CAPÍTULO 2
LA CERCANÍA CON EL USUARIO (HTTP)

Luis Marrone

Paradigmas de Aplicaciones

Las aplicaciones o servicios en el mundo de TCP/IP no son otra cosa que programas que satisfacen necesidades de los usuarios como:

- Acceder a un equipo de cómputo remoto
- Transferir un archivo
- Acceder a una página Web
    - Hacer una transacción bancaria
    - Hacer compras
    - Inscribirse en un curso
    - Sostener una reunión de trabajo o una clase
    - Escuchar música
    - ⋮
- Enviar/recibir un correo electrónico
- Mantener una conversación telefónica

Terminamos la lista aquí para darle un fin. No es exhaustiva ni mucho menos.

Habíamos mencionado en el capítulo anterior que a la hora de programar estas aplicaciones el desarrollador normalmente se va a encontrar con dos paradigmas disponibles, el de *cliente-servidor* y el de *peer-to-peer*.

Paradigma Cliente-Servidor

- Es un caso más de comunicación entre procesos
- Los servidores se implementan como programas de aplicación
- Resultan así transportables a todo sistema que soporte comunicaciones TCP/IP
- Generalmente tienen un hardware (computador) dedicado a ellos. Así es que se hace referencia a la máquina como servidor

Aprovechamos para tener presentes características del cliente y del servidor.

Servidor

- Invocado automáticamente en el arranque de la máquina
- Espera pasivamente la llegada de peticiones de clientes
- Puede gestionar peticiones simultáneas de varios clientes
- En la misma máquina pueden estar funcionando varios servidores de diferentes Servicios
- Se suele llamar también "servidor" a la máquina donde se ejecuta el programa servidor. Inclusive se hace un uso extensivo del término para identificar equipos de altas prestaciones y que no necesariamente obedecen a este paradigma.
- Dispone de un port bien conocido y/o reservado por el IANA[1]

Cliente

- Invocado por el usuario
- Inicia el contacto con el servidor
- Puede comunicarse con:
  - varios servidores alternativamente
  - varios servidores simultáneamente
  - el mismo servidor concurrentemente
- Dispone de un port efímero dado por el sistema operativo

Independientemente de cual se adopte una vez más tendremos un intercambio de mensajes acorde con la estructura adoptada por la aplicación y también ese mensaje será encapsulado en el Nivel de Transporte según nuestro modelo planteado de TCP/IP. Recordemos que el nivel de transporte adoptado (protocolos TCP o UDP) dependerá de la naturaleza del tráfico propio de la aplicación y/o de los requerimientos de la misma.

Sucintamente podemos decir que si la aplicación requiere confiabilidad y no un tiempo de respuesta comprometido normalmente se lo encapsulará en TCP. Si, por el contrario, el tráfico que genera es de

[1] Organismo que asigna número de ports a las aplicaciones

tiempo real y/o no requiere un alto grado de confiabilidad, entonces lo encapsulará en UDP. Veremos más detalles en el capítulo dedicado al Nivel de Transporte.

Para concluir veamos una figura en la que representamos algunas aplicaciones bajo este paradigma con los port bien conocidos asociados y su ubicación en el modelo de TCP/IP

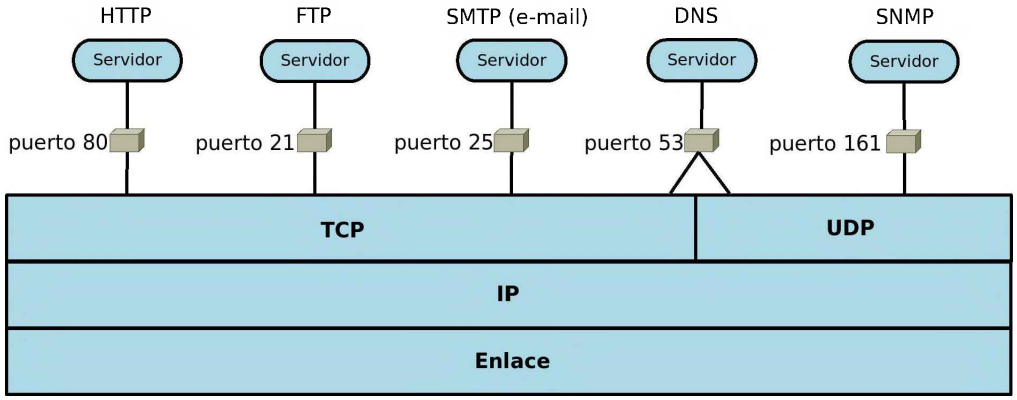

Figura 2.1: Nivel de Aplicación en TCP/IP

El otro modelo, *peer-to-peer*, presenta estas características:

Paradigma Peer-to-Peer

- Compartir recursos.
- Procesamiento distribuido.
- Procesos colaborativos.
- Es un caso más de comunicación entre procesos.
- Red de usuarios.
- Dualidad cliente-servidor en cada nodo.

