Universidad Diego Portales
Facultad de Ingeniería y Ciencias

Informe de tarea 2

Taller de Redes y Servicios

Autores:
Estudiantes: Ezequiel Morales ; Francisco Piñera
RUT: 22602790-4 ; 21667648-3

Carrera: Ingeniería Civil en Informática y Telecomunicaciones
Email 1: ezequiel.morales@mail.udp.cl
Email 2: francisco.pinera@mail.udp.cl

Segundo Semestre - 28 de octubre de 2025

Índice

1. Introducción

Los objetivos planteados para la realización de esta tarea son la creación de tráfico en la red basándose en un protocolo en específico, estudiar y analizar el origen y funcionamiento de ese protocolo y también por su parte analizar el tráfico generado de ese protocolo.

Se plantea elegir un protocolo de red en específico, aquel protocolo estará ligado a dos aplicaciones de software orientadas a un cliente y un servidor respectivamente. Se buscará crear una conexión entre ambas aplicaciones la cual permita generar un tráfico de red controlado que contenga el protocolo de red previamente escogido. Con ello se busca generar solicitudes y respuestas entre el cliente y el servidor que al mismo tiempo serán capturadas utilizando Wireshark para Linux.

Al final del proceso se espera analizar todo ese tráfico generado y realizar un análisis a detalle sobre la relación que existe entre las funciones del protocolo y la evidencia obtenida. Se espera que la investigación del protocolo sea lo suficientemente sólida como para lograr entender de manera clara sus capacidades y funcionamiento.

2. Metodología

Para el desarrollo de este taller se debe trabajar con un protocolo de red específico, y a su vez ese protocolo estará asociado a dos aplicaciones de software que actuarán como un cliente y un servidor respectivamente. El protocolo con el que se trabajará para esta ocasión es el protocolo **HTTP**. Y por otro lado, las aplicaciones asociadas a este protocolo son **Nginx** y **Wget**. Respectivamente Nginx es la aplicación que actuará como el servidor de la conexión, mientras que wget actuará como el cliente de esta conexión.

Para establecer esta conexión se deberá trabajar con Linux y Docker. Adicionalmente se utilizará Wireshark para capturar el tráfico generado entre cliente y servidor.

En resumen, las herramientas a utilizar para este taller son:

- Protocolo de red: HTTP
- SO: Linux Mint Cinnamon 22.1 64-bit
- Servidor: Nginx (latest)
- Cliente: Wget (latest)
- Sistema de imágenes: Docker
- Aplicación de captura: Wireshark for Linux

3. Desarrollo

CAPÍTULO I: El protocolo HTTP

3.1. Introducción al protocolo

El protocolo HTTP o en sus siglas en inglés *Hypertext Transfer Protocol*, es un protocolo de la capa de aplicación desarrollado por *Tim Berners-Lee* entre los años 1989 y 1991. Durante los primeros años de la creación de la World Wide Web, también se trajo consigo la creación del protocolo HTTP. La necesidad surgió cuando en uno de los cuatro bloques fundamentales que componen este sistema, se requería diseñar un protocolo capaz de intercambiar en la red con facilidad documentos en formato de hipertexto, o también conocidos como HTML.

El protocolo diseñado para cumplir esa tarea fue el protocolo HTTP. La mayor gracia de HTTP es que se trata de un protocolo muy sencillo tanto a nivel de código como a nivel de comprensión, y también que su tarea fundamental es simplificar la comunicación entre páginas web y servidores. Es por ello que desde su creación hasta más de 20 años después,

solo se fue actualizando hasta llegar a la versión más nueva que es la que se continúa utilizando actualmente, y la que también trabaja no solo con HTML, sino también con diversos tipos de archivos incluyendo imágenes y modelos 3D.

