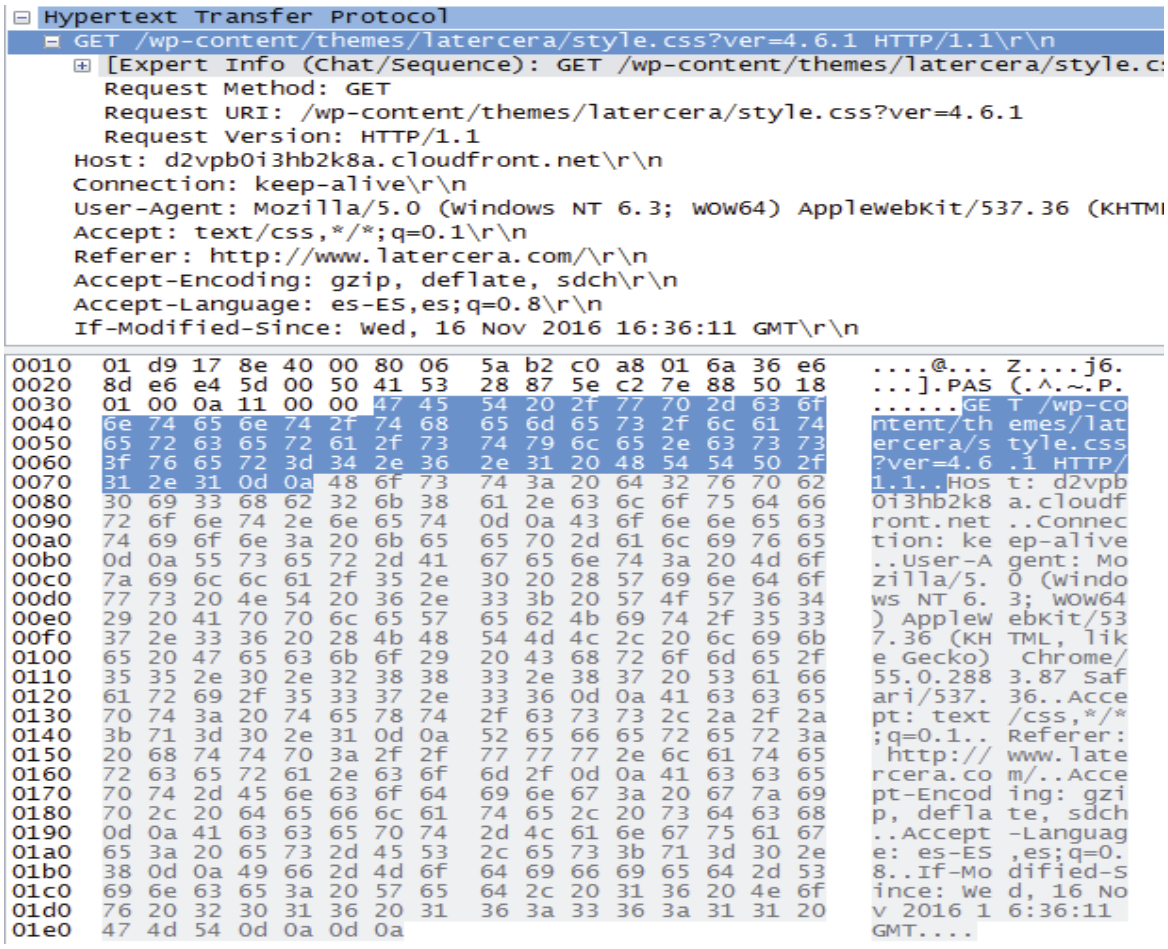


Fig. 4.3 Captura de Transacción HTTP. (Fuente: Sniffer (Wireshark))

Aquí se observa una transacción HTTP de petición GET con distintos encabezados capturada con SNIFFER, y también debajo del contenido de la transacción se observa en el lado derecho el código ASCII de la transacción, que es un código de caracteres basado en el alfabeto latino y utiliza 7 bits para representar los caracteres. Casi todos los sistemas informáticos actuales utilizan el código ASCII o una extensión compatible para representar textos y para el control de dispositivos que manejan texto como el teclado. Para la captura se describe el contenido de la transacción en lenguaje binario hexadecimal y su correspondiente representación. No obstante este código no será de importancia en la descripción del protocolo, pero si para la creación del servidor HTTP.

Pero aún así es muy difícil poder analizar con Wi-Fi, ya que circula demasiada interferencia de datos relacionadas a otras redes conectadas a Wi-Fi.

Para poder realizar un análisis mucho más preciso sobre las transacciones involucradas en el protocolo y a la comunicación cliente-servidor, es que se ha decidido utilizar un HUB del laboratorio de sistemas digitales, el cual contiene el servidor de página 3com, esto permitirá realizar un análisis detallado y un estudio del tráfico local de información.

En la figura 4.4 se tiene una imagen del HUB:

Fig. 4.4 Equipo de red (HUB) del laboratorio de sistemas digitales. (Fuente: Elaboración propia)

4.1 Comunicación cliente-servidor

Al conectarse un usuario desde cualquier PC a Internet, se está comunicando con el determinado servidor al cual está conectada esa página, el usuario está entrando en contacto con el servidor de página mediante el protocolo HTTP, dicha comunicación también involucra otros protocolos que al igual que el HTTP son derivados de TCP, como el caso del ARP, esa comunicación y el tráfico de paquetes de datos es posible analizarla mediante el SNIFFER, el analizador de paquetes de datos, el cual permitirá validar todas las conexiones pertinentes y el traspaso de paquetes de datos que se pretenden crear en este proyecto para el servidor HTTP.

4.2 Diagramas Secuenciales

Mediante los diagramas secuenciales se puede analizar el proceso de la comunicación cliente-servidor, en el proceso de crear el servidor HTTP, mediante los diagramas secuenciales, que no

son otra cosa más que esquemas de conexiones y de traspasos de paquetes de datos mediante líneas que indican el flujo de información entre las partes involucradas en la comunicación, con esto el cliente puede planificar como quiere que se lleven a cabo los traspasos de datos en la comunicación cliente-servidor para posteriormente validar todo ese proceso con el SNIFFER.

4.3 Transacciones HTTP

Al haber comunicación cliente-servidor hay transferencias de información en forma de paquetes de datos, dichas transferencias es lo que se denominan transacciones HTTP y es lo que se pretende analizar con el SNIFFER. Este proceso de análisis de las transacciones HTTP, será de vital importancia para posteriormente planificar como se creará el servidor HTTP en este proyecto.

4.4 Topologías de Red

Mediante distintos esquemas de conexiones de red, se analizará el funcionamiento del protocolo HTTP y el correspondiente proceso de la comunicación cliente-servidor con SNIFFER.

4.4.1 Red con SNIFFER de WINDOWS

En la figura 4.5 se observa el esquema de red utilizado:

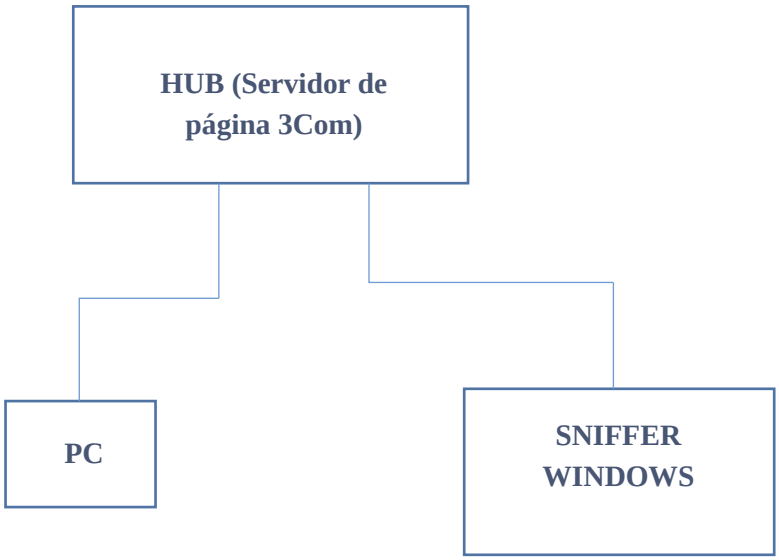


Fig. 4.5 Esquema de red con SNIFFER de WINDOWS. (Fuente: Elaboración propia)

Este es el esquema original de red de computadores en el cual se observa el HUB con su correspondiente servidor de página dentro de él, el servidor 3Com, conectado a un PC de prueba y otro PC con un SNIFFER de WINDOWS abierto para analizar el tráfico de información durante el proceso de la comunicación cliente-servidor en forma transferencia de paquetes de datos. Este esquema servirá para elaborar un correspondiente diagrama de secuencias sobre cómo se pretende llevar a cabo la comunicación entre el usuario y el servidor para posteriormente validar todo eso con el SNIFFER de WINDOWS que al igual que el PC de prueba está conectado al HUB.

Entonces el diagrama de secuencias de este esquema de red está planeado para que la comunicación se lleve a cabo como se observa en la figura 4.6:

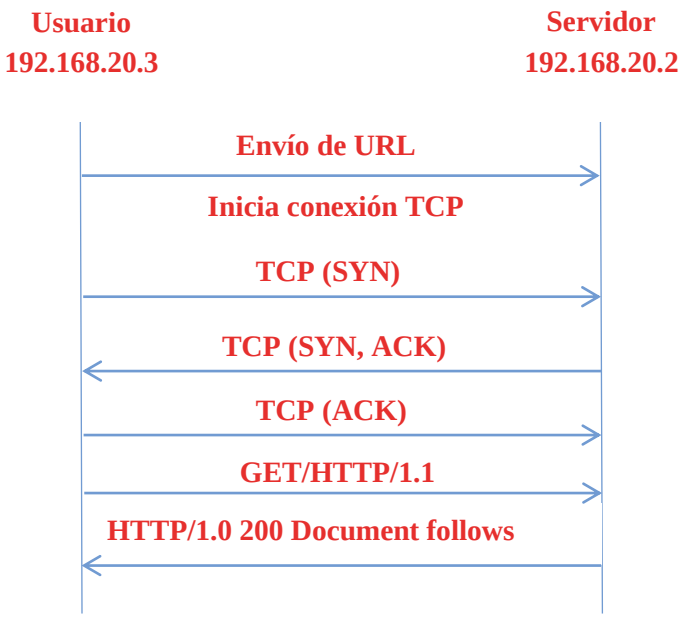


Fig. 4.6 Diagrama secuencial de inicio de la comunicación cliente-servidor para red con SNIFFER de WINDOWS. (Fuente: Elaboración propia)

Es decir, está pensado que la comunicación se lleve a cabo de manera en la cual el usuario entre en contacto con el servidor ingresando la URL correspondiente, que en este caso es la dirección IP del servidor (192.168.20.2), y comiencen a establecerse las conexiones TCP correspondientes, en la cual la etapa SYN indica un pedido para establecer una conexión, mientras que ACK es un acuse de recibo. Posteriormente se realizarán las conexiones HTTP involucradas, que serían GET para pedir el archivo correspondiente y el código de respuesta 200 OK para indicar que el archivo ha sido entregado al usuario. Sin embargo, una vez que se ingresa la URL y comienzan a establecerse las conexiones TCP, la página mostrada en pantalla se observa en la figura 4.7:

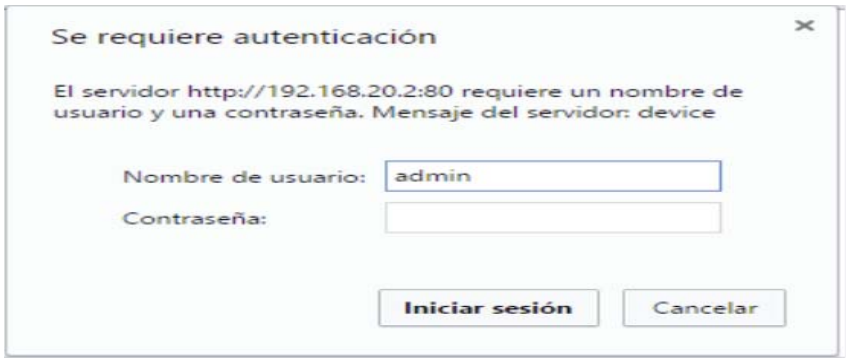


Fig. 4.7 Etapa de autenticación. (Fuente: dirección IP: 192.168.20.2)

Se observa que el servidor requiere de una etapa de autenticación para entrar en contacto con el usuario, es decir el HUB está programado para que la comunicación se realice de la siguiente manera, como se observa en la figura 4.8:

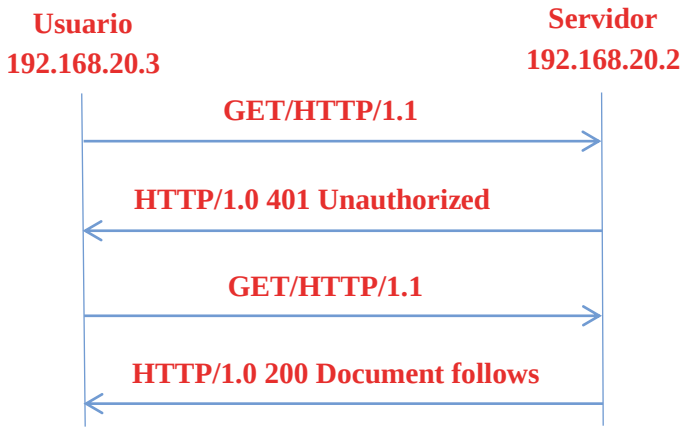


Fig. 4.8 Diagrama secuencial de etapa de autenticación. (Fuente: Elaboración propia)

Es decir, una vez que el servidor acepte la etapa de autenticación entrará en contacto correctamente con el usuario para mostrar el archivo correspondiente. Para que el servidor acepte la etapa de autenticación, se debe ingresar la contraseña correspondiente que como se muestra en la figura anterior será la clave admin y se escribirá en nombre de usuario. Una vez que se ingresó admin, se mostrará la siguiente página correspondiente al archivo que posee el servidor de página 3Com del HUB, la cual se observa en la figura 4.9.

Fig. 4.9 Página web de contacto llevado a cabo correctamente entre el usuario y el servidor. (Fuente: dirección IP: 192.168.20.2)

Esta es la página web con autenticación que muestra el servidor de página del HUB. Al tener abierta esta página y entrando en contacto con el servidor, es posible analizar la comunicación cliente-servidor visualizada con SNIFFER de WINDOWS, la cual es posible de apreciar en la figura 4.10.

No.	Time	Source	Destination	Protocol	Info
11	6.007428	192.168.20.3	192.168.20.2	HTTP	GET / HTTP/1.1
12	6.039659	192.168.20.2	192.168.20.3	HTTP	HTTP/1.0 401 Unauthorized
32	10.265800	192.168.20.3	192.168.20.2	HTTP	GET / HTTP/1.1
33	10.339140	192.168.20.2	192.168.20.3	HTTP	HTTP/1.0 200 Document follows
36	11.097876	192.168.20.2	192.168.20.3	HTTP	Continuation or non-HTTP traffic
45	11.161427	192.168.20.3	192.168.20.2	HTTP	GET /style/banner.rhtm HTTP/1.1
49	11.170188	192.168.20.3	192.168.20.2	HTTP	GET /style/tb.rhtm?1 HTTP/1.1
53	11.472356	192.168.20.3	192.168.20.2	HTTP	[TCP Retransmission] GET /style/tb.rhtm?1 HTTP/1.1
56	11.910765	192.168.20.3	192.168.20.2	HTTP	GET /mimic HTTP/1.1
65	12.424244	192.168.20.2	192.168.20.3	HTTP	HTTP/1.0 200 Document follows
71	13.882051	192.168.20.2	192.168.20.3	HTTP	HTTP/1.0 200 Document follows (text/html)
75	14.273341	192.168.20.2	192.168.20.3	HTTP	Continuation or non-HTTP traffic
82	14.305690	192.168.20.3	192.168.20.2	HTTP	GET /dev01/mimic/index.rhtm HTTP/1.1
86	14.893920	192.168.20.2	192.168.20.3	HTTP	HTTP/1.0 200 Document follows (text/html)
94	15.150785	192.168.20.3	192.168.20.2	HTTP	GET /dev01/mimic/mimic.rhtm HTTP/1.1
97	15.218234	192.168.20.3	192.168.20.2	HTTP	GET /dev01/us/dev_stat.rhtm HTTP/1.1
104	16.432939	192.168.20.2	192.168.20.3	HTTP	HTTP/1.0 200 Document follows (text/html)
121	21.110258	192.168.20.2	192.168.20.3	HTTP	HTTP/1.0 200 Document follows (text/html)
131	24.126584	192.168.20.2	192.168.20.3	HTTP	HTTP/1.0 200 Document follows (text/html)
138	24.162318	192.168.20.3	192.168.20.2	HTTP	GET /favicon.ico HTTP/1.1

Fig. 4.10 Captura de transacciones HTTP con SNIFFER de WINDOWS. (Fuente: Sniffer (Wireshark))

Al capturar el tráfico de paquetes de datos, cabe destacar que el SNIFFER de WINDOWS genera por si mismo muchísimas tramas relacionadas a otros protocolos que no tienen relación alguna con TCP y HTTP, y que contaminan el tráfico de información, y como dichas tramas no son relevantes, se utilizó un filtro HTTP del cual dispone el SNIFFER de WINDOWS, y se obtuvo una captura de tramas mucho más limpia y clara para analizar el tráfico de información. Se observa primero la etapa de autenticación por la cual pasó el servidor. Y una vez que la solicitud se cumplió y el servidor mostró el archivo que se pedía, de la página web principal se obtuvieron otros recursos derivados de ella como lo son cuadros de texto, imágenes, íconos y fondo de pantalla.

4.4.2 Red con 2 SNIFFER

En la figura 4.11 se tiene el esquema de red utilizado:

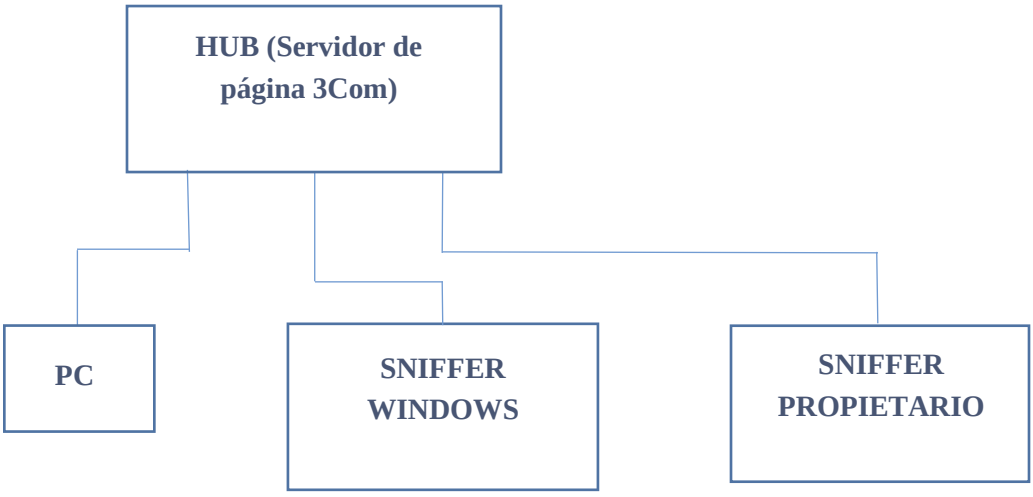


Fig. 4.11 Esquema de red con 2 SNIFFER. (Fuente: Elaboración propia)

En este nuevo esquema se creó un SNIFFER en sistema operativo propietario, el cual al igual que el sistema también está en constante avance y desarrollo, entonces para mostrar de manera sencilla como funciona se tendrá la siguiente captura mostrada en la figura 4.12:

Fig. 4.12 Captura de una transacción con SNIFFER Propietario. (Fuente: Sniffer Propietario)

Con esta captura se observa que el número de tramas que aparece en la parte superior de la captura, ya sea de trama tope actual o de tamaño, están todos esos valores en hexadecimal, mientras que en el recuadro de abajo se puede apreciar la información de dirección MAC e IP, además del protocolo y el puerto al que pertenece la transacción, indicando la fuente y el destino al cual pertenece dicha información.

