Presentación

¿Sabías que muchas empresas de Internet como Twitter, Facebook, Flickr o Wikipedia tienen una API – un conjunto de definiciones y protocolos para que dos aplicaciones puedan interactuar – desde la que podemos descargar datos? Por ejemplo, datos sobre qué usuarios están twitteando o sobre qué temas se están twitteando.

En esta actividad se te propone utilizar una web API con el objetivo puesto en el análisis de protocolos de red. Es decir, no nos fijaremos en el tratamiento de los datos, sino que usaremos una herramienta de software libre muy popular entre los técnicos de red (`Wireshark`). Esta herramienta nos permitirá detectar el tráfico (o hacer *traffic sniffing* en inglés) y poder así analizar los paquetes que pasan por la interfaz de red de nuestro ordenador. A su vez, os planteamos una serie de preguntas teóricas sobre los contenidos del reto.

Competencias

En esta PEC se trabajan las siguientes competencias del Grado de Ciencia de Datos Aplicada:

- Que los estudiantes hayan demostrado poseer y comprender conocimientos en un área de estudio que parte de la base de la educación secundaria general, y se suele encontrar a un nivel que, si bien se apoya en libros de texto avanzados, incluye también algunos aspectos que implican conocimientos procedentes de la vanguardia de su campo de estudio.
- Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.
- Uso y aplicación de las TIC en el ámbito académico y profesional.
- Administrar y gestionar los sistemas operativos y de comunicaciones de los componentes de una red de ordenadores.

Objetivos

Los objetivos específicos de esta PEC son:

- Ser capaz de comprender las funciones asociadas a las capas superiores del modelo OSI (transporte y aplicación) y diferenciarlas de las destacadas en el reto anterior.
- Entender la importancia de las redes como herramienta para obtener datos.
- Conocer las principales aplicaciones orientadas a Internet.
- Conseguir analizar un flujo de datos y encontrar los elementos estudiados en los tres primeros retos.
- Dominar el uso de una herramienta para analizar protocolos de red.

Descripción de la PEC a realizar

Esta PEC hará que pongas en práctica los conocimientos y procedimientos asociados a este reto. Partiremos del mismo navegador que utilizamos cada día para navegar por Internet y teclearemos una dirección web para analizar qué pasa en la red todas las veces que ejecutamos una operación tan básica y cotidiana. Seguidamente, haremos lo mismo accediendo a una API de la que podríamos decidir extraer datos. En esta PEC no nos centraremos en la extracción de datos, que ya se tratará en otros cursos/asignaturas, sino que sólo observaremos y analizaremos lo que pasa en la red.

En cuanto a los ejercicios de esta PEC, se combinan preguntas teóricas referentes a los principales conceptos de este reto, junto con cuestiones referentes al análisis de red a través de una herramienta de software libre: Wireshark.

NOTA: El navegador `Google Chrome` suele utilizar un protocolo particular llamado QUIC que puede dificultarnos la visión de los conceptos a tratar. Por ello, se recomienda usar otros navegadores (p. ej. Mozilla Firefox). En caso de que queráis usar `Google Chrome`, os dejamos **este enlace** que explica cómo deshabilitar QUIC.

Recursos

Recursos Básicos

- Las redes como medio para obtener datos

- Las capas inferiores del modelo OSI
- El nivel de red
- El nivel de transporte
- Las capas superiores del modelo OSI
- El papel de las redes de computadoras en el ciclo de vida de los datos

Recursos Complementarios

- Manual de Wireshark
- Captura Wireshark 'sip-rtp-g711.pcap'
- Computer Networking: A Top-Down Approach, Kurose, 7th edition

Criterios de valoración

- La PEC debe resolverse de manera individual.
- Es necesario justificar todas las respuestas a los ejercicios propuestos en la PEC.
- La puntuación (sobre un total de 10 puntos) asociada a cada ejercicio de esta PEC se indica al inicio de cada enunciado.

Los ejercicios de esta PEC constituyen la parte correspondiente del cómputo de la evaluación continua de la asignatura: $AC = 0{,}2 \cdot PEC1 + 0{,}3 \cdot PEC2 + 0{,}3 \cdot PEC3 + 0{,}2 \cdot PEC4$

Para más información sobre el modelo de evaluación de la asignatura os remitimos al Plan Docente.

Formato y fecha de entrega

La entrega de esta PEC constará de un archivo `.pdf` siguiendo el formato específico **PEC3*Apellido1Apellido2Nombre*.pdf** e incluirá:

- La memoria de la práctica en formato `.pdf` que contenga las respuestas correspondientes a todos los Ejercicios planteados.

Este archivo `.pdf` deberá enviarse a través de la herramienta REC (Registro de Evaluación Continua) del aula **antes de las 23:59 del día 10/12/2021**.

IMPORTANTE: Recordad que la PEC es individual. La detección de falta de originalidad será penalizada conforme a la normativa vigente de la UOC. Además, al hacer la entrega aseguraos de comprobar que el fichero entregado es el correcto, pues es responsabilidad del alumno realizar las entregas correctamente. No se aceptarán entregas fuera de plazo.

1. Primer contacto con Wireshark

1.1. Iniciar Wireshark

Wireshark se ha convertido en el principal analizador de red utilizado por los ingenieros de redes. Este software de código abierto está disponible para muchos sistemas operativos diferentes, incluidos Windows, MAC y Linux. Wireshark se puede descargar desde www.wireshark.org.

Utiliza el recurso *Manual de Wireshark* para instalar y obtener una primera visión de conjunto del aplicativo. Elige la versión de software que necesites según la arquitectura y el sistema operativo de tu PC o laptop. Por ejemplo, si tienes un PC de 64 bits con Windows, selecciona Windows Installer (64-bit) (Instalador de Windows [64 bits]).

Para capturar datos de la red activa, Wireshark requiere la instalación de WinPcap o de Npcap. Si WinPcap o Npcap ya están instalados en el PC, la casilla de verificación Install (Instalar) estará desactivada. Si la versión instalada de estos programas es anterior a la versión que incluye Wireshark, se recomienda que permitas que la versión más reciente se instale haciendo clic en la casilla de verificación Install WinPcap o Npcap.

1.2. Selecciona una interfaz de red

Una vez finalizada la instalación, haz doble clic en el icono de Wireshark en tu escritorio o navega hasta el directorio que contiene el archivo ejecutable e inicia Wireshark. En la pantalla inicial de Wireshark se listarán todas las interfaces de red con las que el Wireshark puede capturar datos. Selecciona la interfaz por la cual estás conectado a Internet. Es muy posible que estés conectado a través de Wi-Fi. En ese caso, selecciona la tarjeta de red Wi-Fi. En caso de que estés conectado a través de Ethernet, selecciona la interfaz correspondiente.