Con este paradigma se han desarrollado numerosas aplicaciones que contribuyen a un porcentaje importante en el tráfico presente en Internet. Presentan algunos problemas de seguridad y de viabilidad en presencia de redes privadas virtuales y NAT que de alguna manera se verán disminuidos con la penetración de IPv6 en una red como Internet. Nosotros nos vamos a ocupar del análisis de aplicaciones basadas en el paradigma Cliente-Servidor, como es el caso de las que utilizan a *http* como protocolo de aplicación.

Antes de incursionar en *http* en detalle tengamos presente características comunes de las aplicaciones desarrolladas bajo este paradigma.

Ante todo tengamos en cuenta que será un software residente en el equipo del cliente/servidor y, como tal, tendrá que interactuar con el sistema operativo. Además, se requiere que la aplicación pueda dar su servicio entre cliente y servidor con diferentes sistemas operativos en cada uno de ellos.

La solución llegó con la implementación de una terminal virtual que brinda una interfaz entre el SO y la aplicación propiamente dicha y logra la independencia del SO en cuanto a su funcionalidad. Esa terminal virtual fue implementada en la que podemos decir fue la primera aplicación basada en ese paradigma cual es el caso de Telnet. Terminal que en mayor o menor grado la tendremos presente en todas las aplicaciones del tipo mencionado anteriormente.

Dado el alcance y extensión del presente libro es que decidimos ver en detalle uno de los servicios más populares como es el caso del servicio WEB mediante el protocolo HTTP [BLFF96]. Mencionamos, siempre dentro del paradigma cliente-servidor, otros servicios como:

Telnet: acceso remoto a otro dispositivo y que de alguna manera dio pie al desarrollo del resto de las aplicaciones.

FTP: transferencia de archivos entre dispositivos con diferentes plataformas.

SMTP: servicio de correo electrónico.

SNMP: gestión de la red.

Siempre dentro del mismo paradigma pero con carácter de auxiliares por cuanto mejoran la performance y funcionalidad de otros servicios:

DHCP: completa la configuración de un nodo IP automáticamente.

DNS: resuelve nombres de recursos a direcciones IP y viceversa.

Como adelantamos en varias ocasiones nos ocuparemos luego en detalle de DNS.

Servicio WEB - HTTP

Para comenzar no podemos dejar de incluir la Figura 2.2 que representa la primer página Web como la conocemos actualmente provista por un servidor del CERN[2], lugar donde nació este servicio. Esta

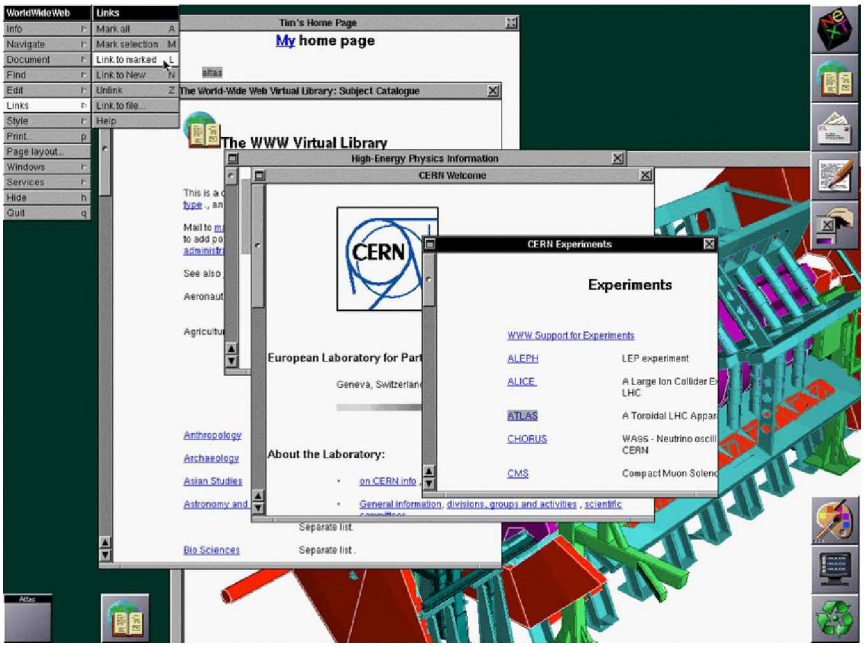

Figura 2.2: Portal del Cern - 1989

aplicación entrega a pedido del cliente información disponible en el servidor bajo un formato particular

[2]Conseil Européen pour la Recherche Nucléaire-Consejo Europeo para la Investigación Nuclear

y que se presenta en el cliente como la vemos en la Figura 2.2. El port bien conocido reservado para el servidor es el 80. El pedido será entregado bajo un mensaje con una estructura especificada por el protocolo *http*.

El usuario que pone en acción al lado cliente del servicio para acceder a la página/información del servidor va a llamar a un programa conocido como Web browser o navegador.

La página web que quiere acceder es en realidad un archivo de formato HTML[3] que a su vez normalmente consta de objetos que pueden ser:

- archivo HTML
- imagen JPEG
- applet Java applet
- archivo de audio
- ...