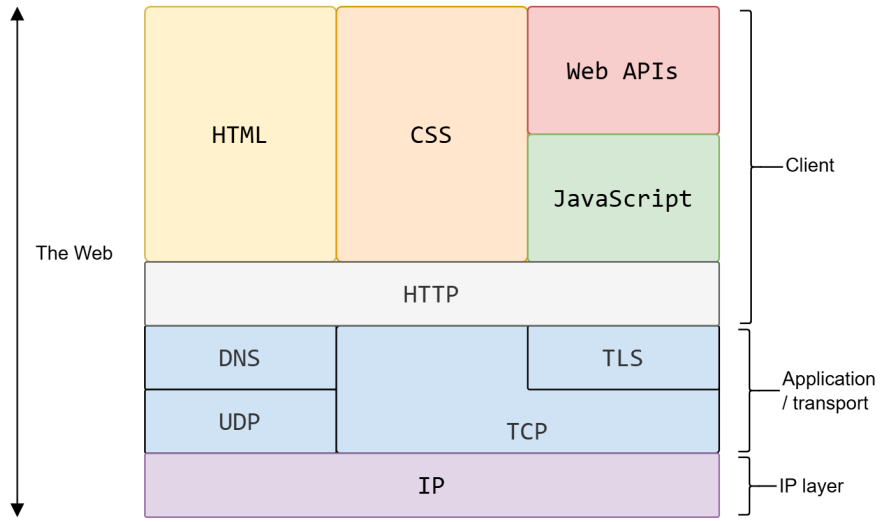

Figura 1: HTTP

A medida que la web fue creciendo, HTTP pasó de ser un mecanismo muy simple de petición-respuesta a un estándar universal para todo tipo de servicios en Internet. Esto explica por qué en esta tarea basta con levantar un contenedor web y un cliente de línea de comandos para poder observar de inmediato las tramas que genera: el protocolo está pensado justamente para ser legible y depurable.

3.2. Funcionamiento del protocolo

El protocolo HTTP basa su funcionamiento en un sistema de peticiones entre cliente y servidor, es decir, que cuando por ejemplo se escribe una URL en el navegador, este envía una solicitud HTTP al servidor que aloja la página web, para luego procesar esa solicitud. Luego de eso, se envía de vuelta una respuesta HTTP con el contenido de la página web, para que después el propio navegador la interprete y la muestre. Un intercambio mínimo se ve así:

```
   GET / HTTP/1.1
Host: servidor-ejemplo
```

y el servidor responde con algo del estilo:

```
   HTTP/1.1 200 OK
Content-Type: text/html
```

más el contenido solicitado. Esto es exactamente lo que ocurre en la tarea cuando el cliente `wget` le pide a Nginx el `index.html`.

Las peticiones HTTP se componen de un pequeño grupo de tipos de mensajes los cuales realizan una acción específica. Estos tipos de mensajes se les puede llamar de mejor manera como métodos y los más importantes son cuatro en total.

- **GET:** Se encarga de recuperar datos que se encuentran alojados en el servidor.
- **POST:** Envía datos al servidor para que sean procesados.
- **PUT:** Se utiliza para modificar algún recurso existente del servidor, y en caso de que no exista, lo crea.
- **DELETE:** Se encarga de comunicarse con el servidor para eliminar datos que estén alojados en él, como registros de una base de datos por ejemplo.

Por otro lado, asociado a los métodos HTTP se encuentran también los **mensajes HTTP.** Un mensaje HTTP es un código de tres dígitos estandarizado que arroja el servidor como respuesta cuando tiene la necesidad de comunicar al resto si una operación fue exitosa, fallida o alguna otra situación en específico.

Actualmente existe un gran número de mensajes HTTP de los cuales la gran mayoría están destinados a entregar mensajes específicos para cada operación y/o situación que pueda ocurrir dentro de una página web, es por ello que su alcance para la comunicación es tremendo.

No obstante, para efectos generales los mensajes HTTP más importantes son aquellos que se utilizan para operaciones fundamentales de una página web.

- **HTTP 200 OK:** Mensaje por defecto para indicar que se logró acceder exitosamente a un recurso. Indica que la solicitud GET fue exitosa.
- **HTTP 404 NOT FOUND:** Indica que el recurso del servidor al cual se estaba intentando acceder no pudo ser encontrado.
- **HTTP 500 INTERNAL SERVER ERROR:** Comunicación de que la solicitud no puede ser realizada debido a un fallo interno del servidor.
- **HTTP 302 FOUND:** Indica que ante la solicitud realizada es necesario redirigir al usuario a otra URL.

3.3. Detalles del software asociado al servicio (SERVIDOR)

El programa alojado en contenedor de Docker y el cual actúa como servidor es **NGINX.** Este programa es un software muy popular de servidor web HTTP, el cual es de código abierto y fue lanzado oficialmente en 2004. Actualmente es el más popular de su clase y una de las imágenes Docker más populares también.