Los nombres de las interfaces dependerán del modelo de tarjetas que lleven vuestros equipos. Una manera de ver qué interfaces nos están proveyendo de tráfico es a través de la pequeña gráfica situada justo al lado del nombre de las interfaces listadas justo al abrir Wireshark. En las interfaces con tráfico se apreciarán curvas variantes, mientras que en las que no tienen tráfico las gráficas se mantendrán constantes en el origen. En el ejemplo de la Figura 1 se observa actividad en la interfaz de Wi-Fi *Wi-Fi 2*, que es la que nos provee de conexión a Internet.[1]

En el menú **Capture**, puedes elegir en la opción *Options* algunos parámetros para especificar la captura. Por ejemplo, en la pestaña *Options*, puedes configurar que la captura se pare automáti-

[1] Obviad la interfaz de red virtual de loopback.

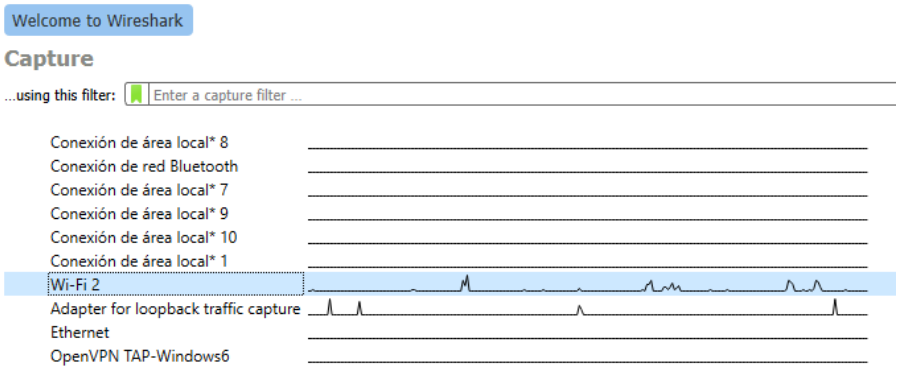


Figura 1: Ejemplo de posibles interfaces listadas en Wireshark.

camente después de un cierto tiempo o después de haber capturado una cierta cantidad de datos. Cuando pulses sobre *Start*, el programa empezará a capturar todos los paquetes que pasen a través de la interfaz que hayas seleccionado. Seguramente observaremos que, aunque no estemos haciendo nada en particular, el programa ya empieza a capturar paquetes. Veamos qué es lo que Wireshark nos está mostrando:

1. *No.*: La primera columna representa un identificador de paquete generado por el programa para tener un orden secuencial de los paquetes en base al orden de llegada.
2. *Time*: La segunda columna muestra el tiempo (por defecto, en segundos) que transcurre entre que se captura un paquete y el tiempo origen (considerado como 0). Por ejemplo, una entrada que indica un *Time* de 3 segundos corresponde a un paquete capturado 3 segundos después de iniciar la captura (es decir, de hacer clic en *Start*).
3. *Source*: La tercera columna muestra la dirección IP que ha generado el paquete.
4. *Destination*: La cuarta columna muestra la dirección IP a la que va dirigido el paquete.
5. *Protocol*: La quinta columna muestra el protocolo asociado al paquete.
6. *Length*: La sexta columna muestra la longitud del paquete en bytes.
7. *Info*: La última columna describe información adicional de los protocolos de comunicación utilizados, como por ejemplo valores específicos de ciertas cabeceras de interés.

En la casilla *Apply a display filter*, ubicada en la parte superior de la pantalla principal de Wireshark, podemos introducir filtros de visualización que nos ayuden a leer e interpretar los datos. Algunos de los filtros de visualización más comunes de Wireshark y que utilizaremos en esta práctica se listan a continuación a través de ejemplos. Puedes encontrar más información sobre los filtros de visualización en: `https://gitlab.com/wireshark/wireshark/-/wikis/DisplayFilters`.

- `ip.addr==192.168.0.1`: Filtra todo el tráfico con dirección IP 192.168.0.1 como origen o destino.
- `tcp.port==80`: Filtra todo el tráfico con el puerto 80 como origen o destino.
- `ip.src==192.168.0.1 and ip.dst==10.100.1.1`: Filtra todo el tráfico que tiene como origen 192.168.0.1 y destino 10.100.1.1.
- `icmp`: Filtra sólo el tráfico del protocolo ICMP.
- `http`: Filtra sólo el tráfico del protocolo HTTP.
- `dns`: Filtra sólo el tráfico del protocolo DNS.
- `sip`: Filtra sólo el tráfico del protocolo SIP.

1.3. ICMP y Ping

Empecemos con un primer ejercicio para contextualizar.

Ejercicio 1 [0,25p]: *¿Cuáles son las direcciones MAC e IPv4 de tu ordenador? Realiza una captura de pantalla del símbolo de sistema (terminal), donde se puedan leer todas las direcciones IPv4 y MAC de tu equipo. ¿Cuál es el comando que has utlizado? Recuerda que para acceder al símbolo de sistema, si estás usando un sistema operativo Windows, escribe en el buscador* `cmd` *y presiona Enter. En MacOS puedes abrir la carpeta Aplicaciones, luego Utilidades y hacer doble clic en Terminal.*

Según el documento 'Las capas inferiores del modelo OSI', cada tarjeta de red tiene un número de identificación único de 48 bits en hexadecimal denominado dirección MAC. En el documento 'El nivel de red', se define la dirección IP como un identificador asociado a routers o equipos finales, para que estos se puedan identificar de manera unívoca y se les pueda enviar información. Tal y como se puede ver en la Figura 2, tanto la dirección IPv4 como la MAC (dirección física) de cada tarjeta se pueden encontrar fácilmente mediante la ejecución en el terminal de los comandos ipconfig /all para Windows, o /sbin/ifconfig para MacOS/Linux.