Con este SNIFFER en sistema propietario, el objetivo es poder analizar de manera más amplia la información de comunicación cliente-servidor, sin tener que aplicar filtro HTTP, ya que también hay información relacionada a otros protocolos que es importante de analizar. Entonces con este nuevo esquema de red sincronizando la captura de tramas con ambos SNIFFER, se tendrá la siguiente captura mostrada en la figura 4.13:

	Source	Destination	Protocol	Info
0000	34:64:a9:be:bb:6b	Broadcast	ARP	Who has 192.168.20.1? Tell 192.168.20.4
0472	34:64:a9:be:bb:6b	Broadcast	ARP	Who has 192.168.20.1? Tell 192.168.20.4
3831	34:64:a9:be:bb:6b	Broadcast	ARP	Who has 192.168.20.1? Tell 192.168.20.4
3782	34:64:a9:be:bb:6b	Broadcast	ARP	Who has 192.168.20.2? Tell 192.168.20.4
5852	3comEuro_f9:bc:c8	4:64:a9:be:bb:6b	ARP	192.168.20.2 is at 00:90:04:f9:bc:c8
7055	192.168.20.4	192.168.20.2	TCP	52957 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_P
4835	192.168.20.2	192.168.20.4	TCP	http > 52957 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
5154	192.168.20.4	192.168.20.2	TCP	52957 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
5626	192.168.20.4	192.168.20.2	HTTP	GET / HTTP/1.1
7505	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 401 Unauthorized
0894	192.168.20.2	192.168.20.4	TCP	http > 52957 [FIN, ACK] Seq=88 Ack=371 Win=1024 Len=0
1162	192.168.20.4	192.168.20.2	TCP	52957 > http [ACK] Seq=371 Ack=89 Win=16529 Len=0
4301	192.168.20.4	192.168.20.2	TCP	52953 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_P
4397	192.168.20.4	192.168.20.2	TCP	52955 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_P
5310	192.168.20.4	192.168.20.2	TCP	52952 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_P
5386	192.168.20.4	192.168.20.2	TCP	52954 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_P
5446	192.168.20.4	192.168.20.2	TCP	52956 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_P
2441	192.168.20.2	192.168.20.4	TCP	http > 52953 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
2913	192.168.20.4	192.168.20.2	TCP	52953 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
0881	192.168.20.2	192.168.20.4	TCP	http > 52955 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
1432	192.168.20.4	192.168.20.2	TCP	52955 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
9882	192.168.20.2	192.168.20.4	TCP	http > 52952 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
0382	192.168.20.4	192.168.20.2	TCP	52952 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
3463	192.168.20.2	192.168.20.4	TCP	http > 52954 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
3944	192.168.20.4	192.168.20.2	TCP	52954 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
7676	192.168.20.2	192.168.20.4	TCP	http > 52956 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
8173	192.168.20.4	192.168.20.2	TCP	52956 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
7397	192.168.20.4	192.168.20.2	TCP	52957 > http [FIN, ACK] Seq=371 Ack=89 Win=16529 Len=0
7995	192.168.20.4	192.168.20.2	HTTP	GET / HTTP/1.1

Fig. 4.13 Comienzo de la captura de transacciones de paquetes de datos con SNIFFER de Windows para la red con 2 SNIFFER. (Fuente: Sniffer (Wireshark))

Aquí se observa el tráfico de información de forma limpia y clara, sin tener que recurrir a filtros, para protocolos ARP, TCP y HTTP, y se observa como antes de entrar en conexiones TCP, el servidor entra en conexiones ARP con el usuario, una vez que se ha reconocido la dirección IP y MAC del servidor, se inician las conexiones TCP, correspondientes a los segmentos SYN y ACK, antes de entrar en conexiones HTTP, y también después de las conexiones HTTP hay conexiones TCP, pertenecientes a distintos puertos. Cabe destacar que en este ejemplo a diferencia del anterior, aquí se cambió la IP del usuario a 192.168.20.4. Pero el tráfico de datos continúa, como se observa en la figura 4.14:

	Source	Destination	Protocol	Info
7995	192.168.20.4	192.168.20.2	HTTP	GET / HTTP/1.1
5204	192.168.20.2	192.168.20.4	TCP	http > 52957 [ACK] Seq=89 Ack=372 Win=1024 Len=0
5856	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows
9437	192.168.20.4	192.168.20.2	TCP	52953 > http [ACK] Seq=402 Ack=64 Win=16553 Len=0
9825	192.168.20.2	192.168.20.4	HTTP	Continuation or non-HTTP traffic
1683	192.168.20.2	192.168.20.4	TCP	http > 52953 [FIN, ACK] Seq=400 Ack=402 Win=1024 Len=0
2128	192.168.20.4	192.168.20.2	TCP	52953 > http [ACK] Seq=402 Ack=401 Win=16217 Len=0
2470	192.168.20.4	192.168.20.2	TCP	52953 > http [FIN, ACK] Seq=402 Ack=401 Win=16217 Len=0
5131	192.168.20.2	192.168.20.4	TCP	http > 52953 [ACK] Seq=401 Ack=403 Win=1024 Len=0
4593	192.168.20.4	192.168.20.2	HTTP	GET /style/banner.rhtm HTTP/1.1
9143	192.168.20.4	192.168.20.2	HTTP	GET /style/tb.rhtm?1 HTTP/1.1
2310	192.168.20.4	192.168.20.2	HTTP	GET /mimic HTTP/1.1
4021	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
3585	192.168.20.4	192.168.20.2	TCP	52955 > http [ACK] Seq=450 Ack=89 Win=16528 Len=0
6475	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
6544	192.168.20.4	192.168.20.2	TCP	52952 > http [ACK] Seq=448 Ack=89 Win=16528 Len=0
3637	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows
3709	192.168.20.4	192.168.20.2	TCP	52954 > http [ACK] Seq=438 Ack=64 Win=16553 Len=0
9186	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
8780	192.168.20.4	192.168.20.2	TCP	52952 > http [ACK] Seq=448 Ack=513 Win=16104 Len=0
8955	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
8693	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows (text/html)
9111	192.168.20.4	192.168.20.2	TCP	52955 > http [ACK] Seq=450 Ack=697 Win=15921 Len=0
9507	192.168.20.4	192.168.20.2	TCP	52955 > http [FIN, ACK] Seq=450 Ack=697 Win=15921 Len=0
7871	192.168.20.2	192.168.20.4	HTTP	Continuation or non-HTTP traffic
1452	192.168.20.2	192.168.20.4	TCP	http > 52954 [FIN, ACK] Seq=144 Ack=438 Win=1024 Len=0
1867	192.168.20.4	192.168.20.2	TCP	52954 > http [ACK] Seq=438 Ack=145 Win=16473 Len=0
2384	192.168.20.4	192.168.20.2	TCP	52954 > http [FIN, ACK] Seq=438 Ack=145 Win=16473 Len=0
0557	192.168.20.2	192.168.20.4	TCP	http > 52955 [ACK] Seq=697 Ack=451 Win=1024 Len=0

Fig. 4.14 Segunda captura del tráfico de paquetes de datos para red con 2 SNIFFER. (Fuente: Sniffer (Wireshark))

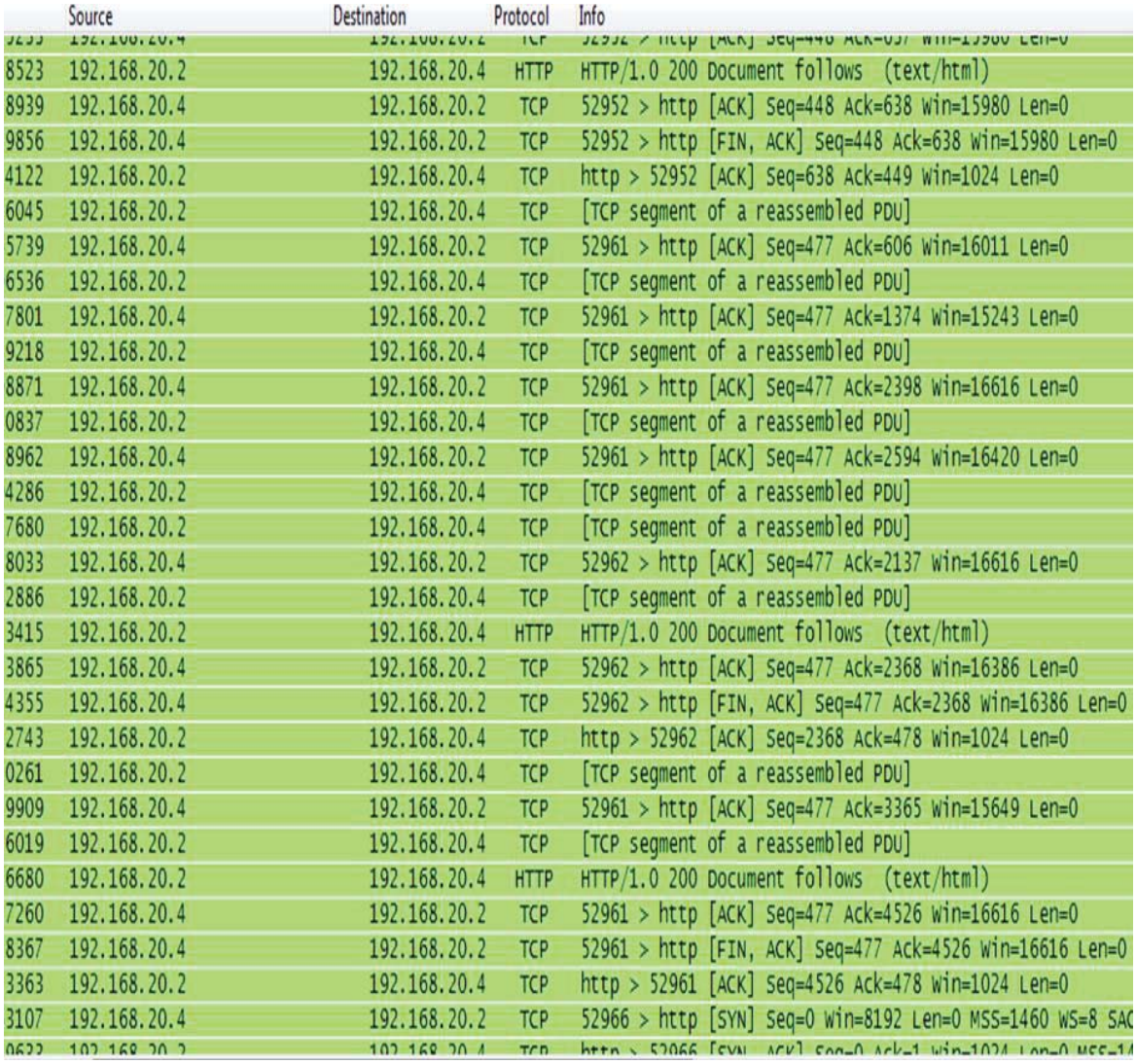
Aquí se observa una continuación en la captura de transacciones y se aprecia la primera aceptación del servidor a la petición del usuario, para luego dar paso a más conexiones TCP, esta vez con segmentos ACK y FIN, para después comenzar con las peticiones GET sobre recursos derivados del documento principal como lo son style/banner y mimic. En la figura 4.15 el tráfico continúa:

	Source	Destination	Protocol	Info
1867	192.168.20.4	192.168.20.2	TCP	52954 > http [ACK] Seq=438 Ack=145 Win=16473 Len=0
2384	192.168.20.4	192.168.20.2	TCP	52954 > http [FIN, ACK] Seq=438 Ack=145 Win=16473 Len=0
0557	192.168.20.2	192.168.20.4	TCP	http > 52955 [ACK] Seq=697 Ack=451 Win=1024 Len=0
0924	192.168.20.4	192.168.20.2	HTTP	GET /dev01/mimic/index.rhtm HTTP/1.1
6133	192.168.20.2	192.168.20.4	TCP	http > 52954 [ACK] Seq=145 Ack=439 Win=1024 Len=0
1993	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
5162	192.168.20.4	192.168.20.2	TCP	52956 > http [ACK] Seq=460 Ack=89 Win=16528 Len=0
7739	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows (text/html)
8260	192.168.20.4	192.168.20.2	TCP	52956 > http [ACK] Seq=460 Ack=312 Win=16306 Len=0
9129	192.168.20.4	192.168.20.2	TCP	52956 > http [FIN, ACK] Seq=460 Ack=312 Win=16306 Len=0
1101	192.168.20.4	192.168.20.2	TCP	52961 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_
0941	192.168.20.4	192.168.20.2	TCP	52962 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_
7024	192.168.20.2	192.168.20.4	TCP	http > 52956 [ACK] Seq=312 Ack=461 Win=1024 Len=0
7895	192.168.20.2	192.168.20.4	TCP	http > 52961 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
8389	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
0286	192.168.20.4	192.168.20.2	HTTP	GET /dev01/mimic/mimic.rhtm HTTP/1.1
2637	192.168.20.2	192.168.20.4	TCP	http > 52962 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
3172	192.168.20.4	192.168.20.2	TCP	52962 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
5187	192.168.20.4	192.168.20.2	HTTP	GET /dev01/us/dev_stat.rhtm HTTP/1.1
6453	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
6387	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=477 Ack=89 Win=16528 Len=0
6165	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
6698	192.168.20.4	192.168.20.2	TCP	52962 > http [ACK] Seq=477 Ack=89 Win=16528 Len=0
9907	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
9964	192.168.20.4	192.168.20.2	TCP	52952 > http [ACK] Seq=448 Ack=623 Win=15994 Len=0
4167	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
5233	192.168.20.4	192.168.20.2	TCP	52952 > http [ACK] Seq=448 Ack=637 Win=15980 Len=0
8523	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows (text/html)
8939	192.168.20.4	192.168.20.2	TCP	52952 > http [ACK] Seq=448 Ack=638 Win=15980 Len=0

Fig. 4.15 Tercera captura de transacciones para la red con 2 SNIFFER. (Fuente: Sniffer (Wireshark))

En esta tercera captura se observan más conexiones TCP, esta vez usando los tres segmentos, SYN, ACK y FIN, además de mostrar más peticiones GET por parte del usuario sobre recursos derivados de la página principal, con su correspondiente código de aceptación.

En la figura 4.16 el tráfico continúa:



	Source	Destination	Protocol	Info
8523	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows (text/html)
8939	192.168.20.4	192.168.20.2	TCP	52952 > http [ACK] Seq=448 Ack=638 Win=15980 Len=0
9856	192.168.20.4	192.168.20.2	TCP	52952 > http [FIN, ACK] Seq=448 Ack=638 Win=15980 Len=0
4122	192.168.20.2	192.168.20.4	TCP	http > 52952 [ACK] Seq=638 Ack=449 Win=1024 Len=0
6045	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
5739	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=477 Ack=606 Win=16011 Len=0
6536	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
7801	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=477 Ack=1374 Win=15243 Len=0
9218	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
8871	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=477 Ack=2398 Win=16616 Len=0
0837	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
8962	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=477 Ack=2594 Win=16420 Len=0
4286	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
7680	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
8033	192.168.20.4	192.168.20.2	TCP	52962 > http [ACK] Seq=477 Ack=2137 Win=16616 Len=0
2886	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
3415	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows (text/html)
3865	192.168.20.4	192.168.20.2	TCP	52962 > http [ACK] Seq=477 Ack=2368 Win=16386 Len=0
4355	192.168.20.4	192.168.20.2	TCP	52962 > http [FIN, ACK] Seq=477 Ack=2368 Win=16386 Len=0
2743	192.168.20.2	192.168.20.4	TCP	http > 52962 [ACK] Seq=2368 Ack=478 Win=1024 Len=0
0261	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
9909	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=477 Ack=3365 Win=15649 Len=0
6019	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
6680	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows (text/html)
7260	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=477 Ack=4526 Win=16616 Len=0
8367	192.168.20.4	192.168.20.2	TCP	52961 > http [FIN, ACK] Seq=477 Ack=4526 Win=16616 Len=0
3363	192.168.20.2	192.168.20.4	TCP	http > 52961 [ACK] Seq=4526 Ack=478 Win=1024 Len=0
3107	192.168.20.4	192.168.20.2	TCP	52966 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SAC

Fig. 4.16 Cuarta captura del tráfico de transacciones para la red con 2 SNIFFER. (Fuente: Sniffer (Wireshark))

En esta cuarta captura se observan más conexiones TCP, con los segmentos SYN, ACK y FIN, además de mostrar algunos códigos más de aceptación que habían quedado pendientes sobre algunas peticiones GET a los recursos derivados de la página principal que mostraba el servidor.

En la figura 4.17 se observa la última etapa del tráfico en análisis:

	Source	Destination	Protocol	Info
6680	192.168.20.2	192.168.20.4	HTTP	HTTP/1.0 200 Document follows (text/html)
7260	192.168.20.4	192.168.20.2	TCP	52961 > http [ACK] Seq=477 Ack=4526 Win=16616 Len=0
8367	192.168.20.4	192.168.20.2	TCP	52961 > http [FIN, ACK] Seq=477 Ack=4526 Win=16616 Len=0
3363	192.168.20.2	192.168.20.4	TCP	http > 52961 [ACK] Seq=4526 Ack=478 Win=1024 Len=0
3107	192.168.20.4	192.168.20.2	TCP	52966 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_
9633	192.168.20.2	192.168.20.4	TCP	http > 52966 [SYN, ACK] Seq=0 Ack=1 Win=1024 Len=0 MSS=1436
9922	192.168.20.4	192.168.20.2	TCP	52966 > http [ACK] Seq=1 Ack=1 Win=16616 Len=0
0714	192.168.20.4	192.168.20.2	HTTP	GET /favicon.ico HTTP/1.1
8499	192.168.20.2	192.168.20.4	TCP	[TCP segment of a reassembled PDU]
1988	192.168.20.2	192.168.20.4	TCP	http > 52966 [FIN, ACK] Seq=104 Ack=343 Win=1024 Len=0
2466	192.168.20.4	192.168.20.2	TCP	52966 > http [ACK] Seq=343 Ack=105 Win=16513 Len=0
2734	192.168.20.4	192.168.20.2	TCP	52966 > http [FIN, ACK] Seq=343 Ack=105 Win=16513 Len=0
8030	192.168.20.2	192.168.20.4	TCP	http > 52966 [ACK] Seq=105 Ack=344 Win=1024 Len=0
2843	34:64:a9:be:bb:6b	Broadcast	ARP	Who has 192.168.20.1? Tell 192.168.20.4
0930	34:64:a9:be:bb:6b	Broadcast	ARP	Who has 192.168.20.1? Tell 192.168.20.4
0956	34:64:a9:be:bb:6b	Broadcast	ARP	Who has 192.168.20.1? Tell 192.168.20.4
1172	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
6161	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
6884	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
1140	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
8786	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
8069	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
0766	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
6321	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
6887	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
1279	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
8813	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
8326	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
0835	192.168.20.4	239.255.255.250	SSDP	NOTIFY * HTTP/1.1

Fig. 4.17 Quinta captura de transacciones para la red con 2 SNIFFER. (Fuente: Sniffer (Wireshark))

Para esta quinta y última captura que se ha hecho a las transacciones para la red con 2 SNIFFER se observan las últimas conexiones TCP y HTTP antes de llegar a las conexiones ARP en las cuales al igual que al comienzo de las capturas el usuario pregunta al servidor si posee la dirección IP (192.168.20.1), ante lo cual el servidor no responde, dando a entender que no la posee, y se da paso a las conexiones NOTIFY, las cuales no serán de interés para el análisis.