En esencia nginx es un servidor de páginas web, con multitud de funciones entre ellas servir archivos estáticos como HTML. A diferencia de otros servidores web, una de las mayores ventajas de nginx es que funciona con una arquitectura asíncrona y controlada por eventos, lo que le permite trabajar con miles de conexiones al mismo tiempo sin caerse o perder rendimiento (escalabilidad).

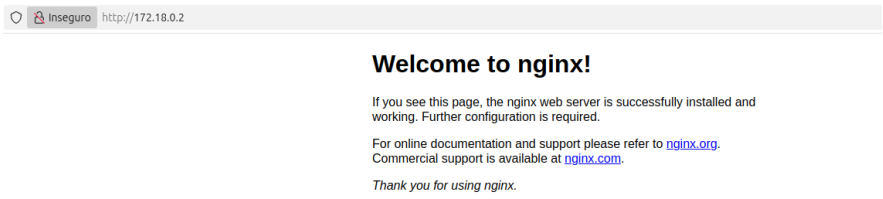

Figura 2: Página Nginx

3.4. Detalles del software asociado al servicio (CLIENTE)

Wget es una herramienta informática que cumple la función de recuperar contenido y archivos de varios servidores web. Como el propio nombre lo sugiere, wget hace una referencia directa a la world wide web, el protocolo HTTP y el método GET, por lo que su rol como cliente está claramente definido.

Su instalación y uso es sumamente sencillo, por lo que se puede utilizar como un comando directo para obtener archivos directamente desde la web.

En este contexto el uso de `wget` tiene dos ventajas muy concretas. La primera es que permite generar tráfico HTTP *a demanda*, es decir, justo después de que el contenedor de `nginx:latest` está arriba y atendiendo en la red de Docker; así se sabe exactamente cuándo comienza la transferencia y se puede detener la captura en Wireshark al terminar la ráfaga de paquetes. La segunda ventaja es que `wget` muestra en la terminal el resultado de la operación (descarga correcta, código de respuesta, tamaño del archivo), por lo que es muy fácil relacionar lo que se ve en consola con los paquetes capturados.

Para lanzar el cliente dentro del mismo entorno dockerizado se creó un contenedor **temporal** que solo ejecuta `wget` en la red `red-tarea2`. En el laboratorio se usó una imagen mínima (`alpine:latest`) solo como base, quedando el comando así:

```
sudo docker run --rm --network red-tarea2 alpine:latest \
    wget http://nginx-tarea2
```

Este comando lo único que hace es usar `wget` como cliente HTTP contra el servidor Nginx que ya está levantado en otro contenedor, enviar una solicitud `GET` al puerto 80 y, una vez recibida la respuesta con el `index.html`, eliminar el contenedor temporal (`--rm`). Desde el punto de vista del tráfico, esto genera el establecimiento TCP, la petición HTTP y la respuesta `200 OK` que se observaron en Wireshark.

3.5. Levantamiento contenedores cliente/servidor y realizar conexión entre ellos.

Para la realización de este proceso solo se utilizó la terminal de Linux para crear y descargar todo lo necesario.

- Primero abrimos una terminal de Linux y actualizamos paquetes con el comando **sudo apt update**.
- Luego de forma directa descargamos la imagen del servidor Nginx desde la terminal con el comando **sudo docker pull nginx:latest**. Yo no lo descargo porque ya lo tengo.

Figura 3: Descarga de Nginx

- El siguiente paso es crear una red virtual la cual se utilizará para separar la conexión entre el cliente y el servidor, ya que de otro modo, la conexión entre ambos no podría funcionar según lo esperado, teniendo paquetes HTTP no deseados por ejemplo. Esta red virtual debe tener un nombre fácil de recordar para así poder invocarla después. Esta red lleva el nombre de **red-tarea2** y se crea con el comando **sudo docker network create red-tarea2**. A mí me da error porque ya la tengo creada.