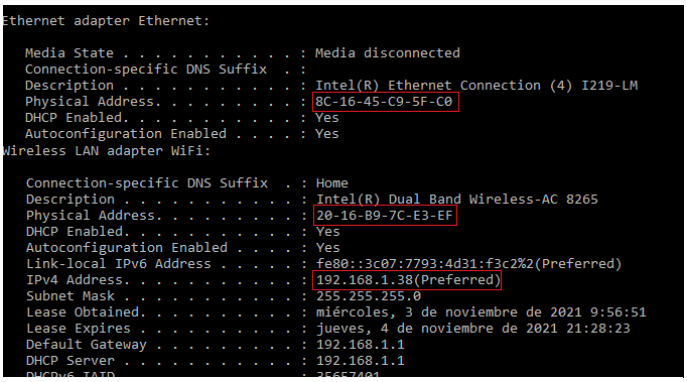


Figura 2: Ejemplo ejecución ipconfig /all

Sigamos con una primera captura de los paquetes generados al realizar un ping. En concreto, vamos a realizar un ping desde el símbolo de sistema (o terminal). Una vez abierta la consola, ejecuta el comando `ping www.uoc.edu` (o la página web que prefieras). Si estás utilizando Linux o MacOS, debes abrir el terminal y ejecutar el comando `ping -c 4 www.uoc.edu` (donde el parámetro `c` del comando `ping` permite especificar el número de paquetes ping a enviar). Después de haber ejecutado el ping correctamente, se puede detener la captura en Wireshark pulsando el botón rojo que indica *Stop*.

Analicemos ahora el intercambio de mensajes ICMP en Wireshark al realizar un ping. Escribe `icmp` en el campo *Filter* en la ventana principal de Wireshark. A continuación, haz clic en *Apply* a la derecha del filtro. Wireshark pasa a mostrar solo los mensajes relativos al protocolo ICMP.

Ejercicio 2 [0,25p]: *¿Cuántos paquetes ICMP se observan? Adjunta una captura de pantalla al respecto. ¿Cómo se identifica un mensaje ICMP en el datagrama IP? ¿Cuál es el tamaño de las peticiones que manda por defecto el comando ping? Indica cuál sería el comando para generar 6 peticiones de 64 bytes cada una.*

Como se muestra en la Figura 3, el comando `ping` genera 8 mensajes ICMP: 4 solicitudes (pings) y 4 respuestas (pongs). Más formalmente, 4 paquetes enviados por el origen (*echo ping request*) y 4 enviados por el destino (*echo ping reply*). Los mensajes ICMP se incorporan dentro de los datagramas IP poniendo el valor del campo protocolo de la cabecera IP a 1. El valor 1 en este campo nos indica que el campo de datos contiene el mensaje ICMP. Como podemos ver en la misma Figura, el tamaño de las peticiones que manda por defecto el comando ping es de 32 bytes.

El comando para generar 6 peticiones de 64 bytes cada una podría ser *ping -n 6 -l 64 www.uoc.edu*

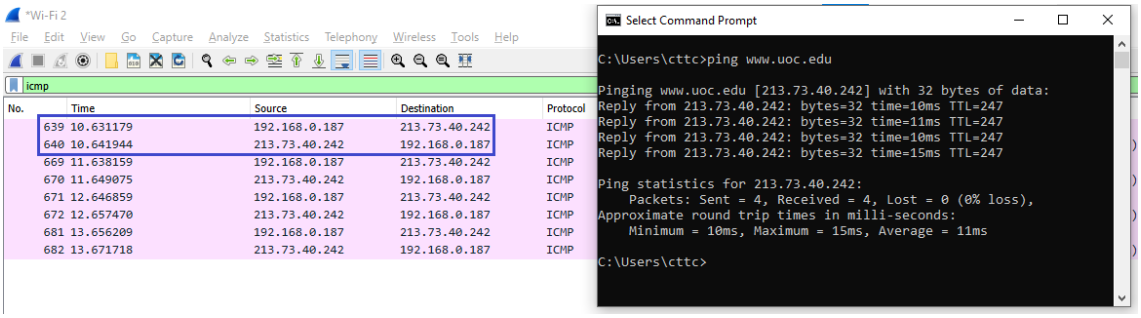


Figura 3: Ejecución de un ping.

2. Observar el comportamiento al navegar por Internet

Para garantizar la interoperabilidad entre implementaciones de distintos fabricantes (por ejemplo, para que clientes Web como pueden ser Chrome o Firefox se puedan 'entender' con servidores de Apache o Microsoft), Internet se basa en el uso de protocolos estándares y abiertos - cualquiera puede acceder a la especificación (se encuentran en Internet) e implementarla. La principal organización que se encarga de definir estos estándares para Internet es la Internet Engineering Task Force (IETF). La IETF se organiza en Working Groups (WG) en los que, personas voluntarias, colaboran para definir conjutamente estos estándares y los publican en las llamadas RFC (Request for Comments). Como ejemplo, el protocolo TCP inicialmente se definió en la RFC número 793, publicado en 1981, y posteriormente se fueron añadiendo funcionalidades adicionales en nuevas RFCs.

Ejercicio 3 [0,25p]: *Indica cuáles son los números de RFC para los siguientes protocolos y, para cada uno de ellos, indica el año de publicación:*

- UDP
- HTTP versión 1.0
- HTTP versión 2.0

- QUIC

- UDP fue inicialmente publicado por J. Postel en la RFC 768 en el año 1980
- HTTP versión 1.0 se publicó en la RFC 1945 en el año 1996
- HTTP versión 2.0 se publicó en la RFC 7540 en el año 2015
- QUIC en su primera versión se ha publicado en la RFC 9000 en Mayo de 2021. QUIC representa un primer paso hacia la publicación de HTTP versión 3.0, que se espera suceda en breve.

Tal y como recogen los materiales del curso, más allá de UDP o TCP existen otros protocolos de transporte como SCTP.

Ejercicio 4 [0,25p]: *Indica una característica del protocolo SCTP que lo diferencie de TCP. ¿Qué significa que el protocolo de transporte SCTP tiene la capacidad de multi-homing?*

El protocolo SCTP ofrece fiabilidad, control de flujo y secuencia como TCP. A diferencia de este, permite el envío de mensajes desordenadamente, el handshake es four-way y tiene las capacidades de multistreaming y multihoming. Multihoming significa que uno (o dos) de los extremos de una asociación (conexión) puede tener más de una dirección IP y por tanto tener acceso a distintas rutas. Esto permite reaccionar de forma transparente ante fallos en la red: si hay algún problema en una de las rutas IP se puede mandar el paquete por una ruta alternativa de forma totalmente transparente para la aplicación. Ver Figura 4.

Veremos ahora los paquetes generados al abrir una página web. Es momento de introducir el protocolo HTTP.