4.4.3 Red con SNIFFER Propietario

En la figura 4.18 se tiene el esquema de red utilizado:

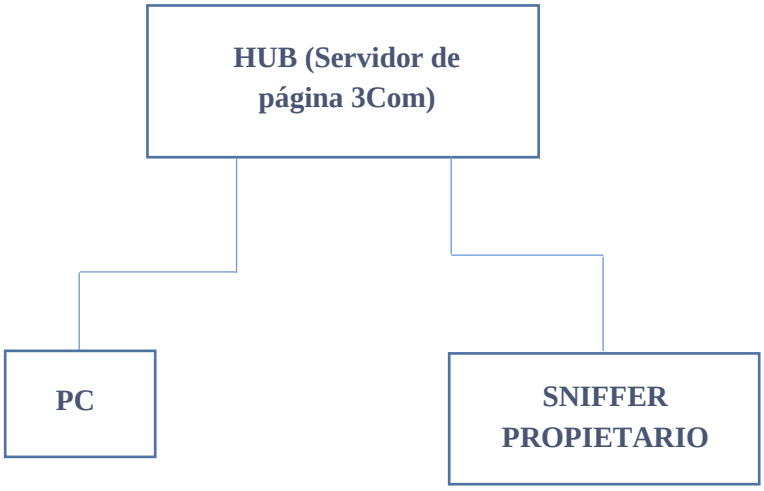


Fig. 4.18 Esquema de red con SNIFFER Propietario. (Fuente: Elaboración propia)

En este esquema se pretende analizar el tráfico de información sólo con el SNIFFER del sistema operativo propietario, el cual está en permanente desarrollo, pero como en general sólo interesa el tráfico de paquetes de datos de los protocolos ARP, TCP y HTTP, se analizará que cumpla con el tráfico visto en el esquema anterior con SNIFFER de WINDOWS, pero en este SNIFFER propietario aún no es posible observar la comunicación cliente-servidor de la misma manera que con el de WINDOWS, ya que no muestra todo el tráfico de transacciones, sino que va mostrando una por una, tal y como se vió en la captura, es que se utilizará un diagrama de secuencias mostrando como es el tráfico que interesa analizar para validar lo que se vió en el esquema anterior, el cual se observa en la figura 4.19.

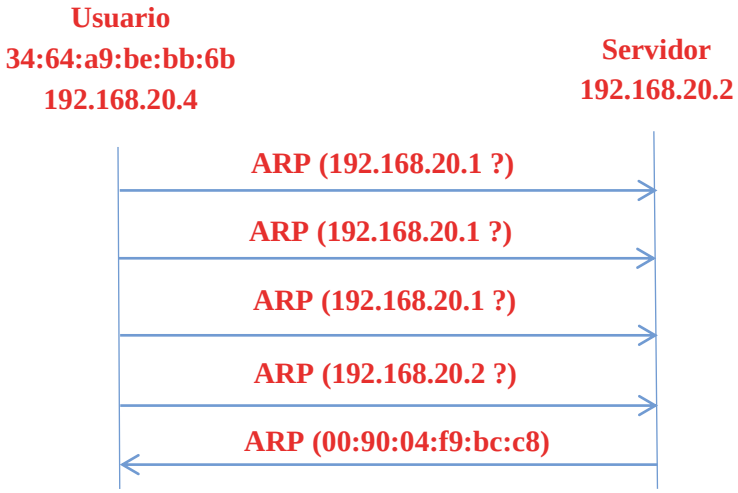


Fig. 4.19 Diagrama secuencial de comienzo de la comunicación cliente-servidor de la red con SNIFFER PROPIETARIO. (Fuente: Elaboración propia)

Este diagrama muestra que cuando comienza el proceso de la comunicación cliente-servidor, el usuario envía una consulta mediante el protocolo ARP al servidor si posee la dirección IP (192.168.20.1), hace dicha consulta 3 veces y el servidor no responde dando a entender que no la posee, pero después de eso viene una consulta por la IP (192.168.20.2), ante la cual el servidor le responde al usuario indicándole que posee la dirección MAC (00:90:04:f9:bc:c8), dando a entender que efectivamente él posee la IP (192.168.20.2). No obstante la comunicación continúa, por lo tanto se tendrán más diagramas de secuencias, como el que se observa en la figura 4.20:

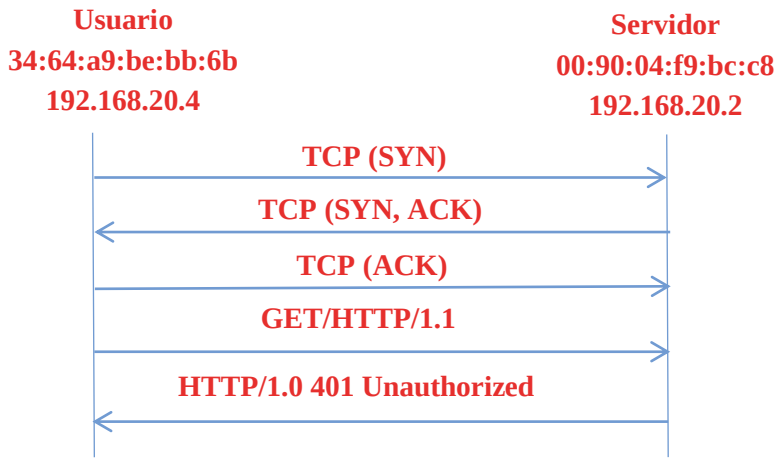


Fig. 4.20 Diagrama secuencial de etapa de autenticación del servidor 3Com. (Fuente: Elaboración propia)

Posteriormente, una vez que se confirma la dirección IP y MAC del servidor se inician las conexiones TCP pertinentes, pertenecientes a los segmentos SYN y ACK, para luego dar paso a las conexiones HTTP, ante lo cual se observa que al igual que en el primer ejemplo de esquema de red cuando el usuario realiza una petición GET para acceder a la información del servidor, éste requiere de una etapa de autenticación, con lo cual le devuelve un código de respuesta 401 no autorizado. Con esto se dará paso a otra etapa más de comunicación, como se observa en la figura 4.21:

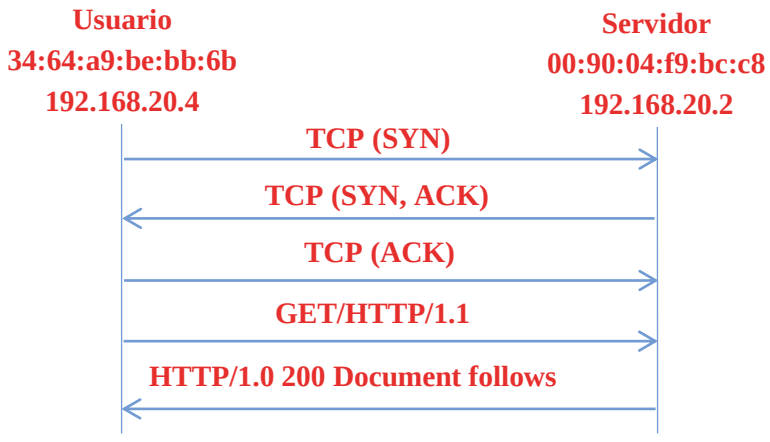


Fig. 4.21 Diagrama secuencial de etapa de aceptación del servidor a la petición del usuario. (Fuente: Elaboración propia)

Para finalizar el análisis de la comunicación cliente-servidor con este esquema de red, con el SNIFFER propietario, se tiene que una vez que se ha llevado a cabo la etapa de autenticación que el servidor requiere para mostrar la documentación pedida por el usuario, también se llevaran a cabo las conexiones TCP correspondientes antes de la solicitud GET y posteriormente de la solicitud el servidor responderá con el código de aceptación 200 OK, indicando que ha aceptado la solicitud para mostrar la documentación pedida por el usuario, la cual desde luego será la página web mostrada en el primer ejemplo de red con el HUB.

4.4.4 Red con SNIFFER y Servidor Propietario

En la figura 4.22 se tiene el esquema de red utilizado:

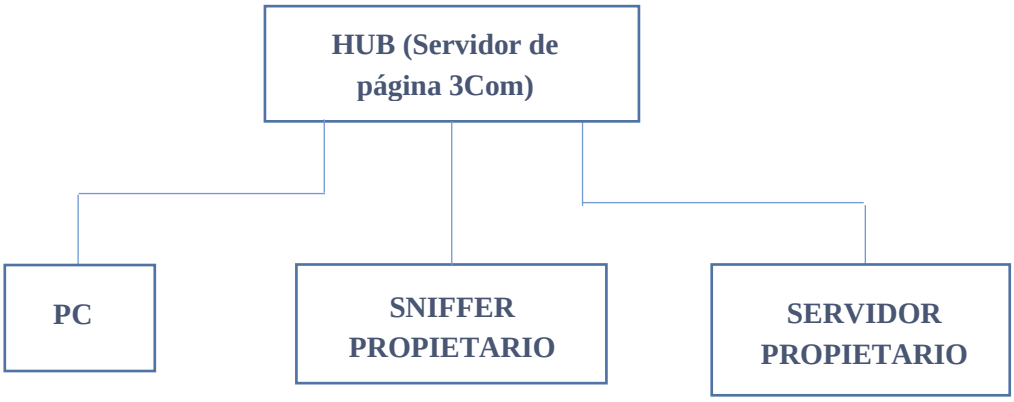


Fig. 4.22 Red con SNIFFER y servidor propietario. (Fuente: Elaboración propia)

En esta topología de red se agrega un servidor propietario, ya que simplemente se pretende crear un servidor de página con sistema operativo propietario, para eso la idea es clonar las tramas obtenidas con el servidor de página 3Com del HUB, y utilizar el SNIFFER propietario para comprobar y validar los resultados y las descripciones de tramas obtenidas. Esta es simplemente la topología con la que se trabajará de aquí en adelante.

5 Desarrollo del Servidor de Página

En esta última etapa del proyecto para el objetivo anteriormente planteado de crear un servidor de página en sistema operativo propietario clonando las tramas del servidor de página 3com del HUB, resultó que los archivos y la documentación del servidor 3com son mucho más complicadas de clonar de lo que se pensaba, dado su código fuente en lenguaje HTML, ya que posee muchas imágenes y otros recursos que hacen que el código HTML sea muy extenso y el sistema no está lo suficientemente desarrollado para poder asimilar todo ese código, con lo cual se utilizará un programa Hola Mundo montado en un servidor APACHE, que es el programa más básico para crear, ya que tiene un código fuente mucho más sencillo de describir. En la figura 5.1 se observa la página web de Hola Mundo:

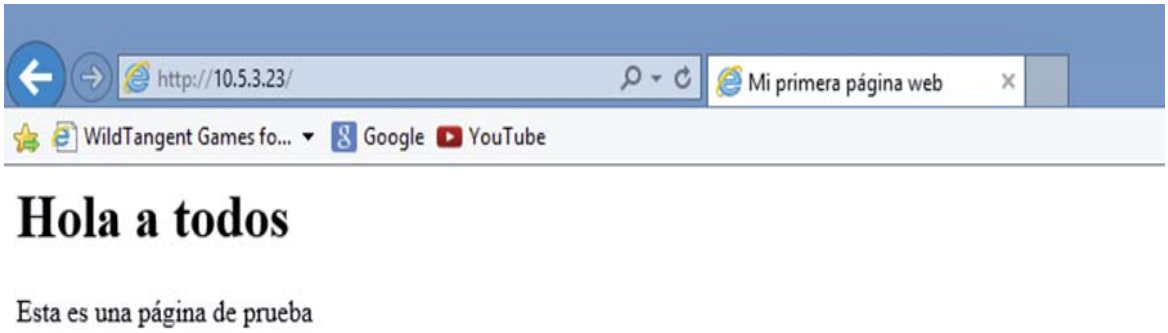


Fig. 5.1 Página Hola Mundo con servidor APACHE. (Fuente: dirección IP: 10.3.5.23)

Esta es la página Hola Mundo que se montó en un servidor APACHE, y la que se utilizará para crear el servidor de página en el sistema operativo propietario.

El correspondiente código fuente de la página, lenguaje HTML o Hipertexto, será el que se observa en la figura 5.2:

```
1  <html>
2    <head>
3      <meta  content="text/html; charset=UTF-8"  http-equiv="content-type">
4      <title>Mi primera página web</title>
5    </head>
6    <body>
7      <h1>Hola a todos</h1>
8      <p>Esta es una página de prueba</p>
9    </body>
10 </html>
11
```

Fig. 5.2 Código HTML de Hola Mundo en Apache. (Fuente: dirección IP: 10.3.5.23)

Con lo cual se tiene un código de HTML muy básico, y mucho más simple de manejar para el sistema.

De aquí en más al igual que en el capítulo anterior se trabajará con un determinado esquema de red para poder planificar y describir el proceso de transacciones HTTP de la comunicación cliente-servidor, y el contenido de dichas transacciones, para posteriormente validar todo eso con SNIFFER.

Entonces el esquema de trabajo utilizado en este caso será:

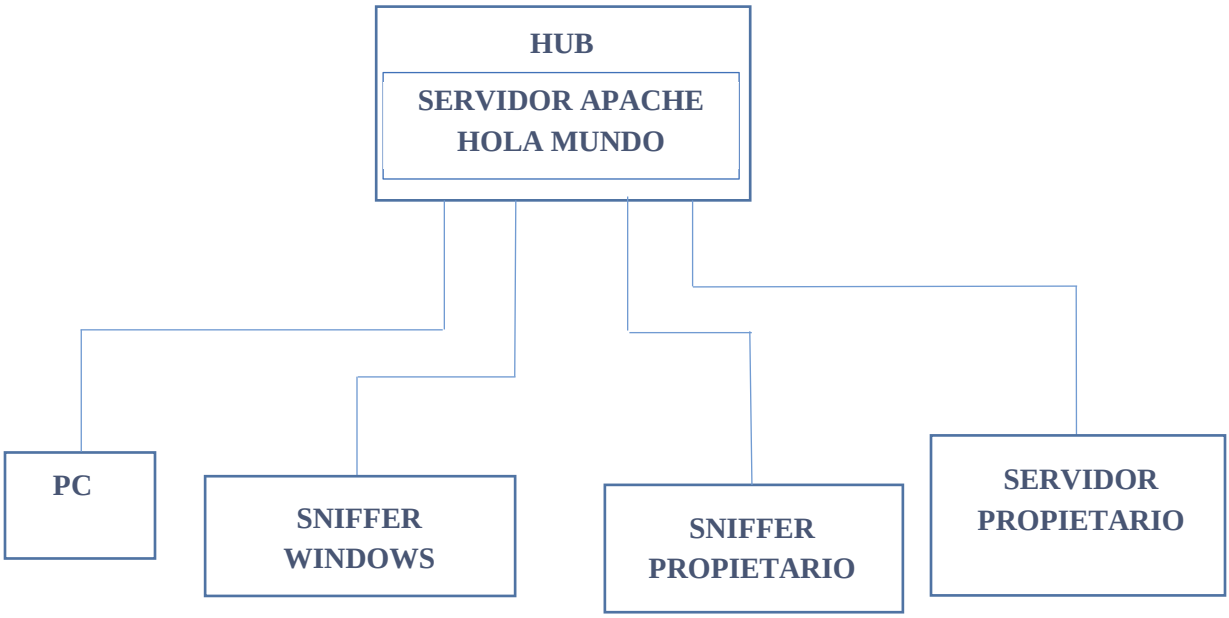
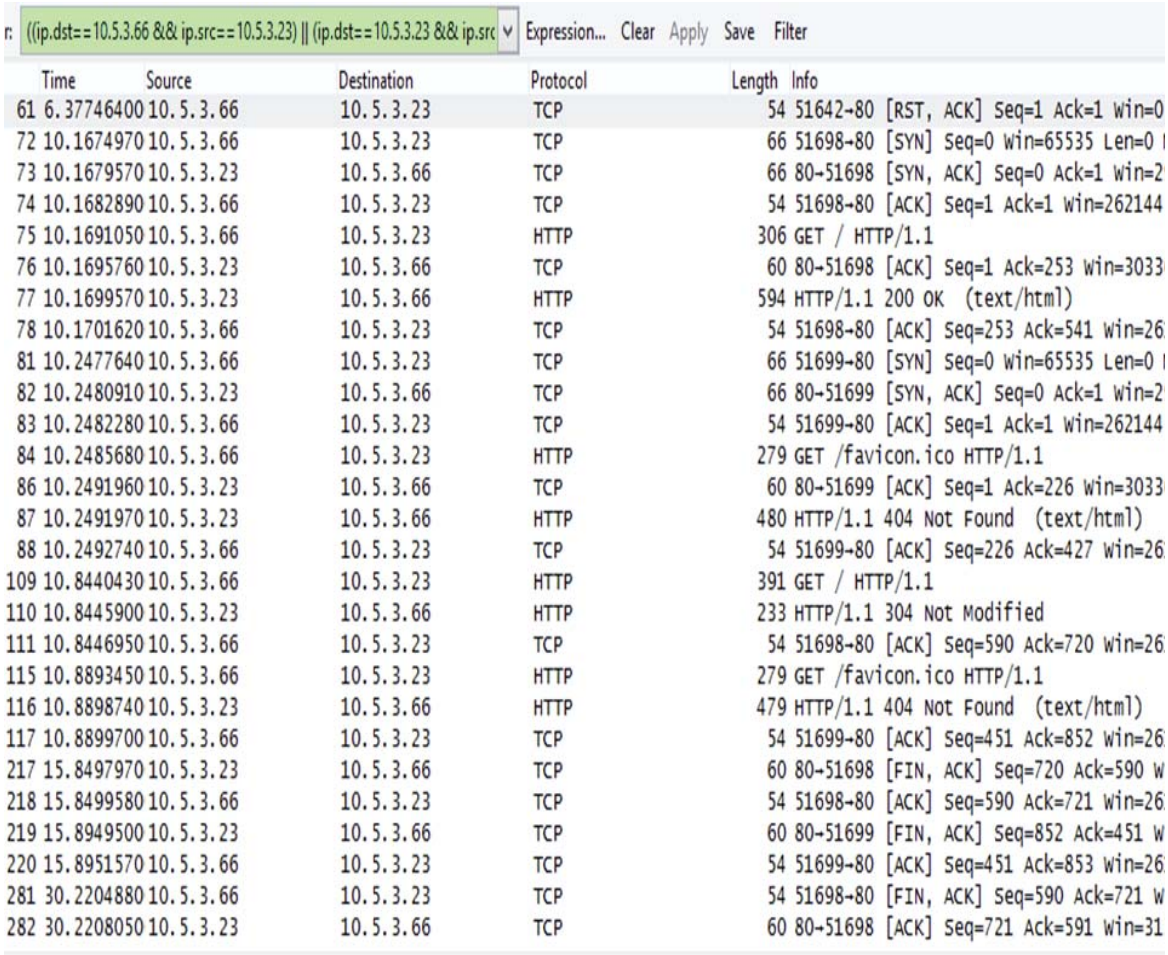


Fig. 5.3 Esquema de Red de Trabajo. (Fuente: Elaboración propia)

Con este nuevo esquema de red se tendrá a diferencia del capítulo anterior, un HUB con un servidor de página distinto al 3Com, un programa Hola Mundo montado en un servidor APACHE. Además de tener el servidor de página que se quiere crear en el sistema operativo propietario para poder clonar las tramas del Hola Mundo en APACHE, y los 2 SNIFFER para poder analizar claramente la comunicación cliente-servidor sin perder información importante.