Referenciados acorde con las reglas del lenguaje HTML. El formato de referencia básico está dado por una URL(Universal Resource Locator):

$$\underbrace{http}_{protocolo} : \underbrace{//www.linti.unlp.edu.ar}_{host\_name} / \underbrace{archivos/banner\_linti.jpg}_{path\_name}$$

Los objetos HTML podrán estar disponibles desde el propio navegador y para acceder a otro tipo de información el navegador acude a visores externos adaptados a cada formato. No son otra cosa que los conectores o *plug-ins*. Pero no solo con este servicio accedemos a información. Actualmente es un servicio interactivo y bidireccional a través del cual desarrollamos la mayoría de las actividades por Internet (sobre todo en estos años tan particulares bajo pandemia, 2020-2021):

- Transacciones bancarias
- Compras de todo tipo
- Correo Electrónico
- Trámites
- Reservas
- ...

Veamos en primer término lo que ocurre al acceder a la página de la UNLP.

1. Tenemos que conocer su nombre: *www.unlp.edu.ar*
2. En la barra de direcciones de nuestro navegador introducimos el url anterior.
3. El navegador podrá acordarse de accesos anteriores y nos traerá lo que tiene almacenado en nuestra computadora. Si no va a acceder a esa página.

[3]Hyper Text Markup Language

4. Como el mensaje de acceso del protocolo http se encapsula en TCP habrá que establecer una sesión TCP con el servidor ubicado en *www.unlp.edu.ar*.

5. como todo segmento TCP se encapsulará en IP y por lo tanto necesitamos conocer la dirección IP de nuestro destino.

6. Aquí se detiene el protocolo *http* dado que comúnmente desconocemos las direcciones IP de los servidores. Se llama a un "resolver" que es un procedimiento que llama a otro servicio dentro de este paradigma TCP/IP llamado DNS. Dicho servicio actúa como auxiliar de *http* dado que realiza un mapeo de nombres bajo el formato *url* a direcciones IP. Es algo parecido en ARP entre las direcciones IP y las direcciones MAC. Sobre el particular servicio nos extenderemos más adelante.

7. De una manera u otra conseguida la dirección IP del servidor se podrá establecer la sesión y enviando el mensaje *http* correspondiente nuestro navegador nos mostrará el contenido de la página de *www.unlp.edu.ar*

Como comentamos anteriormente la página tendrá un contenido acorde con las estructuras del lenguaje *HTML*. Estructuras que permiten un contenido diverso tanto como archivos, imágenes y locaciones de otras páginas web o locaciones dentro del mismo servidor.

En este caso se deberá establecer una sesión *http* por cada acceso. Teniendo en cuenta la nueva versión del protocolo, la *http1.1*, no hará falta si es la misma locación. También se han mejorado los mecanismos de acceso en pos de una mejor performance, pero no nos ocuparemos de ellos en esta instancia.

Como mecanismos auxiliares no podemos dejar de mencionar las "cookies" que dotan de algo de memoria al acceso a páginas frecuentes y nos permiten el envío de información a los servidores dándole una característica de servicio interactivo no contemplado en sus inicios y que, por otra parte, generó un vuelco importante en cuanto a frecuencia de uso de este servicio coincidente con el incremento de tráfico en Internet debido al mismo.

Acceso a una página WEB

Vamos a ver en detalle particularidades del protocolo *http* a través de capturas realizadas por el "sniffer" Wireshark al acceder a la página *www.unlp.edu.ar*

El escenario planteado es el de la Figura 2.3

En este escenario el nodo *n10* accede al nodo *n13-www* que tiene asignado para su servicio web el nombre o *URI: www.unlp.edu.ar*. Analizamos el tráfico que generó este pedido de acceso a través de la captura realizada por el *Wireshark*, *n10-nat-http-firefox.pcapng*. Como los accesos son frecuentes a esta página el navegador *Firefox* conoce la dirección IP del servidor, 163.10.0.72. Puede que el navegador no conociera la dirección IP, entonces acudiría al auxilio del servicio *DNS* para obtenerlo. El análisis de dicho servicio lo veremos ampliamente en otras secciones. En una primera visión de la captura observamos, Figura 2.4.

Queremos aprovechar para un pequeño comentario respecto de las capturas de Wireshark. Van a ver que algunas líneas o párrafos aparecen entre corchetes "[,]". Tengan en cuenta que no forman parte