2.1. Comportamiento del protocolo HTTP

El protocolo HTTP (Hypertext Transfer Protocol) es un protocolo de aplicación para transmitir información en la web a través de una arquitectura cliente-servidor. Para dotar de seguridad al intercambio de datos entre cliente y servidor, el protocolo HTTP se suele transmitir (cada vez más) en combinación con otros protocolos, como por ejemplo el SSL (Secure Sockets Layer) y más

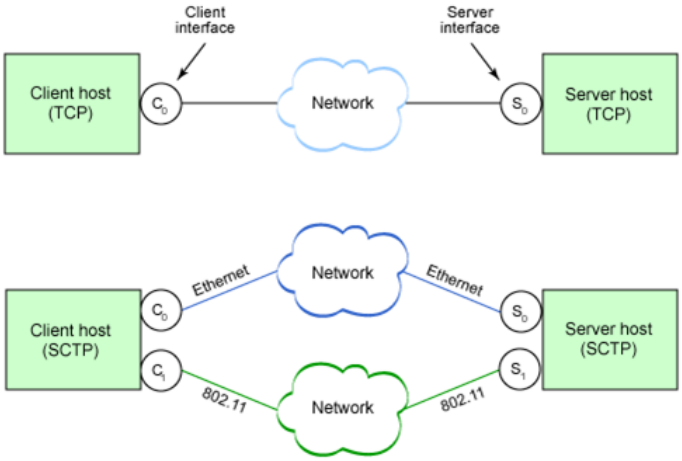


Figura 4: SCTP multi-homing.

recientemente el TLS (Transport Layer Security). A nivel de la arquitectura de red, el protocolo TLS se sitúa entre la capa TCP/IP y el HTTP. Esta combinación da lugar al protocolo HTTPS, también conocido como HTTP over TLS.

Para generar tráfico HTTP tan solo tenemos que introducir en nuestro navegador de Internet habitual la URL de una página web HTTP (y no HTTPS). Un ejemplo de web HTTP es `http://www.washington.edu/` .

Cuando un navegador intenta acceder a una página web realiza una petición a un servidor HTTP. El protocolo HTTP tiene dos métodos principales para que el cliente (navegador) realice esta petición al servidor: el método POST y el método GET. Si todo ha ido bien, tras recibir esta petición, el servidor web donde se encuentra alojada la página web contesta satisfactoriamente a través del mensaje 200 OK. Para ello, encapsula en un mensaje HTTP el código HTML que contiene la información requerida.

Inicia una captura de Wireshark y filtra por protocolo HTTP (para ello, escribe `http` en el campo *Filter* en la ventana principal del Wireshark y haz clic en *Apply*). Verás que al navegar por alguna de estas páginas el número de paquetes HTTP aumenta sensiblemente.[2] Después de haber capturado paquetes durante aproximadamente un par de minutos detén la captura de Wireshark. Vamos a utilizar esta captura para los ejercicios que siguen [5-11].

[2]Si no consigues capturar ningún paquete HTTP al entrar en estas páginas, borra el historial de cookies de tu navegador e inténtalo de nuevo.

Ejercicio 5 [0,75p]: *Proporciona una captura de pantalla del resultado del filtro HTTP, donde se pueda ver una petición POST o GET y su respuesta correspondiente. Contesta a las siguientes preguntas: 1) ¿Qué mensaje HTTP es el mensaje de respuesta a la petición GET o POST? 2) ¿Cuánto tiempo ha pasado entre la petición del cliente y la respuesta del servidor? 3) ¿Cuál es el número asignado por Wireshark al mensaje de petición y al de respuesta? 4) ¿Puedes identificar la dirección IP del cliente? ¿Y la dirección IP del servidor? 5) Por último, indica qué versión del protocolo HTTP estamos utilizando*

1) Como se puede ver de la Figura 5, se ha generado una petición del cliente a través del método GET en el instante 13,06666 s. El servidor ha contestado con un mensaje 200 OK en el instante 13,186484 s. 2) Por tanto, el servidor ha tardado 0,119824 s en responder a la petición del cliente. El servidor contesta con un mensaje HTTP 200 OK, en nuestro caso, pero podría también contestar con otros mensajes. 3) La petición del cliente es el mensaje número #2348 y la respuesta del servidor es el mensaje número #2380. 4) La dirección IP del cliente es la de nuestro ordenador, en nuestro caso 192.168.1.135. Y la dirección IP del servidor es 216.165.47.12. 5) Como se puede observar, estamos utilizando la versión 1.1 del protocolo HTTP, definido en la RFC 2616.

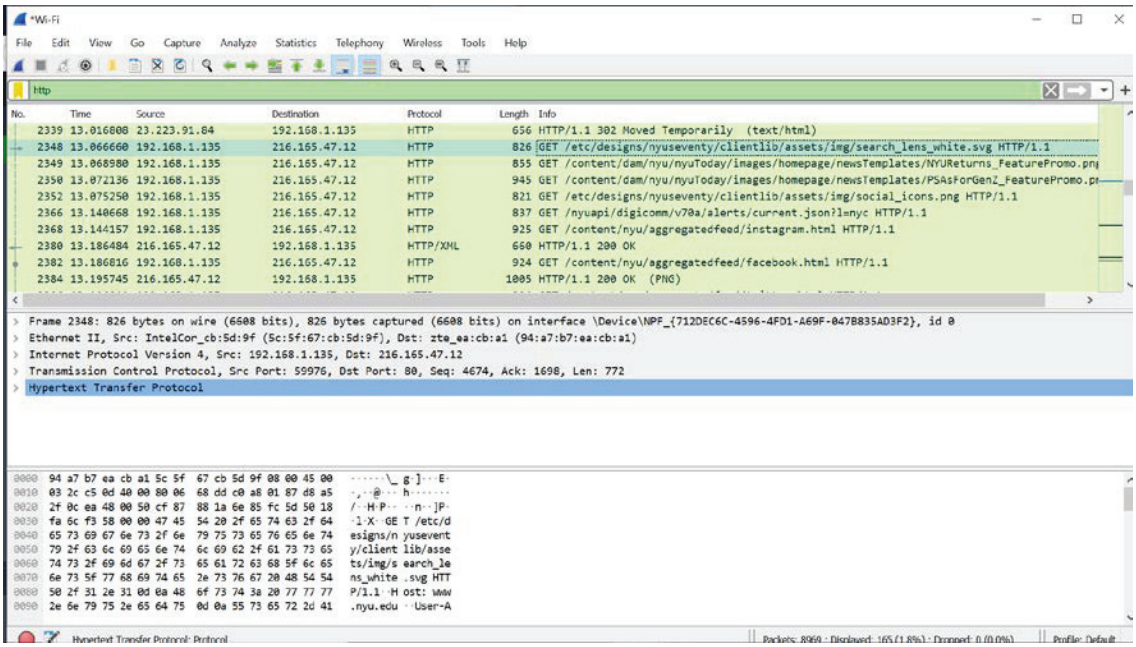


Figura 5: Captura del filtro HTTP y GET.