Entonces utilizando SNIFFER (WIRESHARK) de WINDOWS para realizar la captura de tramas correspondiente del Hola Mundo en APACHE, se tiene la siguiente captura, mostrada en la figura 5.4:



	Time	Source	Destination	Protocol	Length	Info
61	6.37746400	10.5.3.66	10.5.3.23	TCP	54	51642→80 [RST, ACK] Seq=1 Ack=1 Win=0
72	10.1674970	10.5.3.66	10.5.3.23	TCP	66	51698→80 [SYN] Seq=0 Win=65535 Len=0
73	10.1679570	10.5.3.23	10.5.3.66	TCP	66	80→51698 [SYN, ACK] Seq=0 Ack=1 Win=2
74	10.1682890	10.5.3.66	10.5.3.23	TCP	54	51698→80 [ACK] Seq=1 Ack=1 Win=262144
75	10.1691050	10.5.3.66	10.5.3.23	HTTP	306	GET / HTTP/1.1
76	10.1695760	10.5.3.23	10.5.3.66	TCP	60	80→51698 [ACK] Seq=1 Ack=253 Win=3033
77	10.1699570	10.5.3.23	10.5.3.66	HTTP	594	HTTP/1.1 200 OK (text/html)
78	10.1701620	10.5.3.66	10.5.3.23	TCP	54	51698→80 [ACK] Seq=253 Ack=541 Win=26
81	10.2477640	10.5.3.66	10.5.3.23	TCP	66	51699→80 [SYN] Seq=0 Win=65535 Len=0
82	10.2480910	10.5.3.23	10.5.3.66	TCP	66	80→51699 [SYN, ACK] Seq=0 Ack=1 Win=2
83	10.2482280	10.5.3.66	10.5.3.23	TCP	54	51699→80 [ACK] Seq=1 Ack=1 Win=262144
84	10.2485680	10.5.3.66	10.5.3.23	HTTP	279	GET /favicon.ico HTTP/1.1
86	10.2491960	10.5.3.23	10.5.3.66	TCP	60	80→51699 [ACK] Seq=1 Ack=226 Win=3033
87	10.2491970	10.5.3.23	10.5.3.66	HTTP	480	HTTP/1.1 404 Not Found (text/html)
88	10.2492740	10.5.3.66	10.5.3.23	TCP	54	51699→80 [ACK] Seq=226 Ack=427 Win=26
109	10.8440430	10.5.3.66	10.5.3.23	HTTP	391	GET / HTTP/1.1
110	10.8445900	10.5.3.23	10.5.3.66	HTTP	233	HTTP/1.1 304 Not Modified
111	10.8446950	10.5.3.66	10.5.3.23	TCP	54	51698→80 [ACK] Seq=590 Ack=720 Win=26
115	10.8893450	10.5.3.66	10.5.3.23	HTTP	279	GET /favicon.ico HTTP/1.1
116	10.8898740	10.5.3.23	10.5.3.66	HTTP	479	HTTP/1.1 404 Not Found (text/html)
117	10.8899700	10.5.3.66	10.5.3.23	TCP	54	51699→80 [ACK] Seq=451 Ack=852 Win=26
217	15.8497970	10.5.3.23	10.5.3.66	TCP	60	80→51698 [FIN, ACK] Seq=720 Ack=590 W
218	15.8499580	10.5.3.66	10.5.3.23	TCP	54	51698→80 [ACK] Seq=590 Ack=721 Win=26
219	15.8949500	10.5.3.23	10.5.3.66	TCP	60	80→51699 [FIN, ACK] Seq=852 Ack=451 W
220	15.8951570	10.5.3.66	10.5.3.23	TCP	54	51699→80 [ACK] Seq=451 Ack=853 Win=26
281	30.2204880	10.5.3.66	10.5.3.23	TCP	54	51698→80 [FIN, ACK] Seq=590 Ack=721 W
282	30.2208050	10.5.3.23	10.5.3.66	TCP	60	80→51698 [ACK] Seq=721 Ack=591 Win=31

Fig. 5.4 Captura de transacciones HTTP para Hola Mundo en APACHE. (Fuente: (Sniffer (Wireshark))

Esta es la captura realizada, a la cual se aplicó un filtro que permite observar la captura de tramas TCP y HTTP vinculadas al Hola Mundo, el cual es de suma importancia, ya que para poder generar las correspondientes tramas HTTP en el servidor propietario primero hay que generar las correspondientes tramas TCP. En la captura realizada se observa que la IP (10.5.3.66) corresponde al cliente, mientras que la IP (10.5.3.23) es del servidor APACHE.

Para el objetivo de crear un servidor HTTP en sistema operativo propietario clonando las tramas del servidor APACHE, se tomarán en cuenta las primeras tramas de la captura, para crear un servidor lo más sencillo posible, ya que el sistema también tiene claras limitaciones. Por esto mismo es que se tomarán en cuenta las tramas que llegan hasta el primer código de respuesta, es decir hasta el primer 200 OK, pero la primera trama correspondiente a un segmento RST no se

tomará en cuenta ya que no es de interés, y las tramas que realmente importan son las de SYN y ACK, correspondientes a los segmentos TCP.

Explicado esto en un diagrama de secuencias como el que se observa en la figura 5.5, se tiene lo siguiente:

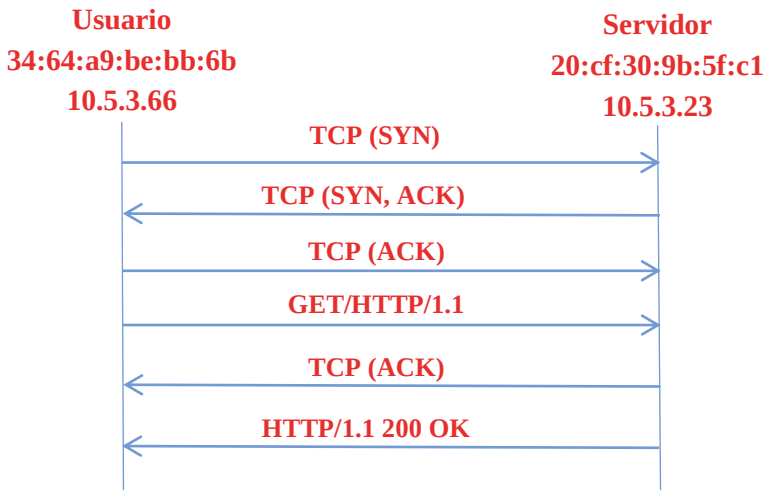


Fig. 5.5 Diagrama secuencial de comunicación para el servidor propietario. (Fuente: Elaboración propia)

Vale decir que este es el esquema de comunicación cliente-servidor que se pretende planificar y utilizar en este proyecto para crear el servidor HTTP en sistema operativo propietario.

Entonces para la creación del servidor, de aquí en adelante se explicará cómo opera el servidor y los códigos de programación utilizados en él, explicados en distintas etapas.

Entonces para la primera etapa de programación se tiene:

```
Program TCP_129;  {  EX PING_300 }
Const
    LF  = #$0A;
    { tamanno maximo de las tramas }
    max_size          = 2016;{ maximo tamanmo  de las tramas }
    SERVER_DHCP  = TRUE  ;
    SERVER_SIP   = TRUE  ;
    SERVER_HTTP  = TRUE  ;
    RTL_1 = $DC00;
    #INCLUDE  <RTL_8169.TXT>
    #INCLUDE  <TCP__CTE.TXT>
    #INCLUDE  <SIP__CTE.TXT>
```

```
{Configuraciones para DHCP}
 DHCP_Net_IP              = 192.168. 20.  0;
 DHCP_Brdcst_addr         = 192.168. 20.255;
 DHCP_SUBNET_MASK_        = 255.255.255.  0;
 DHCP_ROUTER_             = 192.168.20.  1;
 DHCP_DOMAIN_NAME_SERVER_ = 192.168.20 .1;
 DHCP_DOMAIN_NAME_        = 'Labproce';
 DHCP_HOST_NAME_          = 'Danux';
 DHCP_LEASE_TIME          = 43200; {Prestamo de IP: 12 horas}
 DHCP_REBINDING_TIME      = 36000; {Revinculacion : 10 horas}
 DHCP_RENEWAL_TIME        = 21600; {Renovacion    :  6 horas}

 {Tipos de mensaje ARP}
 ARP_REQUEST   = 1;
 ARP_REPLY     = 2;

 {Tipos de mensaje ICMP}
 ICMP_ECHO_REQUEST = $08;
 ICMP_ECHO_REPLY   = $00;

 {Flags TCP}
 TCP_SYN      = $02;
 TCP_ACK      = $10;
 TCP_FIN_ACK  = $11;
 TCP_SYN_ACK  = $12;
 TCP_PSH_ACK  = $18;
```

Esto representa la primera etapa del código de programación en el sistema propietario, en la cual se indica que tanto el servidor DHCP como el HTTP se encuentran habilitados, y entre las configuraciones para DHCP se tiene que la dirección IP que servirá para hacer contacto entre el usuario y el servidor es la 192.168.20.1, y la máscara de subred es la 255.255.255.0. Mientras que entre los segmentos TCP se indica para cada segmento un número de dos dígitos con signo $, el cual indica un valor en sistema hexadecimal que se le asigna a los segmentos TCP.

```
Produccion build_Header_Fields;
 Desde
     header_fields := '';

    {header_fields =+ 'Accept'; }                    {RFC 7231, sec 5.3.2}

    {header_fields =+ 'Accept-Charset'; }            {RFC 7231, sec 5.3.3}

    {header_fields =+ 'Accept-Encoding'; }           {RFC 7231, sec 5.3.4}

    {header_fields =+ 'Accept-Language'; }           {RFC 7231, sec 5.3.5}

     header_fields =+ 'Accept-Ranges: ';             {RFC 7233, sec 2.3}
     header_fields =+ 'bytes '+CRLF;

    {header_fields =+ 'Age'; }                       {RFC 7234, sec 5.1}

    {header_fields =+ 'Allow'; }                     {RFC 7231, sec 7.4.1}

    {header_fields =+ 'Authorization'; }             {RFC 7235, sec 4.2}
```

```
    {header_fields =+ 'Cache-Control'; }              {RFC 7234, sec 5.2}

    header_fields =+ 'Connection: ';                  {RFC 7230, SEC 6.1}
    header_fields =+ 'Keep-Alive '+CRLF;

    {header_fields =+ 'Content-Encoding'; }           {RFC 7231, sec 3.1.2.2}

    {header_fields =+ 'Content-Language'; }           {RFC 7231, sec 3.1.3.2}

    {header_fields =+ 'Content-Length'; }             {RFC 7230, sec 3.3.2}

    {header_fields =+ 'Content-Location'; }           {RFC 7231, sec 3.1.4.2}

    {header_fields =+ 'Content-Range'; }              {RFC 7233, sec 4.2}

     header_fields =+ 'Content-Type: ';               {RFC 7231, sec 3.1.1.5}
     header_fields =+ 'text/'; {type}
     header_fields =+ 'html '; {subtype}
     header_fields =+ 'charset=UTF-8 '+CRLF ; {parameter}

     header_fields =+ 'Date: ';                       {RFC 7231, sec 7.1.1.2}
     header_fields =+ 'Tue, ';{day-name}
     header_fields =+ '30 Jun 2016 '; {date-1}
     header_fields =+ '20:37:36 '; {time-of-day}
     header_fields =+ 'GMT '+ CRLF; {GMT}

     header_fields =+ 'ETag: ';                       {RFC 7232, sec 2.3}
     header_fields =+ '"84-5365dc1119a5b" '+CRLF ;

     {header_fields =+ 'Expect'; }                    {RFC 7231, sec 5.1.1}

     {header_fields =+ 'Expires'; }                   {RFC 7234, sec 5.3}

     {header_fields =+ 'From'; }                      {RFC 7231, sec 5.5.1}

     {header_fields =+ 'Host'; }                      {RFC 7230, sec 5.4}

     {header_fields =+ 'If-Match'; }                  {RFC 7232, sec 3.1}

     {header_fields =+ 'If-Modified-Since'; }         {RFC 7232, sec 3.3}

     {header_fields =+ 'If-None-Match'; }             {RFC 7232, sec 3.2}

     {header_fields =+ 'If-Range'; }                  {RFC 7233, sec 3.2}

     {header_fields =+ 'If-Unmodified-Since'; }       {RFC 7232, sec 3.4}

     header_fields =+ 'Last-Modified: ';              {RFC 7232, sec 2.2}
     header_fields =+ 'Tue '; {day-name}
     header_fields =+ '28 Jun 2016 '; {date-1}
     header_fields =+ '21:59:42 '; {time-of-day}
     header_fields =+ 'GMT '+CRLF; {GMT}

     {header_fields =+ 'Location'; }                  {RFC 7231, sec 7.1.2}

     {header_fields =+ 'Max-Forwards'; }              {RFC 7231, sec 5.1.2}

     {header_fields =+ 'Mime-Version'; }              {RFC 7231, app A.1}

     {header_fields =+ 'Pragma'; }                    {RFC 7234, sec 5.4}

     {header_fields =+ 'Proxy-Authenticate'; }        {RFC 7235, sec 4.3}
```

```
     {header_fields =+ 'Proxy-Authorization'; }      {RFC 7235, sec 4.4}

     {header_fields =+ 'Range'; }                    {RFC 7233, sec 3.1}

     {header_fields =+ 'Referer'; }                  {RFC 7231, sec 5.5.2}

     {header_fields =+ 'Retry-After'; }              {RFC 7231, sec 7.1.3}

     header_fields =+ 'Server: ';                    {RFC 7231, sec 7.4.2}
     header_fields =+ 'Danux '+CRLF ;

     {header_fields =+ 'TE'; }                       {RFC 7230, sec 4.3}

     {header_fields =+ 'Trailer'; }                  {RFC 7230, sec 4.4}

     {header_fields =+ 'Transfer-Encoding'; }        {RFC 7230, sec 3.3.1}

     {header_fields =+ 'Upgrade'; }                  {RFC 7230, sec 6.7}

     {header_fields =+ 'User-Agent'; }               {RFC 7231, sec 5.5.3}

     {header_fields =+ 'Vary'; }                     {RFC 7231, sec 7.1.4}

     {header_fields =+ 'Via'; }                      {RFC 7230, sec 5.7.1}

     {header_fields =+ 'Warning'; }                  {RFC 7234, sec 5.5}

     {header_fields =+ 'WWW-Authenticate'; }         {RFC 7235, sec 4.1}

 Hasta;
```

En esta segunda etapa se construyen los distintos encabezados involucrados en la creación del servidor de página en base a herramientas de metalenguaje ABNF para HTTP/1.1 descritas en la normativa RFC 7230 hasta la 7235, lo cual se encuentra en el apéndice B de este informe. No obstante sólo se han tomado en cuenta algunos encabezados para la creación de las tramas, ya que son los principales y los que hacen referencia para la primera trama de respuesta HTTP. En el lado derecho de cada encabezado se tiene entre paréntesis de llaves simplemente un comentario al respecto señalando la normativa y sección en que aparece la documentación correspondiente a dicho encabezado.

```
Produccion Resp_a_solicitud_de_recurso;
Var   iden    : word;

   pos_get     : Longint;
   largo_trama : Longint;
   delta       : Longint ;
   long_1      : Byte ;
   long_2      : Byte ;
   long_3      : Byte ;
   long_4      : Byte ;
   long_aux    : Longint absolute long_1;
   byte_aux    : Byte ;
   largo_str_http: longint;
   html_size_str: string;
```

```
Desde
        html_doc:='<!DOCTYPE html>'+LF +'<html>'+ LF+'  '+
                   '<head>'+LF+ '  </head>'+LF+ '  <body>'+LF +
                  '    <h1>Hola a Todos</h1>' + LF+
                              '    <p>esta es una pagina de prueba<p>' + LF +
                              '  </body>' + LF + '</html>' + LF;
{documento html, la "pagina"}
        html_size_str:=#(html_doc_size-4);
        writeln('tamano documento :  ',html_size_str);
        { define trama de respuesta a get ack}
        Fill (mem [@trama_o ],max_size,$00);
        Fill (mem [@trama_o2],max_size,$FF);

       ETh (trama_o | des_mac, trama_in[i] | sou_mac);
       Eth (trama_o | sou_mac  , My_mac);
       Eth (trama_o | tipo  , PROTOCOLO_IP);
       Eth (trama_o | version  , $45);
       Eth (trama_o | servicio , trama_in[i] | servicio);
       iden:= trama_in[i] | identif + $0100;
       Eth (trama_o | identif  ,iden);
       Eth (trama_o | flags  , $0040);
       Eth (trama_o | TTL   , trama_in[i] | TTL);
       Eth (trama_o | protocolo, PROTOCOLO_TCP);
       Eth (trama_o | SOU_IP   , my_ip);
       Eth (trama_o | DES_IP   , trama_in[i] | SOU_IP);
       Eth (trama_o | SOU_PORT , PUERTO_HTTP);
       Eth (trama_o | DES_PORT , trama_in[i] | SOU_PORT);
       Eth (trama_o | TCP_SEQ_NUM, trama_in[i] | TCP_ACK_NUM);

        largo_trama := trama_in[i] | frame_length ;
        pos_get   := p1_longint-@trama_in[i];
        pos_get   := pos_get-4;
        delta    := largo_trama- pos_get ;
        writeln($delta);

         long_aux  := trama_in [i] | TCP_SEQ_NUM;
         {writeln ($long_aux,' ',$long_low,' ',$long_hi );}

         byte_aux := long_1;
         long_1   := long_4;
         long_4   := byte_aux;
         byte_aux := long_2;
         long_2   := long_3;
         long_3   := byte_aux;

         long_aux := long_aux+largo_trama - pos_get;
         byte_aux := long_1;
         long_1   := long_4;
         long_4   := byte_aux ;
         byte_aux := long_2;
         long_2   := long_3;
         long_3   := byte_aux;

       Eth (trama_o | TCP_ACK_NUM, long_aux);

       Eth (trama_o | TCP_DATA_OFFSET , trama_in[i] | TCP_DATA_OFFSET);

       Eth (trama_o | TCP_flags_sync , TCP_ACK);
       Eth (trama_o | TCP_WIN_SIZE , trama_in[i] | TCP_WIN_SIZE);
       Eth (trama_o | TCP_URG_PTR  , $0000);

       enviar_paquete_tcp; { ok }
```

```
Fill(mem[@trama_o],max_size,$00);
Fill(mem[@trama_o2],max_size,$FF);
Eth (trama_o | des_mac , trama_in[i] | sou_mac);
Eth (trama_o | sou_mac , My_mac);
 Eth (trama_o | tipo     , PROTOCOLO_IP);
 Eth (trama_o | version  , $45);
 Eth (trama_o | servicio , trama_in[i] | servicio);
 iden:= iden + $0100;
 Eth (trama_o | identif , iden);

Eth ( trama_o | flags   , $0040);
Eth (trama_o | TTL     , trama_in[i] | TTL);
Eth (trama_o | protocolo , PROTOCOLO_TCP);
Eth (trama_o | SOU_IP  , my_ip);
Eth (trama_o | DES_IP   , trama_in[i] | SOU_IP);
Eth (trama_o | SOU_PORT, PUERTO_HTTP);
Eth (trama_o | DES_PORT , trama_in[i] | SOU_PORT);
Eth (trama_o | TCP_SEQ_NUM, trama_in[i] | TCP_ACK_NUM);

 largo_trama := trama_in[i] | frame_length ;
 pos_get   := p1_longint-@trama_in[i];
 pos_get   := pos_get-4;
  delta    := largo_trama- pos_get ;

  long_aux  := trama_in [i] | TCP_SEQ_NUM;
  {writeln ($long_aux,' ',$long_low,' ',$long_hi );}

  byte_aux := long_1;
  long_1   := long_4;
  long_4   := byte_aux;
  byte_aux := long_2;
  long_2   := long_3;
  long_3   := byte_aux;

  long_aux := long_aux+largo_trama - pos_get;
  byte_aux := long_1;
  long_1   := long_4;
  long_4   := byte_aux ;
  byte_aux := long_2;
  long_2   := long_3;
  long_3   := byte_aux;

 Eth (trama_o | TCP_ACK_NUM, long_aux);
 Eth (trama_o | TCP_DATA_OFFSET   , trama_in[i] | TCP_DATA_OFFSET);
 Eth (trama_o | TCP_flags_sync    , TCP_ACK);
 Eth (trama_o | TCP_WIN_SIZE      , trama_in[i] | TCP_WIN_SIZE);
 Eth (trama_o | TCP_URG_PTR       , $0000);

 str_http:= 'HTTP/1.1 200 OK'+ CRLF  +
            'Date: Tue, 30 Jun 2016 20:37:36 GMT' + CRLF +
            'Server: Danux TCP_129 (Jaiba) ' + CRLF +
            'Last-Modified: Tue, 28 Jun 2016 21:59:42 GMT'+CRLF+
            'ETag: "84-5365dc1119a5b"' + CRLF +
            'Accept-Ranges: bytes' +CRLF +
            'Content-Length: '+html_size_str + CRLF +
            'Keep-Alive: timeout=5, max=100' + CRLF +
            'Connection: Keep-Alive'+ CRLF+
            'Content-Type: text/html; charset=UTF-8'+CRLF+
             CRLF+html_doc ;


largo_str_http  := trama_o | http_largo;
```

```
      trama_o  | http_str := str_http;
      trama_o2 | http_str := str_http;
      writeln (trama_o|http_largo) ;
      trama_o  | http_largo := largo_str_http;
      trama_o2 | http_largo := largo_str_http;
      enviar_paquete_tcp;

Hasta;
```