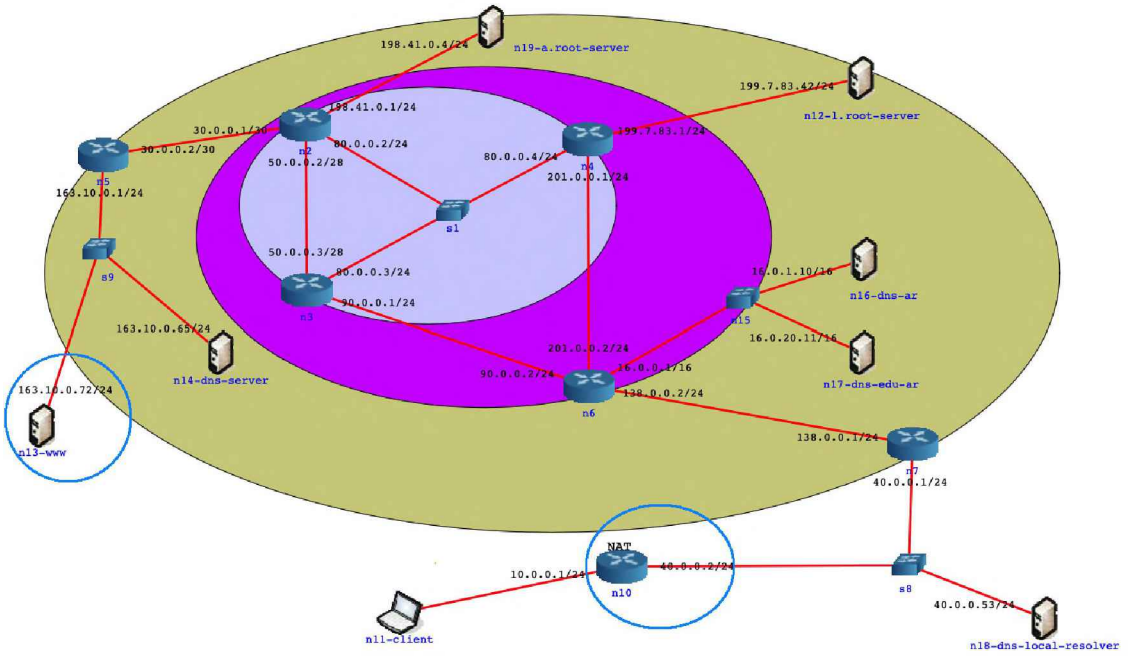

Figura 2.3: Escenario

de la trama capturada, es una interpretación/aclaración que corre por cuenta de los desarrolladores de la herramienta.

Si analizamos en detalle la trama sombreada en azul vemos que nos indica que es un mensaje del protocolo HTTP encapsulado en un segmento TCP. Comprobamos también que el servidor tiene asignado el port bien conocido 80 y, en este caso, el cliente, nodo 10, obtuvo del sistema operativo el port efímero 39796. El protocolo HTTP en su primera versión 1.0 está especificado en [BLFF96]. En este protocolo una vez que se obtenía la página se cerraba la sesión TCP.

Por lo tanto, para extraer objetos dentro de la página era necesario establecer sendas sesiones TCP. Esto generaba una no buena performance por el overhead resultante. Al poco tiempo aparece una nueva versión, la 1.1, que está especificada en [FGM+99]. La misma fue superada por:

- RFC 7230, RFC 7231, RFC 7232, RFC 7233, RFC 7234, RFC 7235.

y actualizada por:

- RFC 2817, RFC 5785, RFC 6266, RFC 6585.

Analicemos en detalle el contenido de este mensaje tomando un sector de la captura como vemos en la Figura 2.5

Este mensaje está solicitando el objeto *index2.html* a través de lo que llamamos uno de los métodos de HTTP, GET.

La primer línea de este método contiene su nombre: Request Method; Request URI, el objeto deseado, y Request Version, la versión de HTTP, en este caso 1.1. La presentación de Wireshark no muestra el fin de la línea, pero si analizamos a nivel de byte, en la última ventana veremos un $(0d0a)_{16}$,(representamos la codificación de `\r` y `\n` tal como lo hace Wireshark; siendo dígitos hexadecimales deberían estar en mayúscula), o sea retorno de carro y line feed. Herencia de la terminal virtual de Telnet. La segunda línea:

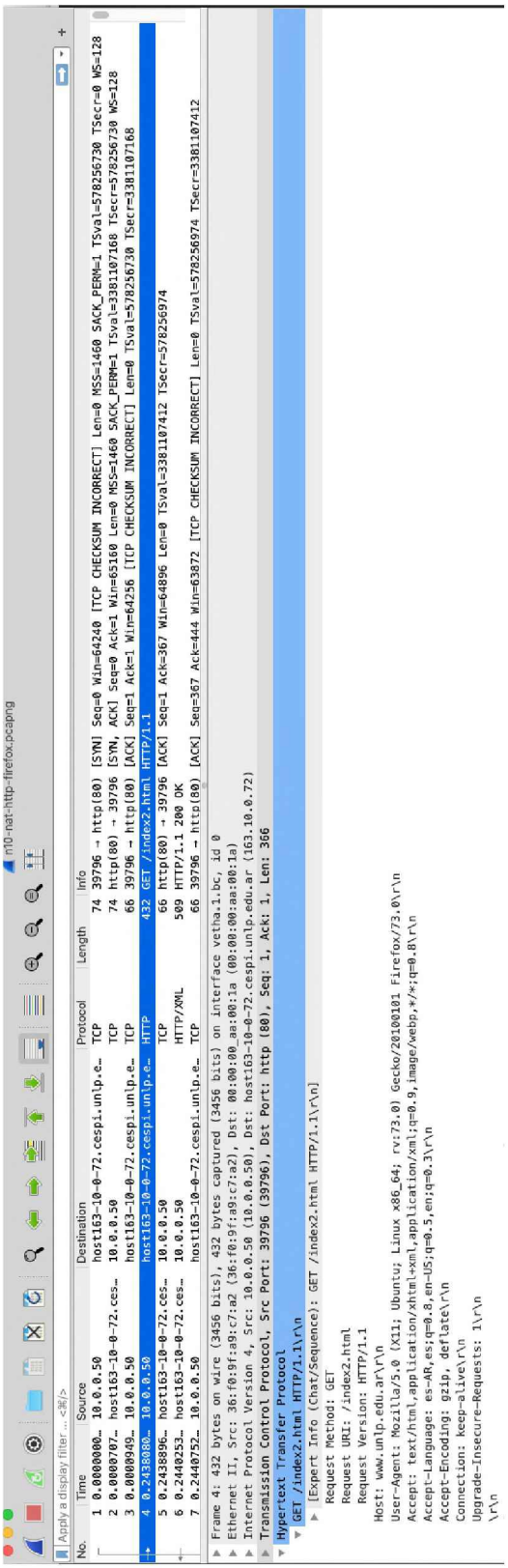

Figura 2.4: Captura acceso Web

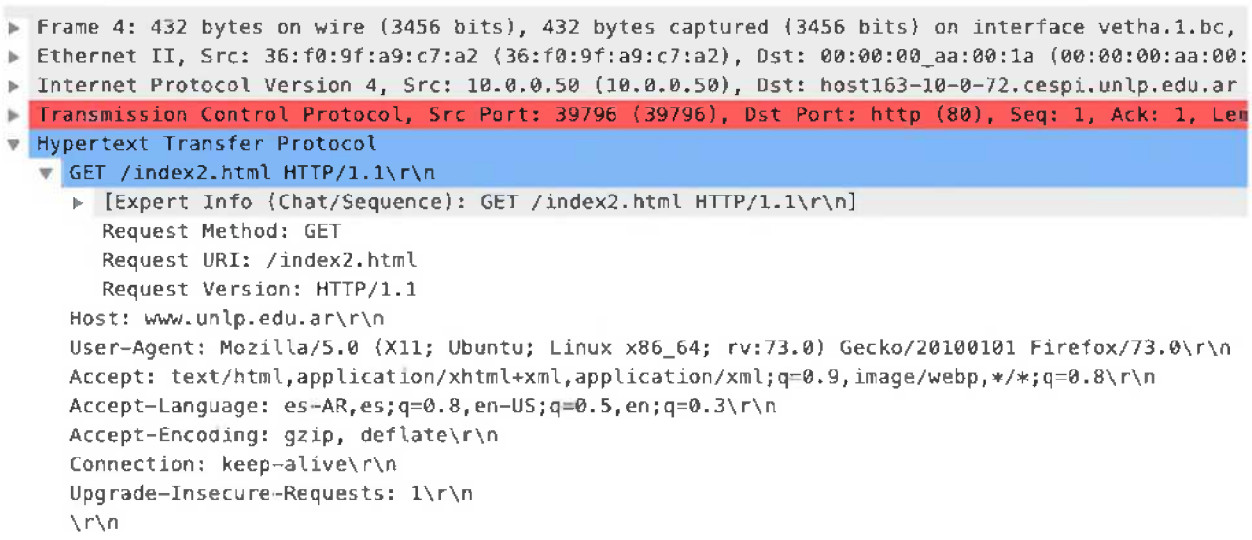

Figura 2.5: Captura GET

`Host:www.unlp.edu.ar\r\n`, el host del recurso solicitado. Aquí si Wireshark nos muestra el fin de línea.

La tercera:

`User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:73.0)`

`Gecko/20100101 Firefox/73.0\r\n`, es el agente/cliente que originó el request.

Seguimos:

`Accept: text/html,application/xhtmlxml,+`

`application/xml;q=0.9,image/webp,*/*;q=0.8\r\n`, tipos de media que se aceptan en la respuesta.

`Accept-Language: es-AR,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n`, es obvio, no?. Aclaramos, también que para la línea anterior el parámetro q es una suerte de factor de preferencia que se le da al medio o lenguaje indicado.

`Accept-Encoding: gzip, deflate\r\n`, a modo de mejorar la performance acepta que el recurso venga en cualquiera de los dos esquemas de compresión, gzip o deflate.

`Connection: keep-alive\r\n`, es un tanto obvio caracterizar Connection con "keep-alive" porque estamos en HTTP1.1 con característica persistente. Se definió para poder cerrar la conexión pese a que se estaba indicando 1.1, caracterizando a Connection con "close".

`Upgrade-Insecure-Requests: 1\r\n`, permite al agente de usuario actualizar las solicitudes a priori, inseguras por seguras antes de recuperarlas.

`\r\n`, así finaliza el mensaje. Otra herencia más de la terminal virtual de Telnet.

Retomamos la Figura 2.6 y en la trama 6 vemos el fin de la transferencia del recurso solicitado, index2.html con el detalle indicado en la Figura 2.6:

Igual que en el caso anterior la primer línea contiene tres campos del response dado por el servidor, la versión de HTTP; el código 200, todo OK como en Telnet; indicando el significado del código, OK. Si vemos la ventana de bytes veremos que finaliza con `0d oa`. Siguiendo con el análisis:

`Wed, 14 Apr 2021 11:41:50 GMT\r\n`, indica fecha y hora en que se completó la entrega del recurso.