Ejercicio 6 [0,5p]: *Selecciona un paquete HTTP de tu captura Wireshark y busca los detalles de la cabecera IP. Incluye una captura de pantalla con los detalles del paquete seleccionado. Ayudándote de los documentos de la capa de red que hemos visto en el reto 2, identifica los elementos de la cabecera IP y descríbelos.*

El datagrama IP se describe en el documento 'El nivel de red'. La cabecera IP contiene: 1) la versión, 2) la longitud de la cabecera, 3) el servicio, 4) la longitud total, 5) el identificador, 6) los flags, 7) el offset, 8) el tiempo de vida (TTL), 9) el protocolo de capa superior que ha generado el paquete, 10) los header checksum, 11) y las direcciones de origen y destino y se pueden todos reconocer en la cabecera tal y como se muestra en el Wireshark. Ver ejemplo en Figura 6.

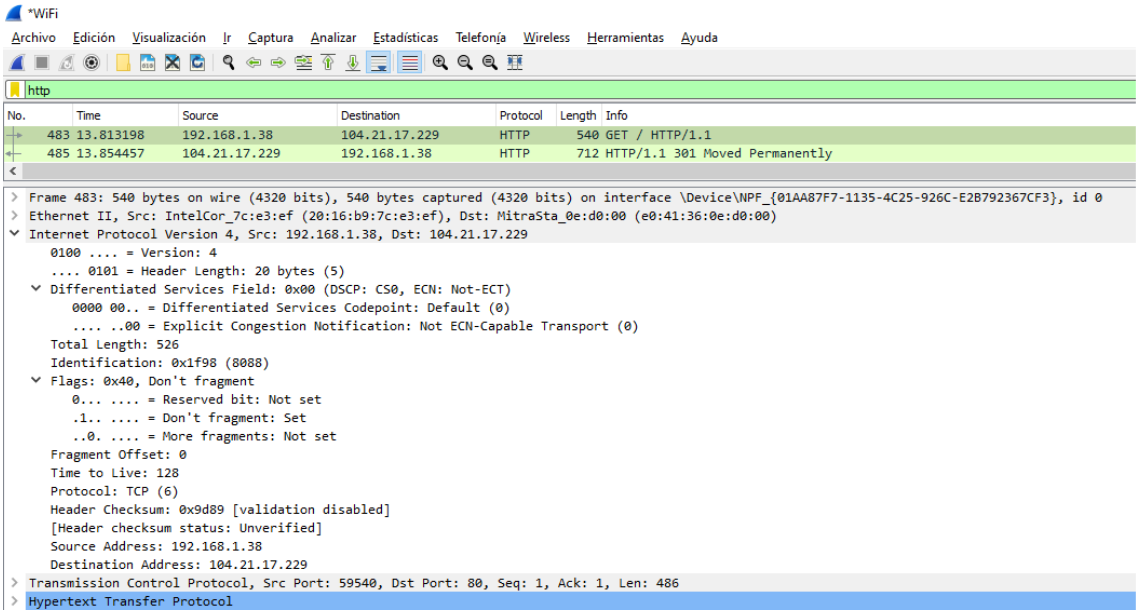


Figura 6: Detalles IP.

Ejercicio 7 [0,25p]: *Filtra ahora los mensajes capturados para visualizar sólo los que tengan tu dirección IP como origen e incluye una captura de pantalla.*

En la Figura 7 podemos observar los paquetes filtrados según la dirección IP de la origen, seleccionando sólo los que han salido de nuestro ordenador. Para ello, hemos utilizado el filtro de visualización ip.src == 192.168.1.135 (siendo 192.168.1.135 la dirección IP de nuestro ordenador).

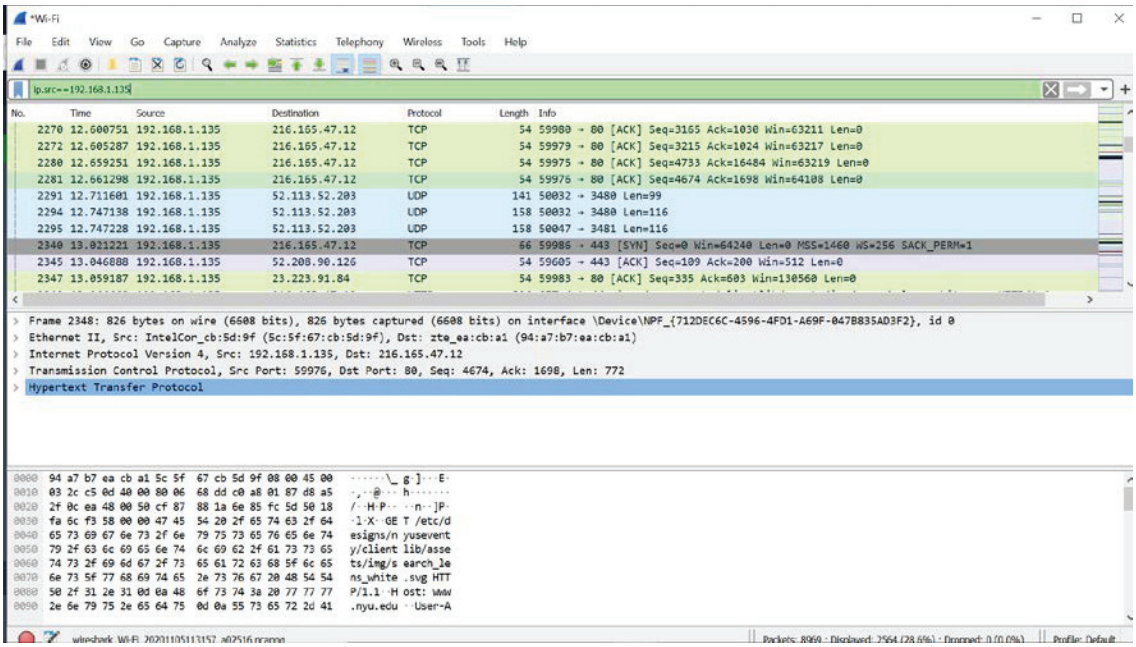


Figura 7: Captura del filtro dirección IP origen.

Ejercicio 8 [0,75p]: *Manteniendo el filtro de tu dirección IP, identifica la dirección MAC de tu ordenador. ¿Cuál es? En los varios paquetes filtrados, salientes de tu ordenador, identifica también la dirección MAC de destino. ¿A qué dispositivo corresponde? Incluye una captura de pantalla donde se pueda ver la dirección MAC de origen y destino.*

Como podemos ver en la Figura 7, estamos filtrando paquetes que tienen como origen nuestro ordenador, por eso, la dirección MAC es la que se indica como origen. En mi caso es 5c-5f-67-cb-5d-9f. La dirección MAC de destino de los paquetes salientes de mi ordenador es 94:a7:b7:ea:cb:a1. La dirección MAC de destino corresponde al siguiente salto al que tu ordenador enviará los paquetes en su camino hacia el dispositivo de destino. Normalmente, esa dirección MAC corresponde a la del router que nos da acceso a Internet.