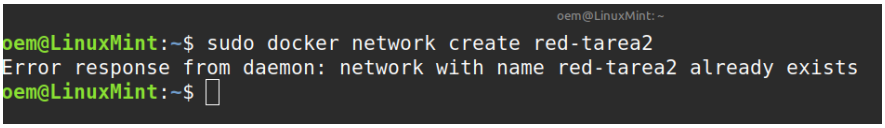

Figura 4: Creación de red

- Ahora es cuando se ejecuta el contenedor del servidor Nginx, para ello es necesario utilizar la red creada previamente y definir un nombre característico. En este caso se decidió optar por el nombre **nginx-tarea2**. Ahora incluimos ambos nombres dentro del comando quedando de la forma **sudo docker run -d –name nginx-tarea2 –network red-tarea2 nginx:latest**. A mí me da error porque ya lo tengo. Con este comando el servidor arranca y queda corriendo en segundo plano, y por otro lado si se desea arrancarlo o frenarlo más tarde se utilizan los comandos **sudo docker start/stop nginx-tarea2**.

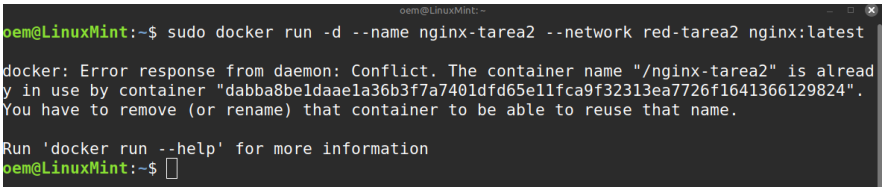

Figura 5: Creación de contenedor de servidor

- Ahora, si revisamos ciertos comandos podremos verificar el funcionamiento de todo, y si por ejemplo, ejecutamos **sudo docker ps** veremos que el contenedor nginx se está ejecutando.

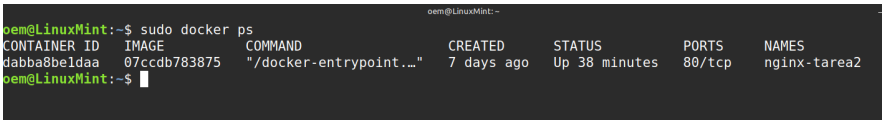

Figura 6: Lista de contenedores

Por otro lado si revisamos con el comando **sudo docker network ls** veremos la lista de redes que hay actualmente, donde aparecerá un id y el nombre al que pertenece. Este dato del id de nuestra red virtual será de utilidad cuando toque analizar tráfico en Wireshark.

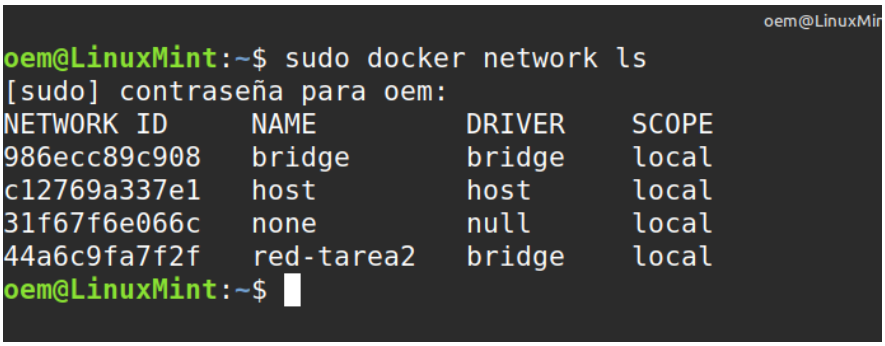

Figura 7: Lista de redes

- Ahora lo último que se debe hacer es ejecutar el cliente de Wget. Esta ejecución generará el tráfico que necesitamos. Lo que hace este comando es utilizar una imagen ligera llamada alpine junto con wget con el objetivo de iniciar el cliente de wget, conectarlo a la red virtual, comunicarse con el servidor nginx, obtener archivos y auto eliminarse una vez finalizado el proceso. Este comando se puede ejecutar cada vez que sea necesario aunque solo con una vez basta. Para utilizarlo hay que ingresar los nombres de nuestro contenedor y red creados previamente quedando todo de esta forma **sudo docker run –rm –network red-tarea2 alpine:latest wget http://nginx-tarea2**.