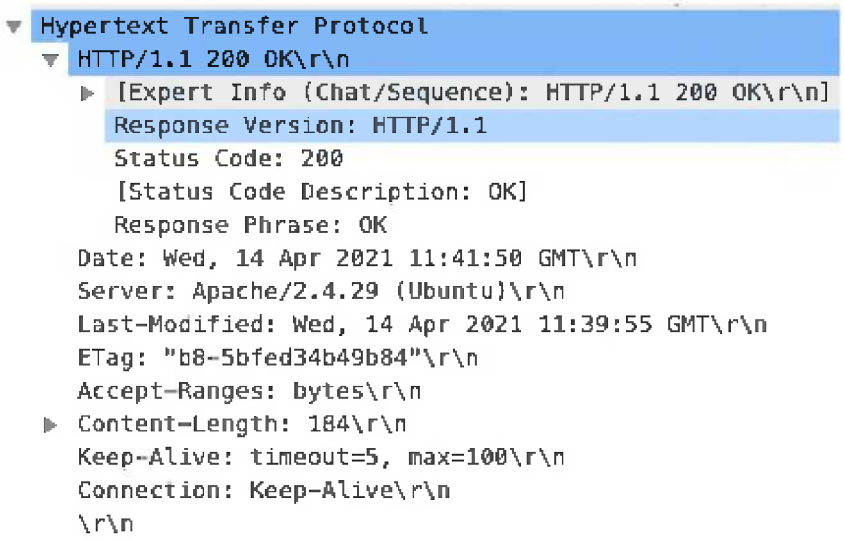

Figura 2.6: Captura Fin del GET

`Server: Apache/2.4.29 (Ubuntu)\r\n`, características del servidor.

`Last-Modified: Wed, 14 Apr 2021 11:39:55 GMT\r\n`, fecha y hora en que fue actualizado el recurso.

`ETag: "b8-5bfed34b49b84"\r\n`, tag empleado en caso de haberse requerido el recurso con algún condicionamiento. `Accept-Ranges: bytes\r\n`, la unidad del rango solicitado.

`Content-Length: 184\r\n`, no hace falta aclarar.

`Keep-Alive: timeout=5, max=100\r\n`, permite al servidor dar pistas sobre cómo se puede usar la conexión para establecer un tiempo de espera y una cantidad máxima de solicitudes.

`Connection: Keep-Alive\r\n`, ya la conocemos.

Finaliza el mensaje de response al request del GET como en el caso anterior con línea `\r\n`

Continuando con la misma captura analicemos ahora la trama 8 donde n10 hace un GET de una imagen. Mostramos en la Figura 2.7 el detalle del mensage del método GET para este caso.

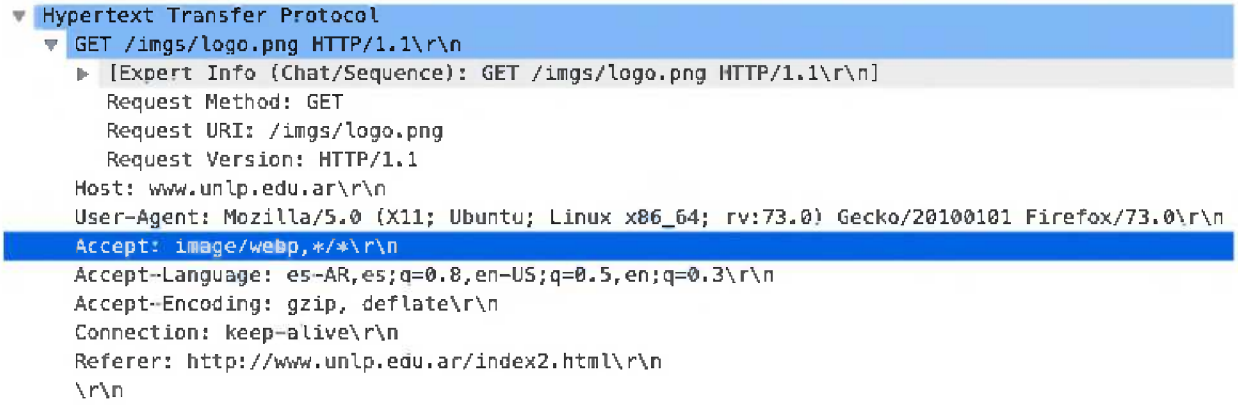

Figura 2.7: Nuevo GET

Como vemos hay campos conocidos del GET anterior. Como novedad tenemos

`Referer: http://www.unlp.edu.ar/index2.html\r\n`, que permite al cliente especificar, para beneficio del servidor, la dirección (URI) del recurso del que se obtuvo el Request-URI. Lo podemos ver como una evidencia del carácter persistente de HTTP1.1.

El recurso solicitado es un archivo png por lo que es de esperar una alta transferencia de datos. Efectivamente analizando la captura encontramos que recién en la trama 203 finaliza la descarga del archivo solicitado. Nuevamente tenemos un mensaje de fin de transferencia con código 200 previamente analizado, prácticamente igual al de la Figura 2.6.