Ejercicio 9 [0,5p]: *Selecciona un paquete HTTP de tu captura Wireshark y busca los detalles de la cabecera TCP. Incluye una captura de pantalla con los detalles del paquete seleccionado. Ayudándote con los documentos de la capa de transporte que hemos visto en el reto 3, identifica*

los elementos de la cabecera TCP y desríbelos brevemente.

El segmento TCP se describe en el documento 'El nivel de transporte'. La cabecera TCP contiene: 1) puerto de origen y destino (indican el número del puerto que inicia la transmisión al host origen y el número del puerto destino de la transmisión), 2) número secuencia (identifica el primer byte del campo de datos), 3) número de ACK (se utiliza para enviar un acuse de recibo), 4) longitud de la cabecera (la longitud es variable, dado que puede haber opciones o no), 5) suma de verificación (este se aplica a todo el segmento TCP y permite al host de destino determinar si el segmento recibido tiene errores), 6) punto de urgencia (indica si hay datos urgentes en el campo datos y su localización), 7) flags (utilizados para instruir sobre las propiedades del envío), 8) tamaño de ventana (para indicar la cantidad de datos esperada por el receptor) y se pueden todos reconocer en la cabecera tal y como muestra Wireshark. Ver ejemplo en Figura 8

Ejercicio 10 [0,25p]: *Identifica el puerto destino que usa el protocolo de la capa de transporte en el primer paquete GET o POST de la conexión HTTP que has iniciado. ¿Hay alguna razón por la que se use ese puerto? ¿Cambiaría ese puerto si en vez de conectarnos contra un servidor HTTP lo hicieramos contra un servidor HTTPS?*

El puerto TCP donde escucha peticiones HTTP el servidor es el 80. Este puerto está reservado para conexiones HTTP y todas las peticiones deben realizarse al mismo (ver Figura 8), ya que este es el puerto estándar para dicho protocolo. Para HTTPS, el puerto estándar es el 443.

Ejercicio 11 [0,5p]: *En el menú Statistics-Protocol Hierarchy de Wireshark podrás obtener una lista porcentual de los diferentes paquetes capturados. En particular podrás analizar qué porcentaje de paquetes pertenece a cada protocolo. Vamos a fijarnos en los protocolos de transporte. Contesta a las siguientes preguntas: 1) De los paquetes que has capturado, ¿qué porcentaje de paquetes usa TCP como transporte, y qué porcentaje usa UDP? Incluye una captura de pantalla que muestre esta estadística. 2) Teniendo en cuenta esta estadística, ¿qué podemos deducir respecto al protocolo de transporte dominante cuando navegamos por Internet?*

1) De la captura realizada, y como se ve en la Figura 9, un 89,5 % de los paquetes usa TCP y sólo un 10,5 % usa UDP. 2) Deducimos que TCP es el protocolo de transporte predominante cuando navegamos por Internet. Al realizar una videollamada, o cualquier otra aplicación en tiempo real, el tráfico predominante sería UDP, ya que en este tipo de servicios prima el real-time. Es decir, preferimos dar paquetes por perdidos (y no ver la imagen perfectamente o saltarnos algún frame) a tener que esperar un tiempo de buffer cuando algunos paquetes se pierden o no llegan en orden.

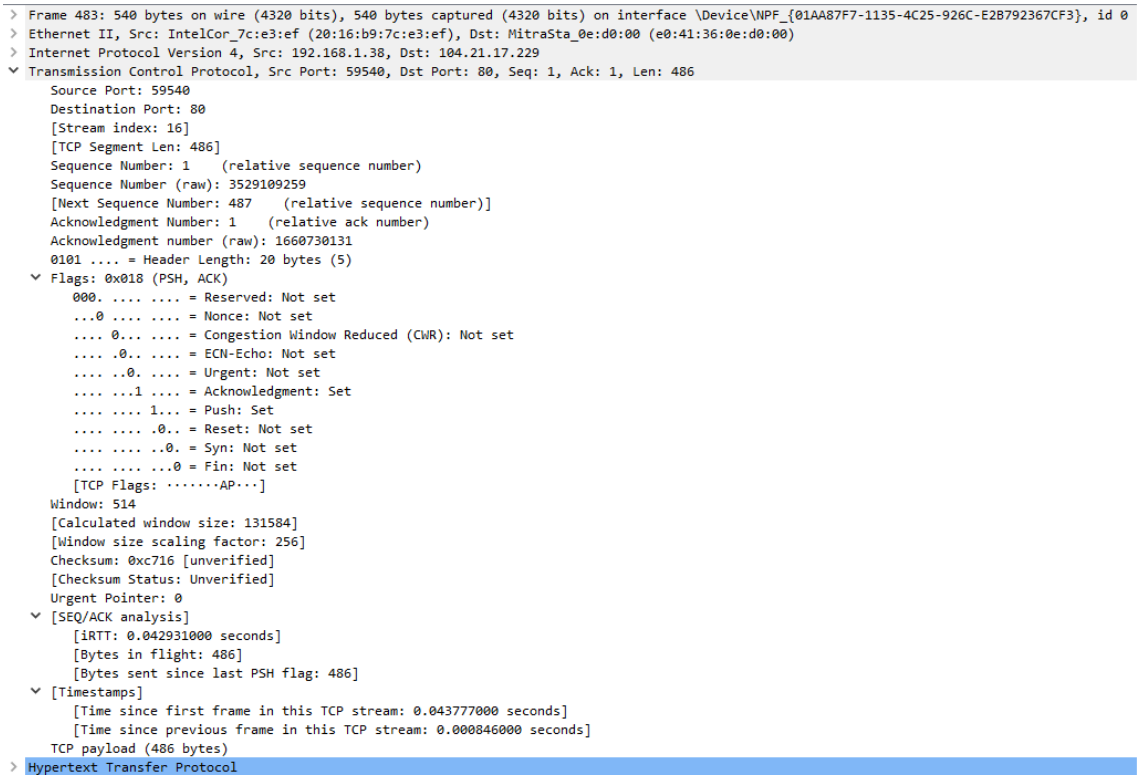


Figura 8: Detalles TCP.

Vamos a analizar ahora algunos aspectos básicos del protocolo TCP. Cuando dos nodos se comunican mediante TCP, se establece una conexión antes de que estos puedan intercambiar los datos. Este procedimiento toma el nombre de *three-way handshake* y se puede consultar en el libro *Computer Networking: A Top Down approach*.