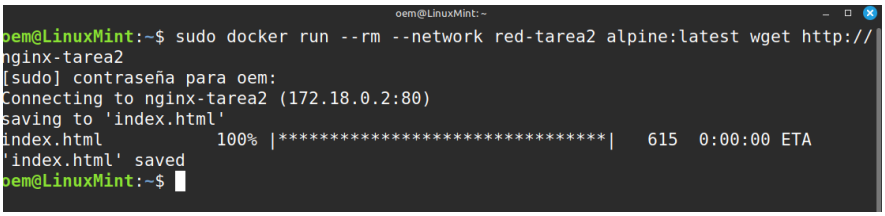

Figura 8: Tráfico generado

En esta etapa es importante destacar que la conexión se comprobó de manera efectiva porque el cliente recibió el archivo por HTTP y porque en Wireshark se observaron las tramas correspondientes. Esto demuestra que la configuración de red, el despliegue de los contenedores y el uso de las herramientas fueron correctos.

CAPÍTULO II: Análisis de tráfico

3.6. Análisis de tráfico

Realizar el análisis de tráfico es sumamente sencillo, ya que los pasos importantes para poder generar el tráfico que se busca capturar ya fue realizado previamente.

Primero se debe abrir Wireshark desde la terminal y observar la lista de redes, dentro de esa lista no se debe utilizar la red por defecto que viene de la red del equipo, sino la red virtual creada previamente. Esta red virtual aparecerá según el ID asociado que aparece cuando utilizamos el comando **sudo docker network ls**. Este id es el **br-44a6c9fa7f2f**, y ese id es el que aparece como red en la lista de Wireshark.

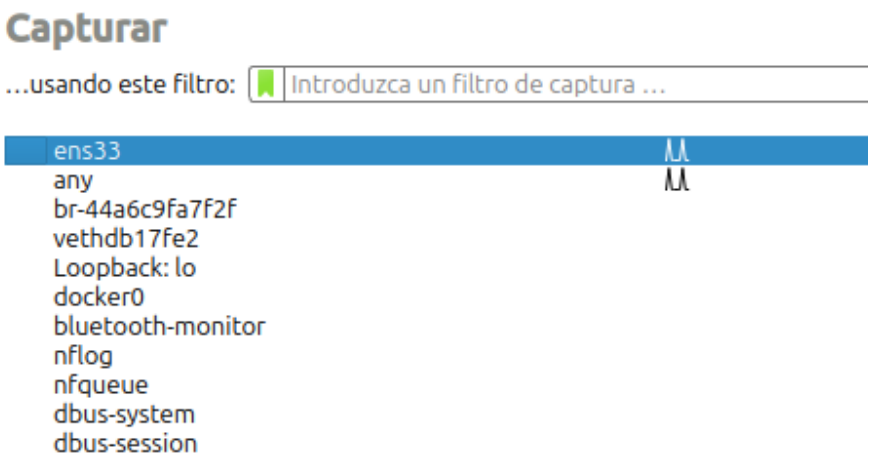

Figura 9: Lista de redes en Wireshark

Es por ello que se selecciona esa red y como paso siguiente nos dirigimos a otra terminal de Linux, y en esa terminal de Linux ejecutamos el comando que genera el tráfico con los mismos parámetros de antes. Una vez que aparece el mensaje que indica que el archivo se descargó correctamente, volvemos a Wireshark y observamos la captura. En la captura corriendo, aparecerá una ráfaga de mensajes TCP juntos, y cuando termina esa ráfaga es cuando ya es momento de detener la captura pues ya se capturaron todos los paquetes necesarios.

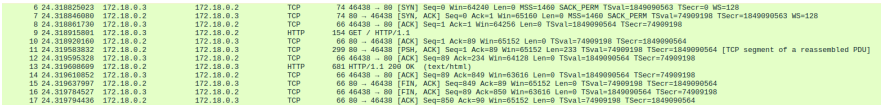

Figura 10: Ráfaga TCP y HTTP

Con la captura ya guardada, es posible empezar a realizar el análisis. Lo más básico es observar la ráfaga obtenida, y dentro de ella contamos con un total de 12 paquetes. Lo siguiente es filtrar por el protocolo HTTP, obteniendo así solamente dos paquetes los cuales corresponden a la comunicación entre cliente y servidor.