En esta producción se pretende generar las correspondientes tramas de respuesta a las distintas peticiones realizadas por el usuario, y es así como se generan las tramas de respuesta a la petición GET, es decir, se generó el TCP (ACK) de respuesta, para luego dar paso a la trama de respuesta HTTP 200 OK con sus correspondientes encabezados.

En esta producción se indica el formato en que están determinadas las variables usadas en la producción, como el caso de Longint que representa una cifra de 32 byte, el byte representa 8 bit, y el string una cadena de caracteres entre los cuales se permiten palabras. Además se muestra al final de la captura el lenguaje de hipertexto sacado de la página Hola Mundo creada con servidor Apache.

En la etapa posterior al hipertexto, se tiene primero el comando fill, el cual permite que en las tramas generadas, primero se generen dos tramas de respuesta, trama O y trama O2, esto porque necesita calcularse el tamaño de las tramas, y para ello se crean dos tramas, una rellenada con sólo 00 y otra con FF, y max size representa la cantidad máxima de bytes que soporta la tarjeta de red, que en este caso son 2016 bytes. Una vez que se tienen las tramas O y O2 rellenadas con 00 y FF, comienzan a completarse los campos presentes en cualquier trama de respuesta 200 0K, indicando que los datos son enviados del servidor al cliente, y ante lo cual se utiliza un comando ETH ya que esto permite no tener que escribir cada campo para ambas tramas, basta con escribir para una, y con el comando ETH se llenarán los datos para ambas tramas, y una vez que se completan todos los campos del ETH para determinar el tamaño de la trama se hace una comparación entre las tramas con los datos que fueron completados en las tramas del 00 y FF, realizando un barrido de datos desde el byte más alto hasta el más bajo, y aquél valor en que coincidieran pasaría a ser el tamaño de la trama. No obstante para generar la trama de respuesta se necesitan muchos más datos para completar, como es el caso posterior a eso perteneciente a unos datos de posición de determinados bytes, el de pos_get indica que el GET está escrito de la siguiente manera ´GET ´, el espacio antes de cerrar comillas es de la posición 4, la cual indica que se corre cuatro espacios desde el origen, por eso aparece la comilla de cierre corrida un espacio. Después se tendrán más llenados de paquetes con ETH, y la etapa de enviar paquete TCP hace que calcule el tamaño de las tramas, y con eso se podrá comenzar a crear la trama de respuesta 200 OK.

Para finalizar la producción se tendrán los campos de la trama 200 OK, indicando entre otras cosas que pertenece al servidor propietario, y al final se observan algunos datos de string para ambas tramas O y O2, pero que cuando se ocupa string no se puede ocupar comando ETH, para posteriormente enviar paquete TCP, y concluye la producción de respuesta a solicitud de recurso.

Con esto se da paso a la última etapa de programación en el sistema propietario para crear el servidor HTTP, para lo cual se tiene el siguiente código:

```
Produccion HTTP;
Var
uri        : string;
host       : string;
largo      : longint absolute host;
seq_number : longint;
iden       : word;
opciones   : parrafo [320];

 Desde
   Desde
       ?trama_in[i] | des_port =puerto_http;{80}
       trama_in[i] | http_opc ? 'GET ' p1;      {RFC 7230, sec 4.3.1 }

       p1 | http_opc_par ? ' ' p2;

      uri := string [p1,p2];
       {writeln(uri);}   { uri tiene el recurso solicitado   }
       resp_a_solicitud_de_recurso;

    {   p2 | http_opc_par ? 'Host: ' p1;}
    {   p1 | http_opc ? #$0D  p2 ;}
     {  host := string [p1,p2]; }
    {   largo := largo - 1;  }
 {      writeln ('  ',host, uri);}

    Sino
       ?trama_in[i] | tcp_flags_sync  = tcp_syn ;
{        writeln ('sync ok ');}
        Fill(Mem[@trama_o ],max_size,$00);
        Fill(Mem[@trama_o2],max_size,$FF);
        Eth (trama_o | des_mac, trama_in[i] | sou_mac);
        Eth (trama_o | sou_mac, My_mac);
        Eth (trama_o | tipo, PROTOCOLO_IP);
        Eth (trama_o | version, $45);
        Eth (trama_o | servicio, trama_in[i] | servicio);
        {trama_o | largo := }

        iden := trama_in [i] | identif +$0100;
        Eth (trama_o | identif, iden);
        Eth (trama_o | Flags, $0040);
        Eth (trama_o | TTL, trama_in[i] | TTL);
        Eth (trama_o | protocolo, PROTOCOLO_TCP);
         {trama_o | cksum_ip;}
        Eth (trama_o | SOU_IP, my_ip);
        Eth (trama_o | DES_IP, trama_in[i] | SOU_IP);
        Eth (trama_o | SOU_PORT, PUERTO_HTTP);
        Eth (trama_o | DES_PORT, trama_in[i] | SOU_PORT);

        Eth (trama_o | TCP_SEQ_NUM, $00080000);
        seq_number:= trama_in [i] | TCP_SEQ_NUM +$1000000;
        Eth (trama_o | TCP_ACK_NUM, seq_number);

        Eth (trama_o | TCP_DATA_OFFSET, trama_in[i] | TCP_DATA_OFFSET);
        Eth (trama_o | TCP_flags_sync, TCP_SYN_ACK);
        Eth (trama_o | TCP_WIN_SIZE, trama_in[i] | TCP_WIN_SIZE);

        Eth (trama_o | TCP_URG_PTR, $0000);
```

```
        opciones :=#$02+#$04+#$05+#$B4+#$01+#$01+#$04+#$02+
                                  #$01+#$03+#$03+#$07;
        ETH (trama_o [$37],$02);
        ETH (trama_o [$38],$04);
        ETH (trama_o [$39],$05);
        ETH (trama_o [$3A],$B4);
        ETH (trama_o [$3B],$01);
        ETH (trama_o [$3C],$01);
        ETH (trama_o [$3D],$04);
        ETH (trama_o [$3E],$02);
        ETH (trama_o [$3F],$01);
        ETH (trama_o [$40],$03);
        ETH (trama_o [$41],$03);
        ETH (trama_o [$42],$07);

       { writeln ($trama_o|frame_length,'in');}
        enviar_paquete_TCP;
       { writeln ($trama_o|frame_length,'out');}

  sino
      trama_in[i] | http_opc ? 'HEAD ' p1;      {RFC 7230, sec 4.3.2}

  sino
      trama_in[i] | http_opc ? 'POST ' p1;      {RFC 7230, sec 4.3.3}
  sino
      trama_in[i] | http_opc ? 'PUT ' p1;       {RFC 7230, sec 4.3.4}
  sino
      trama_in[i] | http_opc ? 'DELETE ' p1;    {RFC 7230, sec 4.3.5}
  sino
      trama_in[i] | http_opc ? 'CONNECT ' p1;   {RFC 7230, sec 4.3.6}
  sino
      trama_in[i] | http_opc ? 'OPTIONS ' p1;   {RFC 7230, sec 4.3.7}
  sino
      trama_in[i] | http_opc ? 'TRACE ' p1;     {RFC 7230, sec 4.3.8}

  Hasta;
 Hasta;
```

Con esta producción HTTP se genera la respuesta SYN+ACK si la trama de entrada no es el GET, en caso contrario se genera la producción de respuesta a solicitud de recurso.

Para comenzar, se tiene que el servidor pregunta si es que la trama de entrada pertenece al método de petición GET, si es correcto el servidor genera la producción de respuesta a solicitud de recurso descrita anteriormente, mientras que si es falso, se tendrá que el servidor pasará a preguntar si es que la trama de entrada es un TCP (SYN), y si es correcto el servidor generará la trama de respuesta TCP (SYN, ACK), mientras que los campos que se llenaron en la trama son los mismos que en la producción de respuesta a solicitud de recurso. No obstante al final de la etapa de generar la trama TCP (SYN, ACK), tenemos otros campos que también se utilizan como es el caso de cifras en cardinalidad, las cuales representan cantidades en hexadecimal de opciones que posee la trama TCP (SYN, ACK), y las cantidades en ETH representan valores en un arreglo, ya que la trama se compone de arreglos.

Con la producción HTTP se vió cómo generar tramas de respuesta ante las tramas de petición GET y TCP (SYN), pero dada la sintaxis del protocolo HTTP, hay más métodos de petición a los

cuales responde el servidor, por lo tanto, si la trama de entrada al servidor no es GET ni TCP (SYN), se programó para que el servidor pregunte por los demás métodos de petición HTTP. En ese caso si el servidor no reconoce las tramas de entrada GET ni TCP (SYN), comenzará preguntar por los demás métodos de petición HTTP, pero no obstante al no haber ningún campo de respuesta en esos métodos el servidor simplemente no generará ninguna trama de respuesta. Las siglas en los paréntesis de llaves son un simple comentario, indicando que están todos los métodos descritos en la normativa RFC 7230, y cada uno con su sección.

Una vez creado el servidor con todos los códigos de programación correspondiente, corresponde llegar a la etapa de validación, es decir, que se cumpla lo esperado, ante lo cual en la figura 5.6 se tiene lo siguiente:

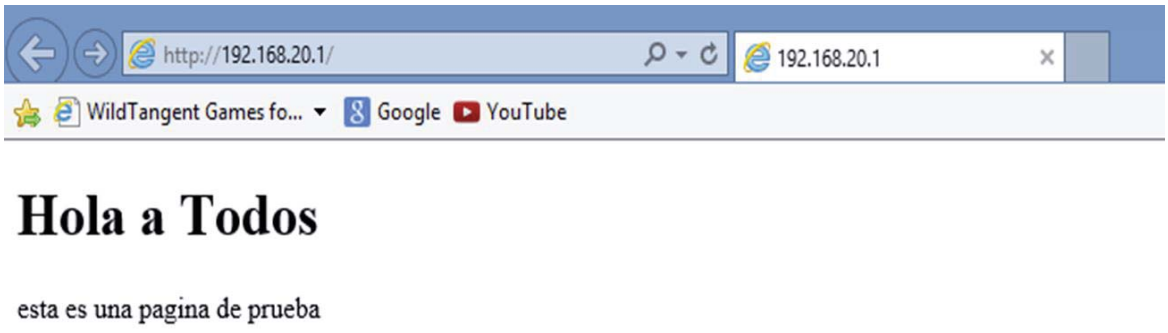


Fig. 5.6 Hola Mundo con servidor HTTP en sistema propietario. (Fuente: Sniffer (Wireshark))

Con la última captura se tiene que el usuario entró en contacto con el servidor ya que se mostró la página esperada, siendo la IP (192.168.20.1), la dirección IP con la que se hizo contacto con el servidor.

Ahora utilizando SNIFFER de WINDOWS se capturan las tramas correspondientes, esperando que cumpla con lo establecido en el sistema propietario. Entonces se tendrá la siguiente captura mostrada en la figura 5.7:

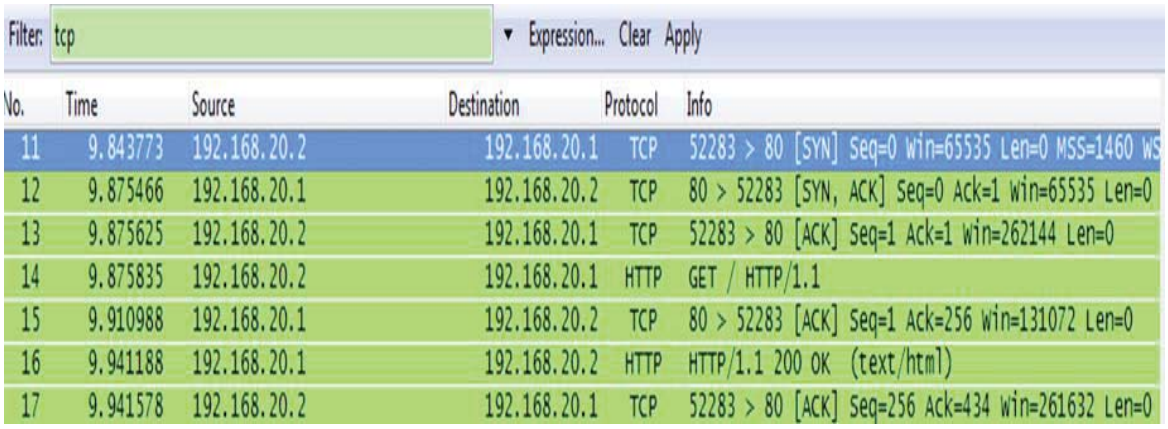


Fig. 5.7 Captura de transacciones de Hola Mundo en sistema propietario. (Fuente: Sniffer (Wireshark))

Con esta captura se tiene que la IP del usuario es 192.168.20.2, lo cual confirma que fue asignada por el servidor propietario, y además se observan las correspondientes tramas TCP de respuesta construida, además de la trama de respuesta HTTP 200 OK. Ante la primera trama de petición ACK, el servidor responde con la trama SYN+ACK, generada en la producción HTTP, mientras que ante la petición GET, se observa la trama de respuesta ACK, para luego dar paso a la trama de respuesta 200 OK. Con esto se verifica que se cumple la etapa de comunicación cliente-servidor que se pretendía establecer al comienzo en el diagrama de secuencias.

Con el SNIFFER de WINDOWS además de capturar el tráfico de red, se puede capturar el contenido de cada transacción, es decir, la trama HTTP correspondiente para analizar que se cumpla lo establecido en la programación del sistema operativo propietario, y se tiene la siguiente captura:

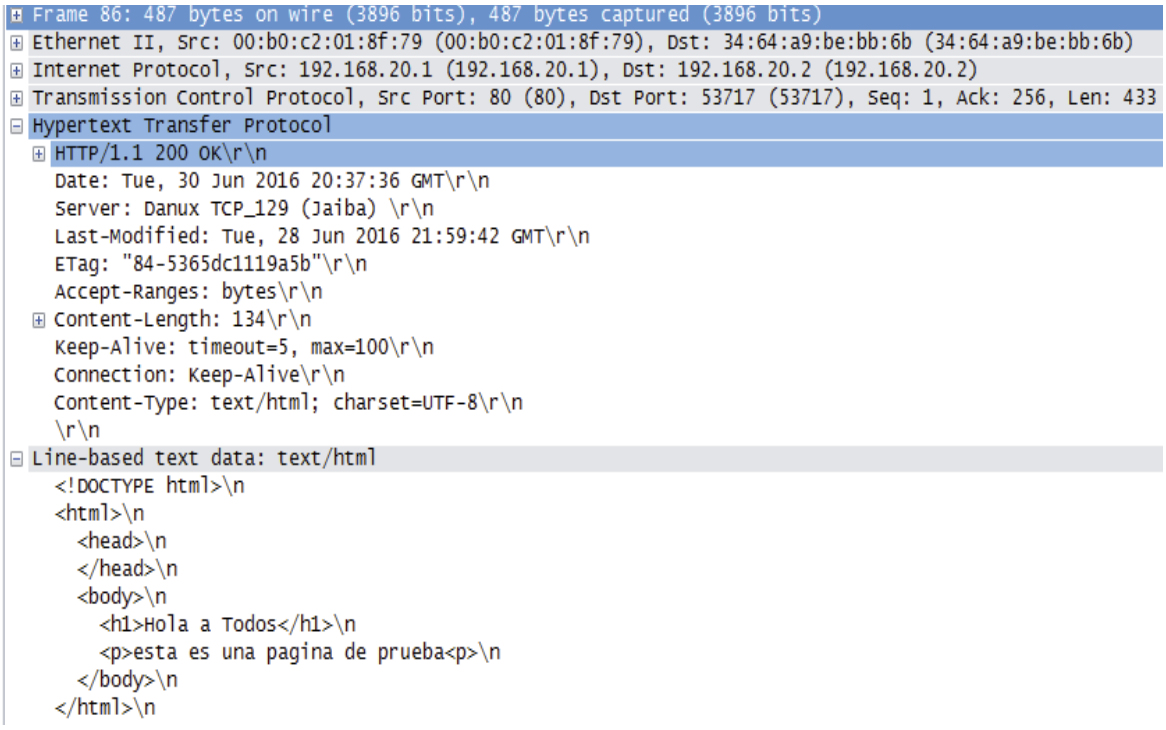


Fig. 5.8 Captura de trama de respuesta HTTP 200 OK creada con sistema operativo propietario. (Fuente: Sniffer Wireshark))

Ante la captura es posible observar que el contenido de la trama, el código de respuesta y los encabezados concuerda perfectamente con lo establecido en el sistema, tanto en la producción de respuesta a solicitud de recurso, como en la de construcción de los campos de cabecera. Además fue posible observar el contenido del cuerpo de la respuesta que coincide perfectamente con el código HTML generado para la página Hola Mundo.