Con el objeto de analizar otros casos vamos a la trama 284 donde encontramos un nuevo GET, similar al analizado anteriormente. Nos interesa la respuesta obtenida en la trama siguiente, 285 con el detalle del mensaje HTTP que vemos en la Figura 2.8.

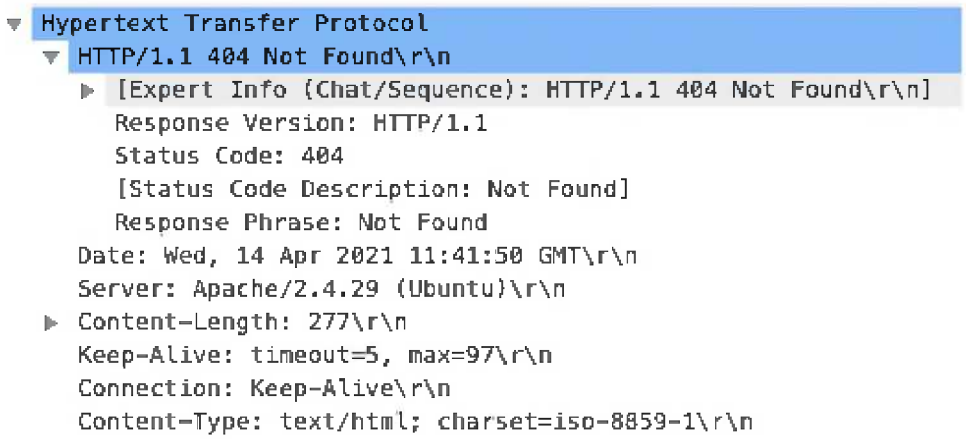

Figura 2.8: Respuesta Negativa

El recurso solicitado no se encontró y por eso el servidor envía un mensaje con código 404 que corresponde a esa situación.

Para completar el análisis de esta captura vamos a la trama 593 con un GET particular que detallamos en la Figura 2.9 Observamos unas cuantas novedades:

Figura 2.9: GET Condicional

`Upgrade-Insecure-Requests: 1\r\n`,envía una señal al servidor expresando la preferencia del cliente por una respuesta encriptada y autenticada.

`If-Modified-Since: Wed, 14 Apr 2021 11:39:55 GMT\r\n`, si el recurso no se ha modificado desde entonces, la respuesta será un 304 sin ningún contenido.

`If-None-Match: "b8-5bfed34b49b84"\r\n`,el servidor devolverá el recurso solicitado, con un estado

200, solo si no tiene un ETag que coincida con los dados. Cuando la condición falla, el servidor debe devolver el código de estado HTTP 304 (No modificado).

`Cache-Control: max-age=0\r\n`,Indica que el cliente está dispuesto a aceptar una respuesta cuya edad no sea mayor que el tiempo especificado en segundos.

Si se fijan en la trama 594 verán la respuesta del servidor con un mensaje con código 304 debido a que el recurso no fue modificado en el rango. Incluimos la Figura 2.10.

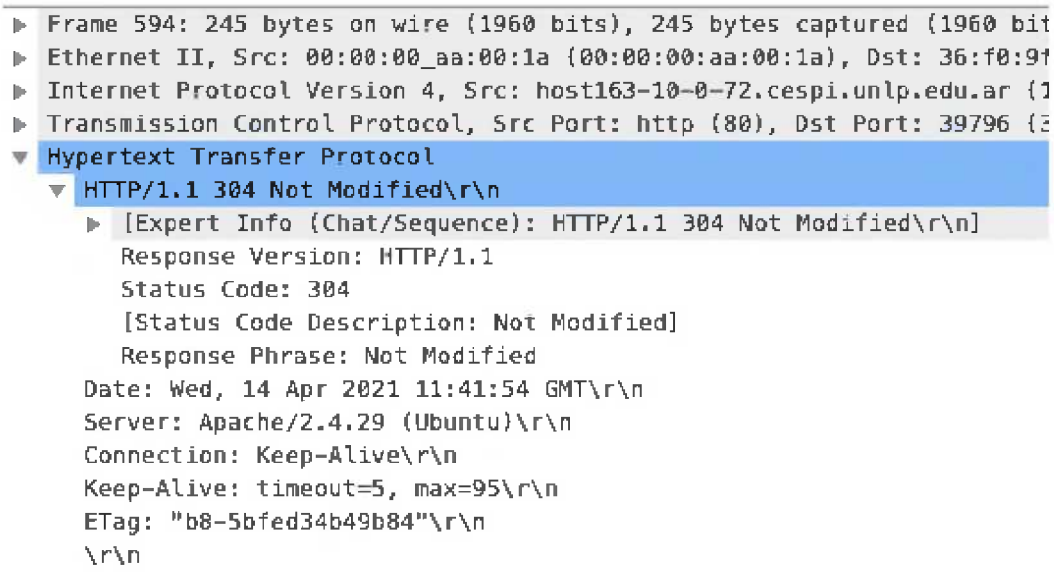

Figura 2.10: GET Condicional-Respuesta

¿Qué es eso de Cache-Control?. Pues bien, en los casos en que se accede repetidamente a una página, para evitar realizar el acceso como el que vimos, las páginas accedidas se suelen almacenar en la memoria del navegador, de modo que antes de disparar el acceso verifica si está disponible. Para evitar información no válida/vencida se almacena en la "Cache" con la fecha de acceso.