Leyendo las direcciones IP nos encontramos con dos IP que al principio parecen desconocidas, pero si inspeccionamos la red con el comando **sudo docker network inspect red-tarea2**...

```
oem@LinuxMint:~$ sudo docker network inspect red-tarea2
[
    {
        "Name": "red-tarea2",
        "Id": "44a6c9fa7f2f702c554bb7aeeac34848cfafbdd325e0f163fde024510a90e45f",
        "Created": "2025-10-18T20:33:37.739357332-03:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "dabba8be1daae1a36b3f7a7401dfd65e11fca9f32313ea7726f1641366129824": {
                "Name": "nginx-tarea2",
                "EndpointID": "ec4b4525ea3fb7ddc471b39cf2ff4cb723cd8957a32658987c3a759ede84caf1",
                "MacAddress": "fa:3b:70:1f:d0:e7",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Figura 11: Inspección de red

Se puede observar con facilidad que la dirección IP asociada al contenedor de Nginx es la IP **172.18.0.2**, dejando así que la otra IP **172.18.0.3** es la que corresponde al cliente wget.

Observando más a detalle ambos paquetes, en el primero de ellos, se pueden observar aspectos como por ejemplo de que efectivamente, se envió una solicitud de tipo request con el **método GET** hacia la dirección IP de Nginx, y junto con eso se observa que en los puertos de origen y destino, se encuentra que el de destino es el TCP 80, por lo que esta petición se está enviando hacia el puerto correcto. Inclusive más abajo se puede observar como el propio mensaje avisa que la respuesta a esa solicitud se encuentra en el paquete número 13.

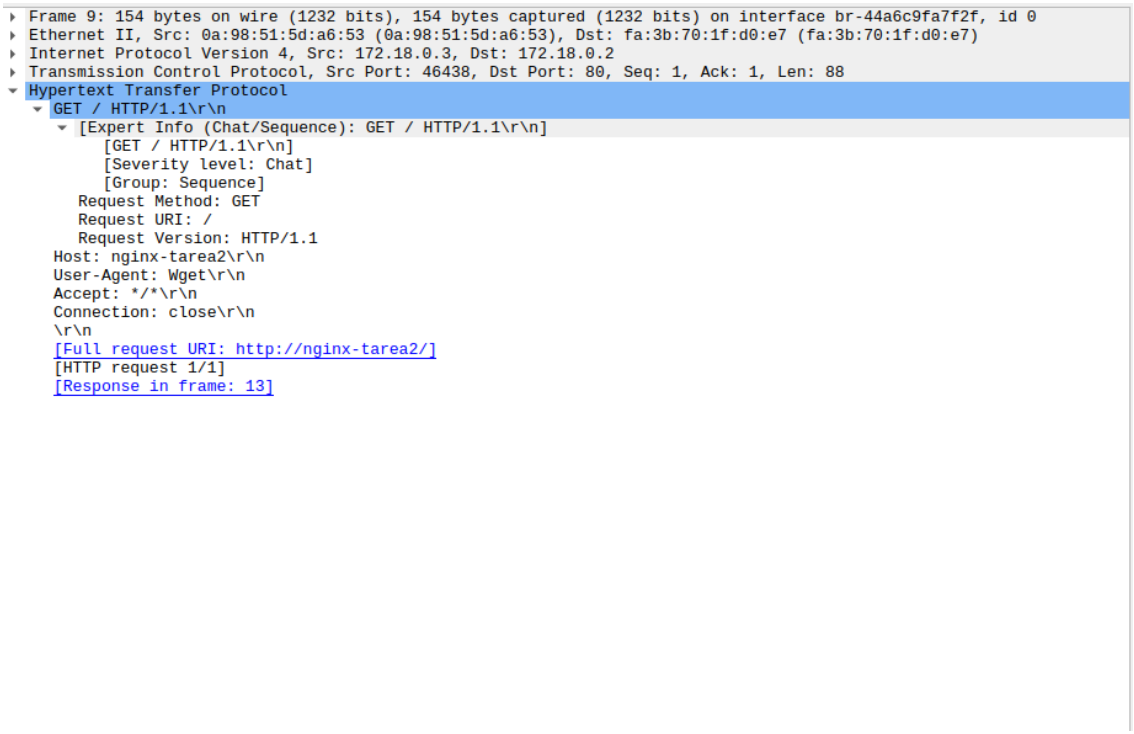

Figura 12: Inspección paquete 9

Por otro lado, al observar la respuesta del servidor, se observan detalles como el mensaje HTTP 200 que indica la operación exitosa, o los puertos indicando que desde el puerto 80 se está enviando la información hacia el puerto del cliente, o el tipo de contenido el cual es HTML, o la URL asociada la cual es la que se definió previamente para el contenedor de Nginx.

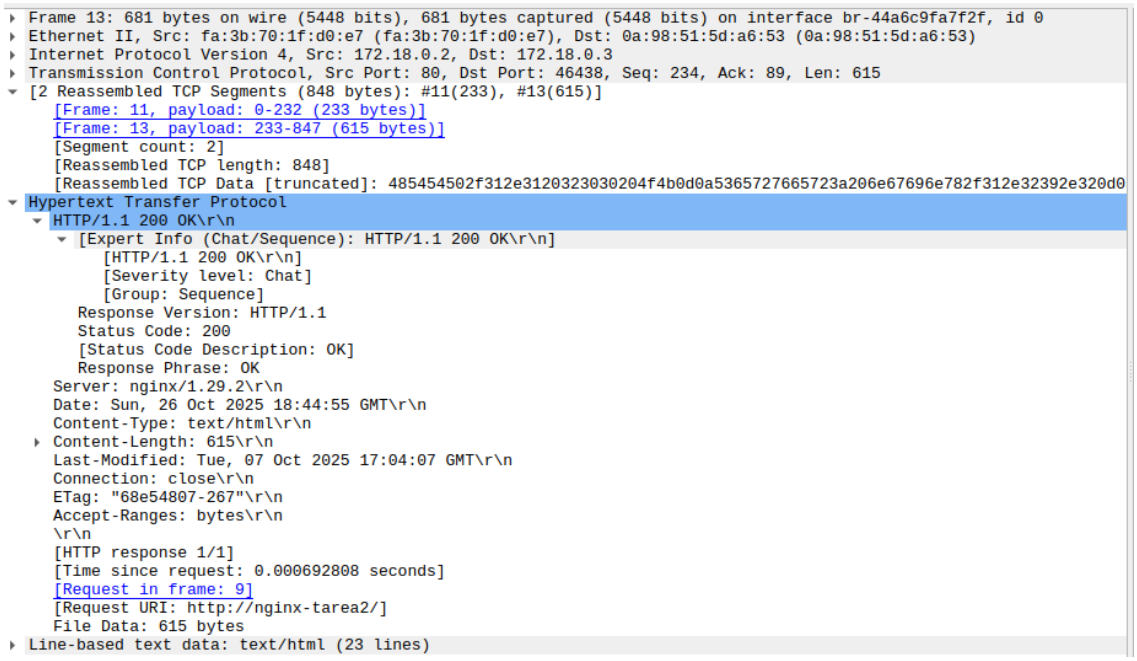

Figura 13: Inspección paquete 13

3.7. Detalle en el tráfico capturado

Como ya se había mencionado antes, al ejecutar el comando referente al contenedor de wget, este iniciaba una comunicación con el servidor y como resultado se generaba tráfico. Este tráfico generado son archivos que el cliente **Wget** "obtiene" por decirlo de cierta forma ya que este cliente funciona con un método GET, por lo que en perspectiva si el cliente solicita la obtención de un recurso, el servidor debe proveerle de ese recurso siempre y cuando este disponible.