6 Análisis Económico

Para el estudio de la factibilidad económica de este proyecto, es decir, que tan viable es realizar este proyecto en materia económica para poder prestarle el servicio a una empresa, se debe tomar en consideración determinados costos asociados al proyecto, para ello se debe tener en cuenta el equipamiento utilizado para la realización del proyecto, y para ello se necesita:

- Un servidor.
- Una pantalla (o monitor).
- Un teclado.
- Un medio de conexión a ISP (enlace dedicado).

Al tener esos cuatro elementos se tiene el equipamiento necesario para la realización del proyecto. Se utilizará el servidor para programar y configurar el servidor web o servidor de página que se pretende crear (servidor HTTP), mientras que con el enlace dedicado se conecta el servidor a Internet, en tanto con el teclado se manipula la información que se quiere programar en el servidor, y con la pantalla se visualiza el contenido de la programación.

6.1 Análisis de Equipos

6.1.1 Servidor

Para el caso del servidor cotizando en distintas empresas de ventas y servicios, es posible encontrar una serie de ofertas, ante lo cual se tiene la siguiente alternativa:

Fig. 6.1 Server PowerEdge R320 Intel® Xeon™ E5-2420v2 1.9 GHz, 8GB RAM, 2x1TB SATA 3.5" hot plug HD

Dell PowerEdge R320 es un servidor de clase empresarial diseñado para manejar altas cargas de trabajos. Este servidor incluye 1 socket y 1U para su montaje en rack una gran capacidad de memoria y ocho unidades de disco duro. La relación costo /rendimiento, junto con la flexibilidad de configuración y de soporte, hacen de este servidor un excelente aliado para empresas y organizaciones. [12]

Entre sus características se encuentra:

- Marca: Dell
- Factor de forma: 1U Rack
- Modelo: PowerEdge R320
- Procesador: Intel® Xeon® E5-2420v2 1.90GHz, 15M Cache, 7.2GT/s QPI, Turbo Máx. 2.4 GHz, 6 nucleos, 95W, frecuencia máxima: 1333MHz
- Memoria: 8GB RDIMM, 1600MT/s
  Máximo 192GB (6 DIMM slots)
- Disco duro: 2 x 1TB 7.2K RPM SATA 3Gbps 3.5" Hot Plug HD
  Máximo 16TB; bahías: máximo ocho 2.5" hot-plug SAS, SATA o SSD ; o cuatro 3.5" SAS, SATA
- Unidad óptica: DVD+/-RW, SATA
- Sistema Operativo: No incluye
- Fuente de alimentación: dual, redundante hot plug, 350W
- Controlador interno RAID: controlador interno RAID PERC H310, RAID 1
  PERC H710 HBA externos (RAID): PERC H810 HBA externos (no RAID): HBA SAS de 6Gbps
- Ethernet: en placa broadcom 5720. Dos puertos de 1Gb LOM
- PCI SLOTS (I/O):

1 x PCIe x8(x4) HH, HL (Gen 2)
1 x PCIe x16, FH, HL (Gen 3)

- Puertos USB: USB 2.0 (2 frontales / 2 traseros / 1 interno)
- Sistemas operativos compatibles: Microsoft® Windows Server® 2012 , Microsoft Windows Server 2012 de Essentials, Microsoft Windows Server 2008 R2 SP1, x 64 (incluye Hyper-V ®), Novell® SUSE® Linux Enterprise Server, Red Hat Enterprise Linux®
- Garantía: 1 año de garantía basic hardware
- Incluye ReadyRailsTM II: carriles de deslizamiento para el montaje sin herramientas en 4 postes bastidores con orificios redondos o cuadrados sin rosca o labrado de montaje de 4 postes roscados bastidores agujero, con soporte para el brazo opcional de gestión de cables sin herramientas.
- No inlcuye Sistema Operativo, CAL de usuario ni CAL RDS.

Precio Oferta Efectivo : $ 1.599.990

Precio Oferta * $ 1.684.190 ((*) Otros Medios de Pago)

Precio Referencial $ 1.999.990

6.1.2 Teclado

Mientras que para el teclado se tiene la siguiente oferta:

Fig. 6.2 Teclado USB KB-125 Negro

Entrega al computador una apariencia más sofisticada con el KB-125 y su clásico acabado, siguiendo las últimas tendencias en periféricos. Durable, compacto y liviano. Un teclado con diseño especial para un computador. [12]

Entre sus características se tiene:

- Modelo: KB-125
- Color: Negro
- Conexión: Conexión USB
- Tipo de Teclas: Concavas
- Modos de hibernación y reinicio One Touch
- Teclado de alta durabilidad, compacto y ligero para computadores de escritorio
- Presenta dos bases en la carcaza que sirven como soporte ergonómico para las muñecas proporcionando una postura adecuada para teclear
- Compatible con: Windows 8, 10 o posterior
- Dimensiones: 452 x 162 x 23 mm
- Peso: 540 gr.

Precio Oferta Efectivo : $ 5.690

Precio Oferta *$ 5.990 ((*) Otros Medios de Pago)

Precio Referencial: $ 6.990

6.1.3 Pantalla

Para la pantalla, o monitor, se tiene:

Fig. 6.3 Monitor LED 19" E970SWM

El Monitor con retroiluminación LED E970SWM de AOC es un monitor LED de 18,5" con resolución HD. Su pantalla de formato 16:9 te permite ver imágenes de alta definición y su tamaño compacto permite tenerlo cómodamente en tu escritorio sin ocupar mucho espacio. Su

contraste dinámico (DCR) de 20.000.000:1 permite a los usuarios ver las zonas más oscuras de contenido con mayor profundidad. [12]

Entre sus características se encuentran:

- Marca: AOC
- Modelo: E970SWM
- Color: Negro
- Pantalla: 18.5”
- Resolución: HD 1366 x 768
- Tiempo de respuesta: 5 ms
- Brillo de pantalla: 200 cd / m²
- Relación de contraste (dinámico): 20.000.000:1
- Razón de contraste (típica): 700:1
- Relación de aspecto: 16:9
- Ángulo de visió: 90°/65°
- Número de colores de la pantalla: 16,7M
- Tamaño de pixel: 0,3 x 0,3 mm
- Intervalo de escaneado horizontal: 30 - 60 kHz
- Intervalo de escaneado vertical: 50 - 76 Hz
- Dispositivos de Entrada y Salida: 1 x VGA
- Consumo de energía (inactivo): 0,5 W
- Consumo energético: 15 W
- Consumo de energía (apagado): 0,5 W
- Compatibilidad VESA: 100x100
- Seguridad: compatible con bloqueo Kensington
- Dimensiones (WxHxD): 437.4 x 271.6 x 47.7 mm
- Peso: 2.15 kg

Precio Efectivo: $ 49.990

Precio Normal *$ 52.590 ((*) Otros Medios de Pago)

6.1.4 Punto de Red (Enlace Dedicado)

Por último para tener un medio de conexión a Internet, se necesitará de un medio de conexión a ISP (Proveedor de servicios de Internet), un punto de red o enlace dedicado, el cual consistirá de algún medio tecnológico para que el usuario se pueda conectar a Internet. [13]

En Entel se tiene lo siguiente:

Pack Negocio: Telefonía fija ilimitada e internet fibra óptica de alta velocidad para tu negocio.

Pack Negocio es una solución de conectividad, recomendada tanto para las micro como pequeñas empresas que requieran de una o dos líneas de telefonía fija e independientes, con planes ilimitados de minutos a teléfonos fijos y paquetes de minutos para llamar a celular, más una conexión a internet de alta velocidad, con todos los beneficios que entrega la red de fibra óptica de Entel.

Además, contratando Pack Negocio puedes mantener tu número telefónico y así no perderás contacto con tus clientes o proveedores.

Entonces entre las distintas ofertas que se tienen se ha elegido lo siguiente:

Un pack de 1 línea telefónica, 1 equipo telefónico con 100 minutos e internet con 20 Mbps de velocidad de bajada, lo cual involucra los siguientes costos:

Servicios	Valor	Descuento	Total
Telefonía	0.41 UF	0.22 UF	0.19 UF
Internet	1.13 UF	0.39 UF	0.74 UF
Infraestructura	0.15 UF	0.15 UF	0.00 UF
Instalación interior	0.02 UF	0.02 UF	0.00 UF
Equipamiento	0.03 UF	0.03 UF	0.00 UF
Extensor Wifi	0.12 UF	0.00 UF	0.12 UF
TOTAL	**1.86 UF**	**0.81 UF**	**1.05 F**

Tabla 6-1: Valores de Pack de negocio en Entel

* Valores no incluyen IVA.

6.2 Análisis de Costos

Entonces para poder realizar este proyecto en materia económica prestándole el servicio a una empresa, se tomarán en cuenta los costos iniciales para la adquisición del equipamiento, y resulta en un valor de $1.747.968 pagando en efectivo.

Ahora bien para realizar un análisis económico se partirá de una situación en la cual se financie el proyecto con un préstamo bancario de $1.750.000, para un período de 1 año, con una tasa de interés del 7%. Esto permitirá pagar inicialmente todo lo necesario para realizar el proyecto, y al cabo de 1 año se pretende determinar si el proyecto produce ganancias, pérdidas o indiferencia, entonces una vez comprados los equipos e instalado el servidor HTTP propietario, se prestará el servicio a empresas o clientes individuales y se les cobrará un precio para instalarles su propio servidor de página lo cual traerá ingresos de acuerdo al tiempo en que quieran mantener su página web, pero esta prestación de servicio de hosting traerá también costos ya que la instalación del ISP y el mantenimiento por mes tienen costos asociados. Al cabo de 1 año se determinará si el

proyecto produce más ganancias o pérdidas y si en general conviene o no seguir prestando el servicio por más tiempo.

Como ya se tiene el sistema operativo establecido y la programación definida en el sistema propietario, entonces una vez que se compran los equipos se puede hacer que circule la respectiva información del servidor HTTP programado en el sistema propietario.

Para prestarle el servicio de hosting a empresas o personas individuales, simplemente con el equipo ya instalado y el servidor HTTP propietario programado ya funcionando, entonces cada cliente que quiera contratar el servicio, sólo tendrá que entregar su correspondiente código fuente de página creada para que el programador se la instale en el servidor del sistema operativo propietario y así tenga su propio servidor web. Con esto se le cobrará un determinado precio al cliente y de esa forma se tendrán ingresos, con esto se tiene que el cliente una vez que contrata los servicios de hosting se le instala su página web en el servidor HTTP propietario y se le cobra una tarifa, que en el caso de prestar un servicio de hosting se cobra una tarifa mensual por mantención de dicha página en el servidor, si la quiere tener por un mes sólo se le cobrará por un mes, mientras que si la quiere tener por más tiempo tendrá que pagar la misma cantidad por cada mes que quiera tenerla habilitada.

Para los costos asociados al servidor HTTP simplemente se tiene el costo inicial de instalación y tarifa mensual en el primer mes para el ISP, la tarifa mensual del primer mes se hará en base a los días restantes del mes, para el segundo mes en adelante se paga al final del mes sólo la tarifa mensual normal, por tanto esto implicará en un costo fijo para el proyecto. No obstante no es el único costo asociado al proyecto ya que también se encuentra el costo de mantenimiento del servidor, es decir, cuanto es el costo por consumo de energía del servidor. Con estos dos costos se tendrá el costo fijo mensual.

Ahora para la instalación del ISP se tiene que de acuerdo a las ofertas de Entel no hay costo por instalación, por lo tanto esto derivará en que sólo hay costo mensual por mantención del ISP, el cual es de 1,05 UF, lo cual equivale a $27.535,52, redondeando queda en $27.536. Ahora en el primer mes se contrata el servicio a mediados de mes, por lo tanto se cobrará la mitad del costo de mantención mensual en este mes, es decir, $13.768, pero este costo no será siempre constante ya que dependerá del valor de la UF, por lo que cada seis meses sufrirá un reajuste, es decir, los primeros seis meses del primer año estará en $27.536, y los restantes seis meses en $30.000. Este costo se sumará al de mantención del servidor el cual se puede determinar de la siguiente manera:

Potencia del servidor = 350 (W) = 0,35 (KW)

Este consumo representa el consumo promedio de energía del servidor, ya que de acuerdo a las especificaciones técnicas del servidor entrega una potencia nominal de 350 (W), por lo tanto se estima en un consumo de 0,35 (KW).

Para determinar el costo de consumo de energía del servidor, se tendrá lo siguiente:

Costo de consumo de energía = 0,35 (KW) * 24 (H) * 30 (días) * 140 ($/KWh) = $35.280

Este costo representa el costo de consumo de energía del servidor, de 0,35 (KW) multiplicado por 24 horas da el consumo de energía en un día, eso multiplicado por 30 días entrega el consumo de energía en un mes, ya que el servidor de página siempre estará activo y disponible para quien lo utilize. El costo de consumo de energía por mes se determinará multiplicando el consumo de energía de un mes por el precio de referencia del KWh, el cual está en $140. Esto resulta en un costo de $35.280 al mes para el consumo de energía del servidor.

No obstante este costo no será para todos los meses del año, ya que cada seis meses el costo del KWh sufre un reajuste, por tanto el costo del KWh de $140 será por los primeros seis meses del primer año, y los restantes seis meses subirá a $150, esto llevará a que se tendrá un nuevo costo de consumo de energía de $37.800.

Para el costo total sólo se sumará el costo de mantención mensual del ISP más el costo mensual de consumo de energía.

Para generar ingresos, a los clientes individuales se les cobrará un precio por prestación de servicio de hosting, que será un precio mensual, el cual quedará establecido en $4.250, mientras que a las empresas se les cobrará un valor de $13.500. Estos valores multiplicados por la cantidad de clientes y empresas que adquieren el servicio en un mes determinarán los ingresos mensuales.

Para resumir toda la información relativa a ingresos y costos que generará el proyecto durante 1 año, se tendrá la tabla 6-2. Con esta tabla se podrá saber el saldo final de cada mes del año, lo cual permitirá tener ordenados todos los datos relevantes a la futura toma de decisiones sobre las posibles ganancias o pérdidas que generará el proyecto al cabo de 1 año.

	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
N° de Clientes	10	12	14	16	15	15	17	18	18	20	20	22
Ingresos por clientes	42.500	$51.000	$59.500	$68.000	$63.750	$63.750	$72.250	$76.500	$76.500	$85.000	$85.000	$93.500
N° de Empresas	5	6	6	7	8	8	10	9	10	12	14	15
Ingresos por Empresas	$67.500	$81.000	$81.000	$94.500	$108.000	$108.000	$135.000	$121.500	$135.000	$162.000	$189.000	$202.500
Ingresos Totales	$110.000	$132.000	$140.500	$162.500	$171.750	$171.750	$207.250	$198.000	$211.500	$247.000	$274.000	$296.000
Costo de ISP	$ 13.768	$27.536	$ 27.536	$ 27.536	$ 27.536	$ 27.536	$30.000	$ 30.000	$ 30.000	$ 30.000	$ 30.000	$ 30.000
Costo de Servidor	$ 35.280	$ 35.280	$ 35.280	$ 35.280	$ 35.280	$ 35.280	$ 37.800	$ 37.800	$ 37.800	$ 37.800	$ 37.800	$ 37.800
Costos totales	$ 49.048	$ 62.816	$ 62.816	$ 62.816	$ 62.816	$ 62.816	$ 67.800	$ 67.800	$ 67.800	$ 67.800	$ 67.800	$ 67.800
Utilidades	$ 60.952	$ 69.184	$ 77.184	$ 99.684	$ 108.934	$ 108.934	139.450	$ 130.200	$ 143.700	$ 179.200	$ 206.200	$ 228.200

Tabla 6-2: Tabla de utilidades mensuales

Para poder determinar al final la rentabilidad del proyecto, se calcula el VAN:

$$VAN = \sum_{t=1}^{n} \frac{BNt}{(1+i)^t} - I_0$$

Donde: $BN_t = Beneficio\ Neto\ del\ flujo\ en\ el\ período\ t$

$i = Tipo\ de\ interés$

$n = Número\ de\ períodos\ considerados$

$I_0 = Inversión\ Inicial$

En la tabla 6-3 se señala la interpretación que se le da al valor obtenido al calcular el VAN.

Valor	Interpretación	Decisión a tomar
VAN > 0	La inversión produciría ganancias por encima de la rentabilidad exigida.	El proyecto puede aceptarse.
VAN < 0	La inversión produciría pérdidas por debajo de la rentabilidad exigida.	El proyecto debería rechazarse.
VAN $= 0$	La inversión no produciría ni ganancias ni pérdidas.	Indiferente.

Tabla 6-3: Tabla de interpretación del VAN.

Cuando el VAN toma un valor igual a 0, i pasa a llamarse TIR (tasa interna de retorno). La TIR es la rentabilidad que nos está proporcionando el proyecto, es el punto de equilibrio entre las ganancias y las pérdidas que se pueden obtener en el proyecto.

Haciendo todos los cálculos requeridos se obtiene un VAN de -759.000,95 y una TIR de -1,5%, lo cual indica que al tener un VAN negativo el proyecto produce pérdidas y no es rentable, por lo tanto conviene cerrarlo. Con esto queda claro que ante los valores obtenidos el proyecto producirá un déficit económico y al cabo de 1 año no se podrá devolver la inversión inicial hecha al pedir el préstamo bancario.

Discusión y conclusiones

En este proyecto se analizó claramente el funcionamiento del protocolo HTTP, ante lo queda en evidencia cómo se maneja el proceso interno de un computador cuando un usuario se conecta a Internet, es decir, que es lo que pasa dentro del PC, como es su proceso interno de manejo de los datos, como es el proceso en que un usuario se comunica con un PC al conectarse a Internet. No obstante al conectarse a Internet queda también en evidencia la enorme cantidad de tráfico de transferencia de datos y de información, ya que cuando un usuario se conecta a Internet desde algún lugar público, la conexión de red que tiene escuchará no sólo sus solicitudes y conexiones a páginas web sino también las de otros usuarios conectados a dicha red, como pasa por ejemplo en la facultad de ingeniería de la PUCV, ya que la red a la que se suele conectar la mayoría de los alumnos es la misma y por tanto se acumulará un tráfico interminable de transferencias de paquetes de datos. Para esto mismo es que el proyecto sirvió para estudiar mejor ese tráfico de transferencias de información, separando el tráfico de otras redes y equipos ajenos al propio, y usando una conexión que permita analizar el tráfico local de información, es por esto que se utilizó el HUB presente en el laboratorio de Sistemas Digitales, lo cual fue fundamental para poder analizar en detalle el tráfico de información al abrir una página de Internet.