A modo de resumen nos parece oportuno presentarles la estructura general de un mensaje HTTP, Figura 2.11:

Completamos algunos detalles con un cuadro del resto de los métodos de HTTP:

HEAD: Similar a GET, pero sólo pide las cabeceras HTTP.

POST: Envía datos a una URL para que el recurso en esa URI los gestione.

PUT: Pone un recurso en la dirección especificada en la URL. Exactamente en esa dirección. Si no existe, lo crea, si existe lo reemplaza.

DELETE: Elimina el documento referenciado en la URL.

TRACE: Rastrea los intermediarios por los que pasa la petición.

OPTIONS: Averigua los métodos que soporta el servidor.

Hemos decidido comentar en forma extensa capturas correspondientes al protocolo HTTP 1.1 por cuanto es ampliamente soportado por la mayoría de los servidores. De todos modos debemos mencionar que existe una nueva versión del protocolo. Está disponible HTTP 2 [BPT15]. En pocas palabras

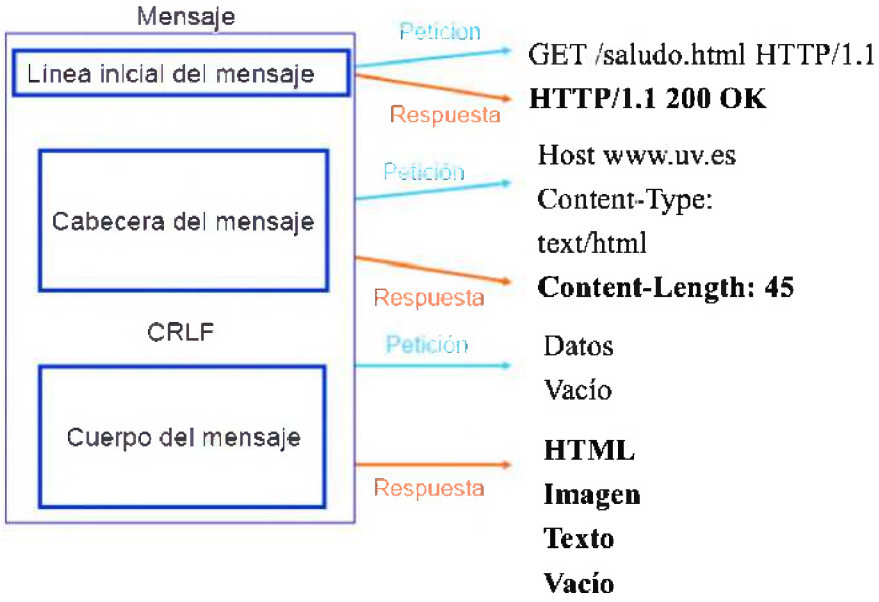

Figura 2.11: Mensaje HTTP

presenta un menor overhead con compresión de los encabezados, requests múltiples en paralelo y un mayor grado de seguridad.

Para finalizar, si se requiere plena seguridad en el acceso entonces se plantea este modelo, Figura 2.12: Como la misma RFC referenciada lo dice HTTPS [Res00] no es otra cosa que HTTP montado

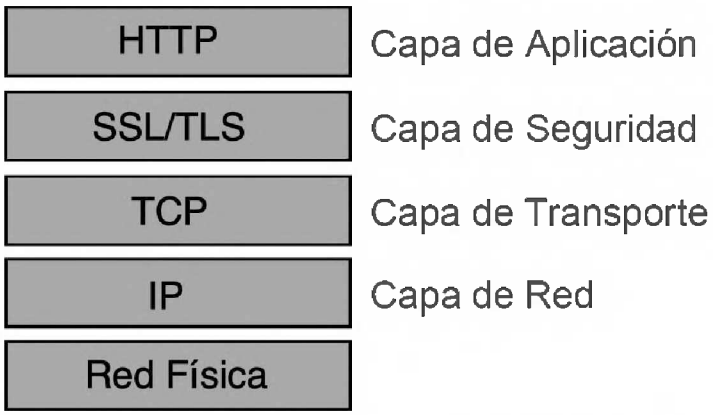

Figura 2.12: HTTPS

sobre SSL[4] o en su actualización por TLS[5]. El port bien conocido dado por el IANA es el 443.

[4] Secure Socket Layer
[5] Transport Layer Security

REFERENCIAS

[BLFF96] T. Berners-Lee, R. Fielding y H. Frystyk. «Hypertext transfer protocol – HTTP/1.0», 1996.

[BPT15] Mike Belshe, Roberto Peon y Martin Thomson. «Hypertext transfer protocol version 2 (HTTP/2)», mayo de 2015.

[FGM+99] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach y Tim Berners-Lee. «Hypertext transfer protocol - HTTP/1.1», jun de 1999.

[Res00] Eric Rescorla. «HTTP over TLS», mayo de 2000.