Para este caso en específico, lo que viene siendo referente al tráfico no es nada más ni nada menos que un archivo de tipo `index.html`. Este archivo HTML es lo que le entrega el servidor de Nginx al cliente, y este archivo en particular es un archivo estático que sirve la página inicial de bienvenida de Nginx, la cual aparece siempre que no se le haga ninguna modificación. Es más, cuando se revisa en Wireshark los datos de este `index.html`, se puede ver en el contenido del paquete las etiquetas que contienen el código de la página de Nginx.

Si se diera la ocasión el `index.html` podría ser diferente o habría un CSS asociado pero solo hay un archivo que se puede rescatar por lo que el tráfico total es de solo unos pocos bytes.

3.8. Suposiciones

Para completar el análisis solicitado es necesario considerar qué ocurriría si las condiciones de la comunicación cliente–servidor cambiaran o si alguno de los elementos de la trama HTTP fuera alterado. A continuación se exponen las suposiciones más relevantes para este escenario.

En primer lugar, si el cliente `wget` en vez de realizar una petición `GET` válida enviara una cabecera incompleta o con un método no soportado por Nginx, el servidor igualmente recibiría el flujo TCP pero respondería con un código distinto a `200 OK`. Dependiendo del error, podría contestar con `400 Bad Request` o `405 Method Not Allowed`. En la captura de Wireshark esto se vería como una respuesta HTTP con otro código de estado y con un cuerpo de mensaje diferente. La herramienta seguiría siendo capaz de decodificar la trama, por lo que el analizador de tráfico funcionaría igual, pero la interpretación cambiaría porque ya no habría transferencia de `index.html` sino un mensaje de error.

En segundo lugar, si se modificara el contenido que sirve Nginx (por ejemplo reemplazando el `index.html` por uno más grande o con recursos adicionales), la consecuencia directa sería un aumento en el tamaño de la respuesta y, por lo tanto, de la cantidad de segmentos TCP capturados. Aun así, la estructura general del intercambio seguiría siendo la misma: una petición de un solo paquete y una respuesta de uno o más paquetes. Esto demuestra que el servicio es sensible al contenido alojado en el servidor, pero no cambia la lógica del protocolo.

Otra suposición importante es la de pérdida o retraso dentro de la red virtual de Docker. Si alguno de los paquetes de la respuesta no llegara, el cliente intentaría retransmitir y en Wireshark se observarían duplicados o *retransmissions* TCP. Desde el punto de vista del funcionamiento del software, `wget` probablemente demoraría más tiempo en terminar la descarga o finalmente declararía error, pero la sesión seguiría identificándose como HTTP porque la cabecera inicial ya fue vista por el analizador.