Con el estudio del protocolo HTTP quedó en evidencia que cuando un usuario se conecta a Internet y entra en contacto con el servidor hay distintas maneras por las cuales se lleva a la comunicación entre ambas partes, ya que lo más habitual es que el usuario pida determinada página web y determinada información al servidor, lo cual deriva en el llamado método de petición GET. Pero esto es sólo un método de comunicación, ya que como se analizó en el capítulo 2 el proceso de comunicación cliente-servidor es una etapa de solicitud/respuesta, y en la etapa de solicitud existen 8 métodos de petición del usuario al servidor, y para la etapa de respuesta existen muchas clases de código de respuesta del servidor al cliente. No obstante también quedó en evidencia que sólo suele utilizarse el método GET, ya que es el más inmediato dado sus características, y POST para llenar formularios, ante los cuales el servidor suele utilizar códigos de respuesta primordiales como 200 OK, 401 NOT AUTHORIZED o 404 NOT FOUND, pero también hay muchos métodos y códigos que casi no se ocupan ya que requieren de una situación muy especial para ser ocupados. La razón principal de porqué no se utilizan es que implican mucho riesgo, son muy peligrosos ya que cuando un servidor web no se encuentra

correctamente configurado y permite el uso público de algunos de los métodos no ocupados (especialmente aquellos que permiten manipular la estructura del servidor) es cuando aparecen los problemas. Métodos HTTP tales como PUT o DELETE permiten que un cliente pueda subir contenidos o eliminar elementos existentes en el servidor, pueden ser utilizados por un atacante para eliminar páginas o modificar contenidos existentes, por esta razón resulta evidente que no se debe permitir que cualquier cliente pueda utilizar dichos métodos contra un servidor web. Del mismo modo para los códigos de respuesta hay muchos tipos de códigos que no se ocupan ya que requieren de situaciones especiales y arriesgadas. De cualquier manera ante el estudio de todos estos métodos y códigos no ocupados, hay que decir que en este proyecto no fue necesario abarcar esos métodos y códigos especiales, no era parte de los objetivos, por tanto en los análisis de capturas de datos no se mostró un método diferente a GET, ya que la situación no lo requería.

Dado el título del tema de proyecto, el objetivo primordial era estudiar en profundidad el funcionamiento del protocolo HTTP, pero hay otra parte del título del proyecto que habla de analizar las transferencias de información o transacciones HTTP mediante metalenguajes, lo cual parecía no tener tanta importancia. Sin embargo también son importantes ya que dadas las funciones del metalenguaje y sus características, permiten analizar de alguna manera la estructura interna de una transacción HTTP, así como una oración se compone de sujeto, verbo y predicado, en este caso las transacciones HTTP se componen de determinados campos y elementos, y con las herramientas de metalenguajes se permite describir dichos campos y elementos de las transacciones HTTP. Como en el ejemplo en BNF del capítulo 3, se describió los elementos que componen una dirección postal, la sintaxis de ABNF para HTTP/1.1 presente en el apéndice B, describe todas las opciones que puedan tener los métodos de petición, códigos de respuesta y encabezados. Esto fue importante para poder describir los campos que se le querían dar a las tramas generadas en el servidor HTTP en sistema operativo propietario.

Con los distintos esquemas de red utilizados fue posible analizar detalladamente el tráfico local de información, y con esto fue posible comprender claramente que antes de que se generen las transacciones HTTP se deben generar las correspondientes transacciones TCP, ya que el protocolo HTTP pertenece al grupo de protocolos derivados de TCP/IP. Al haber transacciones TCP se indica que entre el usuario y el servidor se intenta establecer una conexión y que se recibe dicha conexión, sin esto no es posible generar transacciones HTTP.

Para el desarrollo del servidor HTTP propietario fue posible descubrir la importancia de tener un código HTML que no sea tan extenso y engorroso de describir, por esta razón se realizó con un código lo más sencillo posible, y la importancia de ocupar el SNIFFER para siempre poder comprobar y validar todo lo que se quiere lograr. En la etapa de programación en el sistema propietario simplemente cabe destacar que esto sirve de base para otros proyectos futuros que requieran de la utilización de un servidor HTTP en el sistema operativo propietario, así como en este proyecto se requirió del servidor DHCP y del servidor SIP. De esta manera se podrá también crear a futuro un servidor de página más sofisticado, con un código HTML más extenso y con más recursos, además de tal vez poder explorar la posibilidad de implantar los métodos y códigos que no suelen usarse por el riesgo que implican.

Respecto al estudio económico del proyecto, quedó en evidencia que el proyecto al final del período establecido de 1 año no es rentable ya que genera muchas más pérdidas que ganancias y no se podrá devolver la inversión inicial hecha con el préstamo bancario. Para haber llegado a tal punto es evidente que las utilidades generadas en cada mes fueron muy bajas respecto a lo que se necesitaba para lograr devolver la inversión inicial hecha con el préstamo bancario, los costos no en general eran bajos, pero los ingresos no eran los necesarios, pero ¿porqué no se obtuvieron los ingresos necesarios para devolver la inversión inicial hecha?. Los ingresos estaban dados de acuerdo a la tarifa mensual que se le cobraría a quienes quisieran adquirir el servicio de hosting, para ello se determinaron valores de referencia de acuerdo al mercado nacional, con la diferencia que en el mercado quedan establecidos como valores anuales, mientras que para el proyecto se dividieron dichos valores en 12 meses. No obstante dichos valores no fueron suficientes para generar las ganancias esperadas, ya que a pesar de que eran valores bajos y eso hacía que la demanda de clientes creciera cada vez más con el paso de los meses, tampoco se generaban tantas utilidades ya que al ser tarifas bajas se necesitaban demasiados clientes y empresas interesados en adquirir el servicio, y eso no era posible ya que en general se dispone de un servidor demasiado básico con lo cual las tarifas no podían ser altas y la demanda de clientes tampoco podía ser demasiado grande, y al final los resultados obtenidos fueron evidentes. Quizás a futuro creando un servidor más sofisticado será posible que el proyecto sea rentable ya que al tener tantos métodos y códigos no que no suelen ser utilizados mucha gente querrá adquirir dicho servicio y se podrá también elevar la tarifa de adquisición del servicio, lo cual haría posible generar las utilidades necesarias para la rentabilidad del proyecto.

Bibliografía

[1] Hypertext Transfer Protocol – Wikipedia, la enciclopedia libre; https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [Ultimo acceso: 11/09/16]

[2] El protocolo HTTP – CCM; es.ccm.net › Enciclopedia › Redes › Internet (protocolos). [Ultimo acceso: 16/10/16]

[3] Los códigos de estado HTTP – Libros Web; http://librosweb.es/tutorial/los-codigos-de-estado-de-http/. [Ultimo acceso: 16/10/16]

[4] ElevenPaths Blog: La seguridad en los métodos HTTP; blog.elevenpaths.com/2014/06/la-seguridad-en-los-metodos-http.html. [Ultimo acceso: 03/07/16]

[5] Metalenguaje – Ecured; https://www.ecured.cu/Metalenguaje. [Ultimo acceso: 03/07/16]

[6] Notación de Backus-Naur – Wikipedia, la enciclopedia libre; https://es.wikipedia.org/wiki/Notación_de_Backus-Naur. [Ultimo acceso: 04/09/16]

[7] Grammar: The language of languages (BNF, EBNF, ABNF and more); matt.might.net/articles/grammars-bnf-ebnf. [Ultimo acceso: 24/10/16]

[8] Augmented Backus‑Naur form – Wikipedia; https://en.wikipedia.org/.../Augmented_Backus–Nau..... [Ultimo acceso: 03/07/16]

[9] ABNF for HTTP (RFC 7230-7235); www.tech-invite.com/fo-abnf/tinv-fo-abnf-http.html. [Ultimo acceso: 03/07/16]

[10] HTML – Wikipedia, la enciclopedia libre; https://es.wikipedia.org/wiki/HTML. [Ultimo acceso: 31/07/16]

[11] [12] PC Factory - tu partner tecnológico; https://www.pcfactory.cl. [Ultimo acceso: 10/09/16]

[13] Entel; www.entel.cl/. [Ultimo acceso: 10/09/16]

A Apéndice

Documentación del lenguaje de programación Danux Compiler

En esta sección simplemente se detallará una documentación sobre la sintaxis del lenguaje de programación de DANUX. Se describen las extensiones agregadas al lenguaje de programación base de DANUX, el lenguaje PASCAL, citación.

A.1 Nuevos Tipos definidos en DANUX

"IPv4" Dirección IP en la versión 4 (largo 4 bytes)

Ejemplo: Definición Var My_IP : IPv4;

Counter : Word;

IP_Destin : IPv4;

IP_Source : IPv4;

Asignación My_IP := 192.168.0.1;

Referencia: IF My_IP = IP_Source Then

"MAC" Dirección MAC de dispositívos Ethernet (largo 6 bytes)

Ejemplo:

Definición Var Mac_server : MAC;

Mac_destin : MAC;

Asignación Mac_server:= 11_22_33_AA_DD_FF;

Referencia IF Mac_destin <> Mac_server Then

"MEM_POINTER" Puntero a una memoria física, creado para referenciar estructuras en memoria.

"PORT_POINTER" Puntero a una Puerta física, creado para referenciar estructuras en puertas.

"PARRAFO" Estructura tipo “String”, no tiene largo preestablecido, en memoria solo almacena los datos, se creó para manejar texto en las tramas Ethernet.

"DROW" Tipo “Integer” de dos byte (es Word invertido) los bytes en memoria son almacenados en orden invertido respecto de Word, en este caso se almacena primero el byte más significativo, se creó para soportar los campos especiales de las tramas TCP/IP.

A.2 Nuevos Operadores definidos en DANUX

Comparación de ejecución condicionada

"? " Operadores lógicos de control, asociado a una condición booleana condicionan la ejecución de las siguientes Instrucciones, se evalúa, si cumple sigue la ejecución del código, si no cumple toma la opción alternativa, si no cumple y no existe una alternativa retorna al nivel de anidamiento anterior informando que no se cumple la condición de ejecución en ese nivel.

Ejemplo:

? MY_ip = IP_Destin {Retorna con indicador cumple o no cumple}

{Cumple si MY_ip es igual a IP_Destin }

? IP_server <> IP_Destin {Retorna con indicador cumple o no cumple }

{Cumple si IP_server es distinto de IP_Destin}

"DESDE" Operador que delimita el inicio de un grupo de instrucciones de ejecución condicionadas.

"SINO" Operador que delimita un conjunto de instrucciones, conjunto que es una alternativa excluyente al grupo de instrucciones de ejecución condicionadas anterior.

"ADEMAS" Operador que delimita un conjunto de instrucciones, conjunto que es una alternativa no excluyente al grupo de instrucciones de ejecución condicionadas anterior.

"HASTA" Operador que delimita el fin de un grupo de instrucciones de ejecución condicionadas.

A.3 Nuevas Estructuras definidos en DANUX

"CAMPO" Estructura creado para soportar las complejas estructuras de datos del entorno TCP/IP.

En la declaración de un campo se asocia a un identificador un offset y un tipo, el compilador crea una variable del tipo a partir del offset aplicado a una variable base de referencia.

Ejemplo, descripción del inicio de una trama Ethernet utilizando Campos:

```
Campo   {-- Ethernet IP Header --}

        Version   =   14  Byte;

        Servicio   =   15  Byte;

        Largo     =   16  Word;

        Identif     =   18  Word;

        Flags     =   20  Word;

        TTL       =   22  Byte;

        Protocolo   =   23  Byte;

        Checksum  =  24  Word;

        SOU_IP    =   26  IPv4;

        DES_IP     =   30   IPv4;

        {----- TCP HEADER ----}, etc
```

Asignación: TRAMA_IN [i] | TTL:= $20; {después del caracter "|" debe seguir un Campo}

La variable destino en el ejemplo de asignación es establecida por el caracter "|" asociado a un identificador de campo (TTL en el ejemplo), esto construye una variable del tipo indexado al identificador de campo (Byte en el ejemplo) en una posición de memoria determinada por la dirección base (dirección de TRAMA_IN [i] en el ejemplo) más el Offset declarado en el campo referenciado (22 como Offset de TTL en el ejemplo).

La utilización de campos permiten reducir y simplificar las referencias a estructuras complejas con múltiples opciones de datos en una misma posición de memoria, situación que ocurre en las tramas de datos de los protocolos utilizados en el entorno TCP/IP.

"PRODUCCION": Las estructuras producción fueron exportadas desde BNF y permiten en DANUX utilizar instrucciones con ejecución condicionada, la ejecución de una producción se inicia en una DESDE y termina con una HASTA.

Ejemplo: Declaración de una Producción

```
PRODUCCION puerto_tcp;
Desde  ? trama_in[i] | des_port = puerto_http;
            Writeln ('Protocolo HTTP');
   sino  (1) ? trama_in[i] | des_port = puerto_ftp;
            Writeln ('Protocolo FTP') ;
    sino  (2) ? trama_in[i] | des_port = puerto_telnet;
             Writeln ('Protocolo Telnet');
   sino   (3) Writeln ('Protocolo desconocido');
Hasta;
```

Si la trama TCP tiene el puerto destino igual a puerto_http la ejecución de esta producción escribirá 'Protocolo HTTP' y retornará cumpliendo la producción puerto_tcp.

Si el puerto destino no es igual a puerto_http la ejecución salta a (1) punto que sigue el próximo SINO.

Si la trama TCP tiene el puerto destino igual a puerto_ftp la ejecución de esta producción escribirá 'Protocolo FTP' y retornará cumpliendo la producción puerto_tcp.

Si el puerto destino no es igual a puerto_ftp la ejecución salta a (2) punto que sigue el próximo SINO.

Si la trama TCP tiene el puerto destino igual a puerto_telnet la ejecución de esta producción escribirá 'Protocolo TELNET' y retornará cumpliendo la producción puerto_tcp.

Si el puerto destino no es igual a puerto_telnet la ejecución salta a (3) punto que sigue el próximo SINO, se escribirá 'Protocolo desconocido' y retornará cumpliendo la producción puerto_tcp.

A continuación un ejemplo de una calculadora simple, (con las cuatro operaciones básicas) desarrollada en lenguaje DANUX.

```
PROGRAM CALCULADORA; {Calculadora básica}
Var  Operando1: Longint ;
        Operando2: Longint ;
        Car       : Char ;
PRODUCCION cantidad; {declaración de una producción local}
Desde   Read (Car);
  Desde   ? Car   = "0" ;   Operando2 := Operando2*10 + 0 ;
cantidad;
     Sino    ? Car   = "1" ;   Operando2 := Operando2*10 + 1;
cantidad;
     Sino    ? Car = "2" ;    Operando2 := Operando2*10 + 2 ;
cantidad;
    Sino     ? Car = "3" ;    Operando2 := Operando2*10 + 3 ;
```

```
cantidad;
    Sino     ? Car = "4" ;    Operando2 := Operando2*10 + 4 ;
cantidad;
    Sino     ? Car = "5" ;    Operando2 := Operando2*10 + 5 ;
cantidad;
    Sino     ? Car = "6" ;    Operando2 := Operando2*10 + 6 ;
cantidad;
    Sino     ? Car = "7" ;    Operando2 := Operando2*10 + 7 ;
cantidad;
    Sino      ? Car = "8" ;    Operando2 := Operando2*10 + 8 ;
cantidad;
    Sino       ? Car = "9" ;   Operando2 := Operando2*10 + 9 ;
cantidad;
  Hasta;
Hasta;
PRODUCCION Operacion; {declaración de una producción local}
Desde   Read (car);
  Desde   ? Car = ´+´;
     Sino    ? Car = ´-´;
     Sino     ? Car = ´*´;
     Sino     ? Car = ´/´ ;
  Hasta;
Hasta;
Desde;     {Producción Principal}
   While True do
        Desde {Inicio del Loop de repetición permanente}
              Operando2:= 0 ;
              Cantidad; {referencia a una producción local}
              Operando1:=Operando2;{entra   a   este   punto   si
produccion Cantidad cumplio}
               Desde
                   Operacion;
                    Desde {Operando1 y Operación cumplieron, son
legales}
                           Operando2  := 0 ;
                           Cantidad;
                         {En este punto Operando1, Operación y
Operando2 son legales}
                          {Se ejecuta el cálculo e imprime el
resultado final      }
                               Desde  ?  Car  =  ´+´;    Writeln
(Operando1+Operando2)
                                 Sino   ? Car = ´-´;   Writeln
(Operando1-Operando2)
                                   Sino   ? Car = ´*´; Writeln
(Operando1*Operando2)
                                     Sino  ? Car = ´/´;  Writeln
(Operando1/Operando2)
                                     Hasta;
                              Sino {en este punto Operando2 no
cumplió es ilegal}
                                     Writeln    (´Se    esperaba
numero´);
                            Hasta;   {todo cumplio y se imprimió
el resultado}
                   Sino {en este punto Operación es ilegal}
                           Writeln (´Se esperaba operación´);
               Hasta;  {fin de la opción cumplió Operando1}
         Sino {en este punto Operando1 no cumplió es ilegal}
                   Writeln (´Se esperaba numero´);
      Hasta; {Fin del Loop de repetición}
```

B Apéndice

Aplicación de metalenguajes al protocolo HTTP

Dado este tema de proyecto es importante recalcar la aplicación que pueden tener los metalenguajes al protocolo HTTP, ante lo cual tenemos que se utilizarán herramientas de ABNF para HTTP/1.1, descritas en la normativa RFC 7230 a la 7235, separando los distintos parámetros, elementos, campos, presentes en la estructura de cada transacción HTTP involucrada, es decir, de que campos y elementos se forma la estructura sintáctica de cada método de petición, código de respuesta y encabezados tanto para petición como para respuesta. (ABNF, ya que es el tipo de metalenguaje más amplio para abarcar los distintos elementos y campos que se quieren emplear, y permite especificar cantidades y repeticiones de determinados elementos). [9]

ABNF para HTTP / 1.1 - RFC 7230 hasta 7235

(Esto reemplaza a RFC 2616 y determina la sintaxis de ABNF para HTTP/1.1)

- RFC 7230 - HTTP / 1.1: Sintaxis de mensajes y enrutamiento
- RFC 7231 - HTTP / 1.1: Semántica y contenido
- RFC 7232 - HTTP / 1.1: Solicitudes condicionales
- RFC 7233 - HTTP / 1.1: Rango de solicitudes
- RFC 7234 - HTTP / 1.1: Almacenamiento en caché
- RFC 7235 - HTTP / 1.1: autenticación

Para describir la estructura sintáctica de una transacción HTTP, se realizará de acuerdo al siguiente orden:

1) Formato de mensaje

2) Campos de cabecera

3) URI y Solicitud Objetivo

4) codificaciones de transferencia

5) Fecha / Hora

6) Rango Unidades

7) Desafío-Respuesta Autenticación

B.1 HTTP / 1.1 - Formato de Mensaje

(RFC 7230 – Sección 3)

- HTTP-message = start-line

* (header-field CRLF) CRLF

[message-body]

start-line = request-line / status-line

request-line = method SP request-target SP HTTP-version CRLF

status-line = HTTP-version SP status-line SP reason-phrase CRLF

method = token

; " GET " (RFC 7231 – Sección 4.3.1)

; " HEAD " (RFC 7231 – Sección 4.3.2)

; " POST " (RFC 7231 – Sección 4.3.3)

; " PUT " (RFC 7231 – Sección 4.3.4)

; " DELETE " (RFC 7231 – Sección 4.3.5)

; " CONNECT " (RFC 7231 – Sección 4.3.6)

; " OPTIONS " (RFC 7231 – Sección 4.3.7)

; " TRACE " (RFC 7231 – Sección 4.3.8)

HTTP-version = HTTP-name / DIGIT " . " DIGIT

HTTP-name = % X48.54.54.50 ; "HTTP", case-sensitive

message-body = * OCTET

reason-phrase = * (HTAB / SP / VCHAR / obs-text)

status-code = 3 DIGIT

; 1xx Informational

; 100 Continue (RFC 7231 - Sección 6.2.1)