Inicia una nueva captura Wireshark y realiza búsqueda a través de tu navegador web. Filtra los paquetes del protocolo TCP y, para mayor claridad, fíltralos con tcp.port==80. Se visualizarán solo los paquetes del protocolo TCP que pasan por el puerto 80, relativos al protocolo HTTP.

Ejercicio 12 [0,75p]: *1) Describe el procedimiento del three-way handshake teniendo en cuenta el material de lectura propuesto y las trazas capturadas con Wireshark. 2) ¿Cuál es el número de secuencia del segmento TCP SYN? (Identifica el sequence number raw, y no el relativo.) 3) ¿Cuál*

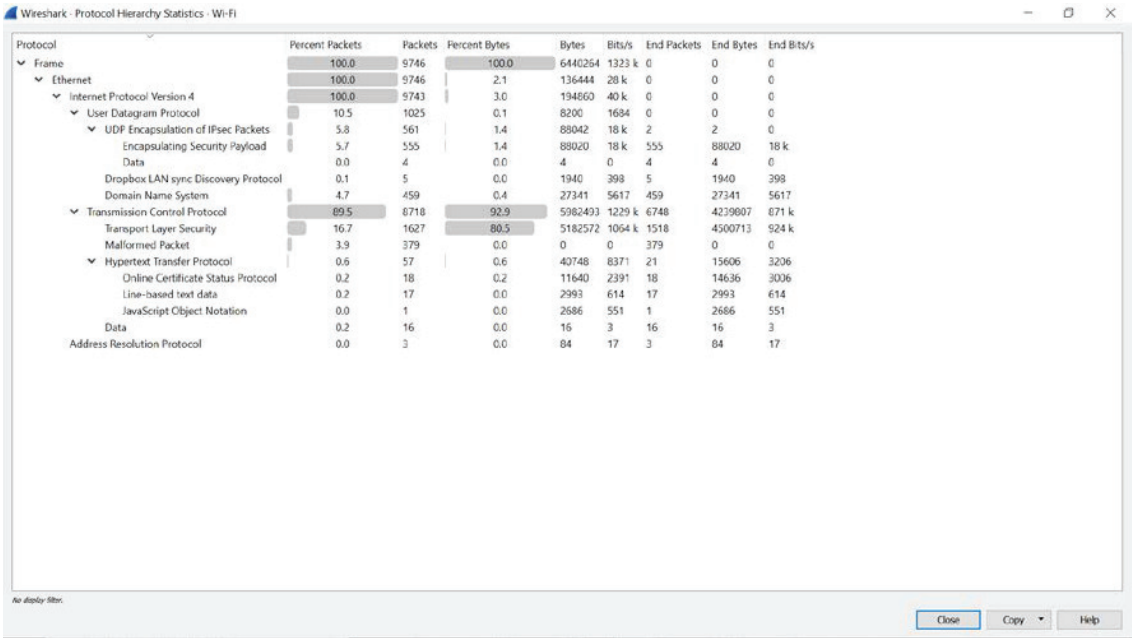


Figura 9: Captura estadísticas de protocolos.

es el número de secuencia (raw) del SYN ACK y el número de acknowledgement enviado por el servidor? ¿Cómo se han determinado estos valores? 4) ¿Cuál es el número de acknowledgement del siguiente ACK? ¿Cómo se ha determinado este valor? Acompaña tus respuestas de una captura de pantalla donde se pueda observar lo descrito.

1) El cliente de una conexión envía un paquete SYN inicial al servidor. El servidor comprueba que el dispositivo destino tenga el servicio solicitado activo y pueda aceptar peticiones en el número de puerto deseado por el cliente. El servidor responde a la petición SYN válida con un paquete SYN ACK. Finalmente, el cliente contesta al servidor con un ACK, completando así la negociación en tres pasos (SYN, SYN/ACK y ACK) y la fase de establecimiento de conexión. 2) En la práctica, y como podemos ver en la Figura 10, el cliente envía un mensaje SYN con número de secuencia 2030341421, y número de acknowledgement igual a 0. 3) El servidor responde con un mensaje SYN ACK con número de secuencia 269475902, y número de acknowledgement igual a 2030341422 (2030341421+1). 3) El cliente acaba el handshake enviando un ACK con número de secuencia 2030341422 (2030341421+1) y número de acknowledgement 269475903 (269475902+1) .

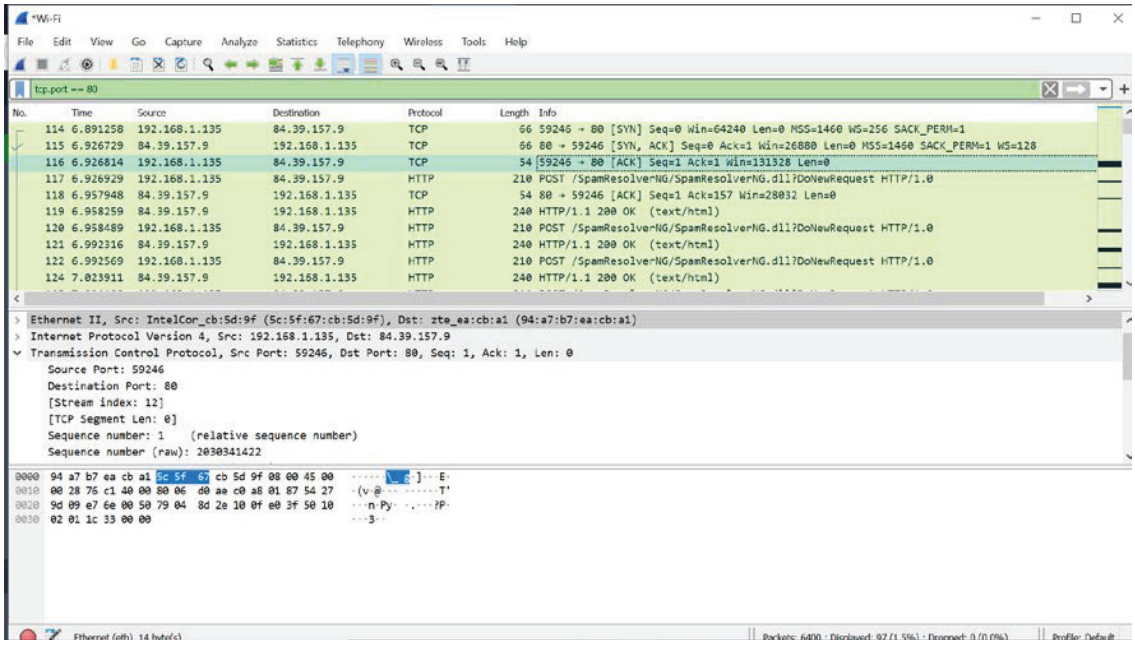


Figura 10: Three-way handshake.