Finalmente, si un atacante o un tercer contenedor inyectara tráfico HTTP dentro de la misma red `red-tarea2`, en la captura aparecerían conversaciones adicionales usando las mismas direcciones IP de la subred 172.18.0.0/16. Esto podría llevar a interpretar mal cuál fue el flujo realmente generado por la prueba. Por eso en la metodología se insistió en usar una red dedicada de Docker y en ejecutar el cliente sólo en el momento de la captura: de esa forma se puede suponer que todo el tráfico visto durante ese intervalo pertenece efectivamente al experimento descrito.

4. Conclusión

La tarea permitió implementar y documentar una arquitectura cliente–servidor basada en HTTP utilizando contenedores Docker conectados en una red virtual aislada. A partir de esta configuración fue posible generar tráfico controlado, capturarlo con Wireshark e identificar con precisión el intercambio típico de este protocolo: una solicitud `GET` emitida por el

cliente y una respuesta `200 OK` entregada por el servidor Nginx junto con el recurso solicitado (`index.html`).

El uso de Docker resultó adecuado porque separó de forma clara los roles de cliente y servidor y, al mismo tiempo, facilitó la elección de la interfaz correcta en Wireshark, evitando la mezcla con tráfico del host. La selección de `wget` como cliente también fue pertinente, ya que permitió disparar la transferencia exactamente en el momento deseado y relacionar el resultado mostrado en la terminal con los paquetes observados en la captura.

El análisis de los paquetes confirmó que el servicio respondió en el puerto estándar (80/TCP), que las direcciones IP asignadas a los contenedores coincidieron con las reportadas por `docker network inspect` y que el contenido entregado por Nginx corresponde al archivo de bienvenida por defecto. Las suposiciones planteadas muestran, además, que pequeñas variaciones en la petición o en el contenido servido se reflejarían de inmediato en la traza, lo que convierte a este montaje en una buena base para experiencias posteriores de modificación o inyección de tráfico.

En conclusión, se cumplieron los objetivos planteados al inicio: se escogió un protocolo, se desplegaron las herramientas necesarias para operarlo, se generó tráfico real y se estableció la relación entre lo observado en la captura y el funcionamiento teórico de HTTP. Esto deja preparado el entorno para trabajos siguientes donde se profundice en fallos, medidas de seguridad o comparación con otros protocolos de aplicación.

5. Bibliografía

- https://keepcoding.io/blog/que-es-el-protocolo-http/
- https://concepto.de/http/
- https://developer.mozilla.org/es/docs/Web/HTTP/Guides/Overview
- https://www.hostinger.com/es/tutoriales/que-es-nginx
- https://nginx.org/en/
- https://www.hostinger.com/es/tutoriales/usar-comando-wget