; 101 Switching Protocols (RFC 7231 - Sección 6.2.2)

; 2xx Successful

; 200 OK (RFC 7231 - Sección 6.3.1)

; 201 Created (RFC 7231 – Sección 6.3.2)

; 202 Accepted (RFC 7231 – Sección 6.3.3)

; 203 Non-Authoritative Information (RFC 7231 - Sección 6.3.4)

; 204 No Content (RFC 7231 - Sección 6.3.5)

; 205 Reset Content (RFC 7231 - Sección 6.3.6)

; 206 Partial Content (RFC 7233 - Sección 4-1)

; 3xx Redirection

; 300 Multiple Choices (RFC 7231 - Sección 6.4.1)

; 301 Moved Permanently (RFC 7231 - Sección 6.4.2)

; 302 Found (RFC 7231 - Sección 6.4.3)

; 303 See Other (RFC 7231 - Sección 6.4.4)

; 304 Not Modified (RFC 7232 - Sección 4.1)

; 305 Use Proxy (RFC 7231 - Sección 6.4.5)

; 306 (Unused) (RFC 7231 - Sección 6.4.6)

; 307 Temporary Redirect (RFC 7231 - Sección 6.4.7)

; 4xx Client Error

```
; 400  Bad Request       (RFC 7231 - Sección 6.5.1)
; 401    Unauthorized  (RFC 7235 - Sección 3.1)
; 402    Payment Required      (RFC 7231 - Sección 6.5.2)
; 403    Forbidden      (RFC 7231 - Sección 6.5.3)
; 404    Not Found     (RFC 7231 - Sección 6.5.4)
; 405    Method Not Allowed    (RFC 7231 - Sección 6.5.5)
; 406    Not  Acceptable           (RFC 7231 - Sección 6.5.6)
; 407  Proxy Authentication Required  (RFC 7235 - Sección 3.2)
; 408    Request Timeout   (RFC 7231 - Sección 6.5.7)
; 409    Conflict           (RFC 7231 - Sección 6.5.8)
; 410    Gone     (RFC 7231 - Sección 6.5.9)
; 411    Length Required         (RFC 7231 - Sección 6.5.10)
; 412    Precondition Failed    (RFC 7232 - Sección 4.2)
; 413     Payload Too Large    (RFC 7231 - Sección 6.5.11)
; 414    URI Too Long   (RFC 7231 - Sección 6.5.12)
; 415    Unsupported Media Type   (RFC 7231 - Sección 6.5.13)
; 416    Range Not Satisfiable   (RFC 7233 - Sección 4.4)
; 417    Expectation Failed     (RFC 7231 - Sección 6.5.14)
; 426    Upgrade Required      (RFC 7231 - Sección 6.5.15)
; 5xx    Server Error
; 500    Internal Server Error   (RFC 7231 - Sección 6.6.1)
; 501    Not Implemented        (RFC 7231 - Sección 6.6.2)
; 502    Bad Gateway     (RFC 7231 - Sección 6.6.3)
 ; 503   Service Unavailable  (RFC 7231 - Sección 6.6.4)
; 504    Gateway Timeout  (RFC 7231 - Sección 6.6.5)
```

; 505 HTTP Version Not Supported (RFC 7231 - Sección 6.6.6)

B.2 HTTP / 1.1 - Campos de cabecera

(RFC 7230 – Sección 3.2; ver RFC 7230 – Sección 7)

header-field = field-name " : " OWS field-value OWS

field-name = token

field-value = * (field-content / obs-fold)

field-content = field-vchar [1 * (SP / HTAB) field-vchar]

field-vchar = VCHAR / obs-text

obs-fold = CRLF 1 * (SP / HTAB) ; ver RFC 7230 - Sección 3.2.4

- Accept

Accept = # (media-range [accept-params])

media-range = (" * / * "/ (type " / "" * ") / (type " / " subtype)) * (OWS " ; " OWS parameter)

Accept-params = weight * (accept-ext)

accept-ext = OWS " ; " OWS token [" = " (token / quoted-string)]

- Accept-Charset (RFC 7231 - Sección 5.3.3)

Accept-Charset = 1 # ((charset / " * ") [weight])

- Accept-Encoding (RFC 7231 - Sección 5.3.4)

Accept-Encoding = # (codings [weight])

Codings = content-coding / "identity" / "*"

- Accept-Language (RFC 7231 - Sección 5.3.5)

Accept-Language = 1# (language-range [weight])

language-range = <language-range, ver RFC 4647 – Sección 2.1>

- Accept-Ranges (RFC 7233 - Sección 2.3)

Accept-Ranges = aceptable-ranges

aceptable-ranges = 1#range-unit / "none"

- Age (RFC 7234 - Sección 5.1)

Age = delta-seconds

- Allow (RFC 7231 - Sección 7.4.1)

Allow = #method

- Authorization (RFC 7235 - Sección 4.2)

Authorization = #credentials

- Cache-control (RFC 7234 - Sección 5.2)

Cache-Control = 1 #cache-directive

Cache-directive = token

[" = " (token / quoted-string)] ; max-age (RFC 7234 - Sección 5.2.1.1, 5.2.2.8) ; max-stale (RFC 7234 - Sección 5.2.1.2) ; min-fresh (RFC 7234 - Sección 5.2.1.3) ; must-revalidate (RFC 7234 - Sección 5.2.2.1) ; no-cache (RFC 7234 - Sección 5.2.1.4, 5.2.2.2) ; no-store (RFC 7234 - Sección 5.2.1.5, 5.2.2.3) ; no-transform (RFC 7234 - Sección 5.2.1.6, 5.2.2.4) ; only-if-cached (RFC 7234 - Sección 5.2.1.7) ; private (RFC 7234 - Sección 5.2.2.6) ; proxy-revalidate (RFC 7234 - Sección 5.2.2.7) ; public (RFC 7234 - Sección 5.2.2.5) ; s-maxage (RFC 7234 - Sección 5.2.2.9)

- Connection (RFC 7230 - Sección 6.1)

Connection = 1 #connection-option

Connection-option = token

- Content-Encoding (RFC 7231 - Sección 3.1.2.2)

Content-Encoding = 1 #content-coding

- Content-Language (RFC 7231 - Sección 3.1.3.2)

Content-Language = 1 #language-tag

- Content-Length (RFC 7230 - Sección 3.3.2)

Content-Length = 1 * DIGIT

- Content-Location (RFC 7231 - Sección 3.1.4.2)

Content-Location = absolute-URI / partial-URI

- Content-Range (RFC 7233 - Sección 4.2)

Content-Range = byte-content-range / other-content-range

byte-content-range = bytes-unit SP (byte-range-resp / unsatisfied-range)

byte-range-resp = byte-range " / " (complete-length / " * ")

byte-range = first-byte-pos " - " last-byte-pos

unsatisfied-range = " * / " complete-length

complete-length = 1 * DIGIT

other-content-range = other-range-unit SP other-range-resp

other-range-resp = * CHAR

- Content-Type (RFC 7231 - Sección 3.1.1.5)

Content-Type = media-type

- Date (RFC 7231 - Sección 7.1.1.2)

Date = HTTP-date

- ETag (RFC 7232 - Sección 2.3)

ETag = entity-tag

entity-tag = [weak] opaque-tag

weak = x57.2F% ; " W / ", case-sensitive

opaque-tag = DQUOTE * etagc DQUOTE

etagc = x21% /% x23-7E / obs-text ; VCHAR except double quotes, plus obs-text

- Expect (RFC 7231 - Sección 5.1.1)

Expect = "100-continue"

- Expires (RFC 7234 - Sección 5.3)

expires = HTTP-date

- From (RFC 7231 - Sección 5.5.1)

From = mailbox

- Host (RFC 7230 - Sección 5.4)

Host = uri-host [" : " port]

- If-Match (RFC 7232 - Sección 3.1)

If-Match = " * " / 1 # entity-tag

- If-Modified-Since (RFC 7232 - Sección 3.3)

If-Modified-Since = HTTP-date

- If-None-Match (RFC 7232 - Sección 3.2)

If-None-Match = " * " / 1 # entity-tage

- If-range (RFC 7233 - Sección 3.2)

If-range = entity-tag / HTTP-date

- If-Unmodified-Since (RFC 7232 - Sección 3.4)

If-Unmodified-Since = HTTP-date

- Last-Modified (RFC 7232 - Sección 2.2)

Last-Modified = HTTP-date

- Location (RFC 7231 - Sección 7.1.2)

Location = URI-reference

- Max-Forwards (RFC 7231 - Sección 5.1.2)

Max-Forwards = 1 * DIGIT

- MIME-Versión (RFC 7231 - Anexo A.1)

- Pragma (RFC 7234 - Sección 5.4)

Pragma = 1 #pragma-directive

pragma-directive = "No-cache" / extension-pragma

extension-pragma = token [" = " (token / quoted-string)]

- Proxy-Authenticate (RFC 7235 - Sección 4.3)

Proxy-Authenticate = 1 #challenge

- Proxy-Autorization (RFC 7235 - Sección 4.4)

Proxy-Autorization = credentials

- Range (RFC 7233 - Sección 3.1)

Range = byte-ranges-specifier / other-ranges-specifier

other-ranges-specifier = other-range-unit " = " other-range-set

other-range-set = 1 * VCHAR

- Referer (RFC 7231 - Sección 5.5.2)

Referer = absolute-URI / partial-URI

- Retry-After (RFC 7231 - Sección 7.1.3)

Retry-After = HTTP-date / delta-seconds

delta-seconds = 1 * DIGIT

- Server (RFC 7231 - Sección 7.4.2)

Server = product * (RWS (product / comment))

- TE (RFC 7230 - Sección 4.3)

TE = # #t-codings

t-codings = "trailers" / (transfer-coding [t-ranking])

t-ranking = OWS " ; " AA " q = " rank

rank = (" 0 " [" . " 0 * 3 DIGIT]) / (" 1 " [" . " 0 * 3 (" 0 ")])

- Trailer (RFC 7230 - Sección 4.4)

Trailer = 1#field-name

- Transfer-Encoding (RFC 7230 - Sección 3.3.1)

Transfer-Encoding = 1 #transfer-coding

- Upgrade (RFC 7230 - Sección 6.7)

Upgrade = 1 #protocol

protocol = protocol-name [" / " protocol-version]

protocol-name = token

protocol-version = token

- User-Agent (RFC 7231 - Sección 5.5.3)

User-Agent = product * (RWS (product / comment))

product = token ["/" product-version]

product-version = token

- Vary (RFC 7231 - Sección 7.1.4)

Vary = "*" / 1 #field-name

- Via (RFC 7230 - Sección 5.7.1)

Via = 1 #(received-protocol RWS received-by [RWS comment])

received-protocol = [Protocol-name "/"] protocol-version

received-by = (URI-host [":" port]) / pseudonym

pseudonym = token

- Warning (RFC 7234 - Sección 5.5)

Warning = #1 warning-value

warning-value = warn-code SP warn-agent SP warn-text [SP warn-date]

warn-code = 3 DIGIT

; 110 Response is Stale [RFC 7234 - Sección 5.5.1]

; 111 Revalidation Failed [RFC 7234 - Sección 5.5.2]

; 112 Disconnected Operation [RFC 7234 - Sección 5.5.3]

; 113 Heuristic Expiration [RFC 7234 - Sección 5.5.4]

; 199 Miscellaneous Warning [RFC 7234 - Sección 5.5.5]

; 214 Transformation Applied [RFC 7234 - Sección 5.5.6]

; 299 Miscellaneous Persistent Warning [RFC 7234 - Sección 5.5.7]

warn-agent = (URI-host [":" port]) / pseudonym

warn-text = quoted-string

warn-date = DQUOTE HTTP-date DQUOTE

- WWW_Authenticate (RFC 7235 - Sección 4.1)

WWW-Authenticate = 1 #challenge

B.3 HTTP / 1.1 - URI y Solicitud Target

(RFC 7230 – Sección 2.7; RFC 7230 – Sección 5.3)

absolute-path = 1 * ("/" segment)

partial-URI = relativ-part ["?" query]

http-URI = "http: ""//" authority path-abempty ["?" query] ["#" fragment]

https-URI = "https: ""//" authority path-abempty ["?" query] ["#" fragment]

request-target = origin-form / absolute-form / authority-form / asterisc-form

origin-form = absolute-path ["?" query]

absolute-form = absolute-URI

authority-form = authority

asterisc-form = " * "

B.4 HTTP / 1.1 - codificaciones de transferencia

(RFC 7230 – Sección 4)

Transfer-coding = "chunked"/ ; (RFC 7230 - Sección 4.1)

"compress"/ ; (RFC 7230 - Sección 4.2.1)

"deflate"/ ; (RFC 7230 - Sección 4.2.2)

"gzip"/ ; (RFC 7230 - Sección 4.2.3)

transfer-extension

transfer-extension = token * (OWS " ; " OWS transfer-parameter)

transfer-parameter = token BWS " = " BWS (token / quoted-string)

chunked-body = * chunk last-chunk tráiler-part CRLF

chunk = chunk-size [chunk-ext] CRLF chunk-data CRLF

chunk-size = 1 * HEXDIG

last-chunk = 1 * (" 0 ") [chunk-ext] CRLF

chunk-data = 1 * OCTET ; a sequence of chunk-size octets

chunk-ext = * (" ; " chunk-ext-name [" = " chunk-ext-val])

chunk-ext-name = token

chunk-ext-val = chunk / quoted-string

tráiler-part = * (header-field CRLF)

B.5 HTTP/1.1 - Fecha/Hora

(RFC 7231 – Sección 7.1; RFC 7234 – Sección 1.2.1)

HTTP-date = IMF-fixdate / obs-date

IMF-fixdate = Day-name " , " SP date 1 SP time-of-day SP GMT

; fixed-length / zone / capitalization

; subset of the format

; ver RFC5322 - Sección 3.3

Day-name = % X4D.6F.6E / ; " Mon " Case-sensitive

% X54.75.65 / ; " Tue " Case-sensitive

% X57.65.64 / ; " Wed " Case-sensitive

% X54.68.75 / ; " Thu " Case-sensitive

% X46.72.69 / ; " Fri " Case-sensitive

% X53.61.74 / ; " Sat " Case-sensitive

% X53.75.6E / ; " Sun " Case-sensitive

```
date 1    = day  SP  month  SP  year   ;  por ejemplo, 02 de junio 1982

day  =  2 DIGIT

month  =  % X4A.61.6E /          ; " Jan "   Case-sensitive

          % X46.65.62 /     ; " Feb "   Case-sensitive

          % X4D.61.72 /    ; " Mar "   Case-sensitive

          % X41.70.72 /     ; " Apr "   Case-sensitive

          % X4D.61.79 /    ; " May "   Case-sensitive

          % X4A.75.6E /    ; " Jun "   Case-sensitive

          % X4A.75.6C /    ; " Jul "   Case-sensitive

          % X41.75.67 /     ; " Aug "   Case-sensitive

          % X53.65.70 /     ; " Sep "   Case-sensitive

          % X4F.63.74 /     ; " Oct "   Case-sensitive

          % X4E.6F.76 /     ; " Nov "   Case-sensitive

          % X44.65.63 /  ; " Dec "   Case-sensitive

year  =  4 DIGIT

GMT  =  % X47.4D.54 ; " GMT "   Case-sensitive

time-of-day  =  hour  " : "  minute  " : "  second   ; 00:00:00 - 23:59:60 (leap second)

hour  =  2 DIGIT

minute  =  2 DIGIT

second  =  2 DIGIT

obs-date  =  RFC 850 - date / asctime-date

RFC 850-date    =  day-name-l  " , "  SP  date 2  SP  time-of-day  SP  GMT

Date 2  =  day  " - "  month  " - "  2 DIGIT          ; por ejemplo, 02-Jun-82

day-name-l  =  % X4D.6F.6E.64.61.79 / ;    " Mon "   Case-sensitive

               % X54.75.65.73.64.61.79 /     ;    " Tue "   Case-sensitive
```

% X57.65.64.6E.65.73.64.61.79 / ; " Wed " Case-sensitive

% X54.68.75.72.73.64.61.79 / ; " Thu " Case-sensitive

% X46.72.69.64.61.79 / ; " Fri " Case-sensitive

% X53.61.74.75.72.64.61.79 / ; " Sat " Case-sensitive

% X53.75.6E.64.61.79 / ; " Sun " Case-sensitive

asctime-date = day-name SP date 3 SP time-of-day SP year

date 3 = month SP (2 DIGIT / (SP 1DIGIT)) ; por ejemplo, 02 de junio

delta-seconds = 1 * DIGIT ; RFC 7234 – Sección 1.2.1

B.6 HTTP / 1.1 - Unidades del Rango

(RFC 7233 – Sección 2)

range-unit = byte-unit / other-range-unit

byte-unit = "bytes"

byte-ranges-especifier = bytes-unit " = " byte-range-set

byte-range-set = 1 #(byte-range-spec / suffix-byte-range-spec)

byte-range-spec = first-byte-pos " - " [last-byte-pos]

first-byte-pos = 1 * DIGIT

last-byte-pos = 1 * DIGIT

suffix-byte-range-spec = " - " suffix-length

suffix-length = 1 * DIGIT

other-range-unit = token

B.7 HTTP / 1.1 - Desafío-Respuesta de autenticación

(RFC 7235 – Sección 2.1)

auth-scheme = token

auth-param = token BWS " = " BWS (token / quoted-string)

token68 = 1 * (ALPHA / DIGIT / " - "" / . "/" _ "/" ~ "/" + "/" / ") *" = "

challenge = auth-scheme [1 * SP (token68 / #auth-param)]

credentials = auth-scheme [1 * SP (token68 / #auth-param)]

B.8 HTTP / 1.1 - Reglas Oficiales

Los espacios en blanco (RFC 7230 – Sección 3.2.3)

OWS = * (SP / HTAB) ; espacio en blanco opcional

RWS = 1 * (SP / HTAB) ; espacio en blanco requerido

BWS = OWS ; "malo" espacios en blanco

B.9 Componentes de valor del campo

(RFC 7230 – Sección 3.2.6)

token = 1 * TCHAR

TCHAR = " ! " / " # " / " $ " / " % " / " Y " / " " "/" * "/

" + "/" - "/" . " / " ^ " / " _ " / " ` "/" | " / " ~ " / DIGIT / ALPHA ; cualquier VCHAR, excepto delimitadores.

quoted-string = DQUOTE * (qdtext / quoted-par) DQUOTE

qdtext = HTAB / SP / x21% / %x23-5B / %X5d-7E / obs-text

obs-text = %X80-FF

comment = " (" * (Ctext / quoted-pair / comment) ") "

ctext = HTAB / SP / x21-27% / %X2a-5B / % X5d-7E / obs-text

quoted-pair = " \ " (HTAB / SP / VCHAR / obs-text)

B.10 Representación de metadatos

(RFC 7231 – Sección 3.1)

media-type = type " / " subtype * (OWS " ; " OWS parameter)

type = token

subtype = simbólico

parameter = token " = " (token / quoted-string)

charset = token

content-coding = token

language-tag = <Language-Tag ; ver RFC 5646 - Sección 2.1>

B.11 Los valores de calidad

(RFC 7231 – Sección 5.3.1)

weight = OWS " ; " AA " q = " qvalue

qvalue = (" 0 " [" . " 0 * 3 DIGIT]) / (" 1 " [" . " 0 * 3 (" 0 ")])