2.2. Comportamiento del protocolo DNS (Domain Name System)

Volvamos a Wireshark y preparémoslo para una nueva captura. Antes de proceder, es recomendable vaciar la caché del DNS para asegurarnos de que la petición se realice. Puedes usar los siguientes comandos: en Windows `ipconfig /flushdns`; en MacOS `sudo killall -HUP mDNSResponder` y en Linux `sudo /etc/init.d/nscd restart`.

Inicia la captura y accede a distintas páginas web como pueden ser `https://webrtchacks.com/`, `https://www.ietf.org/`, `http://phdcomics.com/` u otras. Una vez tengas varias páginas abiertas, detén la captura de Wireshark.

Ejercicio 13 [0,25p]: *Describe el principal objetivo del protocolo DNS. Con la ayuda del menú Statistics de Wireshark, explica cuántos paquetes DNS has capturado en total y su tamaño mínimo, medio y máximo. Incluye una captura de pantalla de la información estadística relacionada.*

El protocolo DNS se utiliza para poder denominar a los computadores mediante nombres simbólicos

en lugar de utilizar las direcciones IP más difíciles de recordar. Los servidores DNS se encargan de responder con la dirección IP buscada. En la Figura 11 podemos observar estadísticas del protocolo DNS. Hemos recogido 78 paquetes, de tamaño mínimo de 29 bytes, máximo de 310 bytes y medio de 72 bytes.

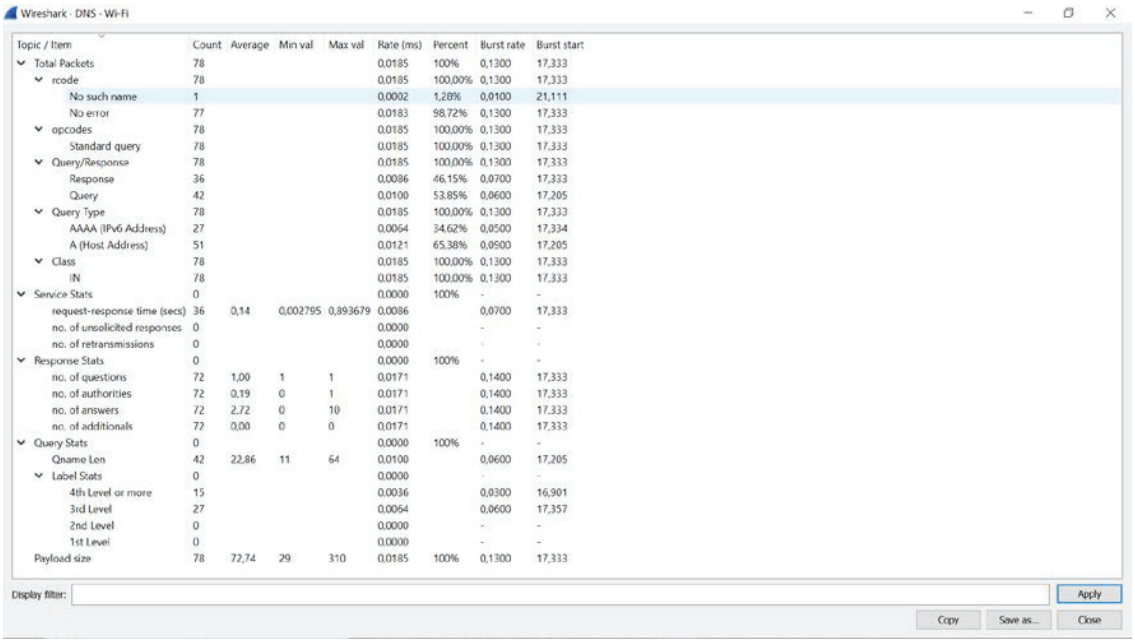


Figura 11: Estadísticas DNS.

Filtra ahora la captura por el protocolo DNS.

Ejercicio 14 [0,5p]: *Identifica las peticiones DNS y sus mensajes de respuesta. 1) ¿Se envían a través de TCP o de UDP? 2) ¿Cuál es el puerto de destino de las peticiones DNS? ¿Cuál es el puerto de origen de los mensajes DNS de respuesta? Adjunta una captura de pantalla donde se pueda apreciar lo observado.*

1) El DNS puede utilizar tanto el UDP como el TCP, pero mayoritariamente hace uso del UDP y todos los mensajes que se han capturado utilizaban el UDP. 2) Tal y como se puede observar en la Figura 12, el puerto de destino de las peticiones DNS es el 53. En la Figura 13, se muestra como el puerto de origen de los mensajes de respuesta es también el 53.

Ejercicio 15 [0,25p]: *¿Cuál es la dirección IP de tu servidor DNS? Justifica la respuesta acom-*

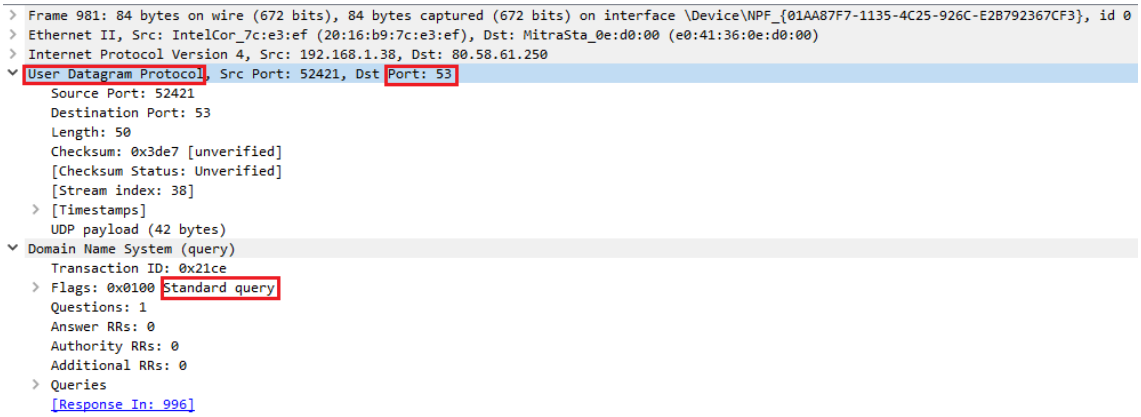


Figura 12: Petición DNS.

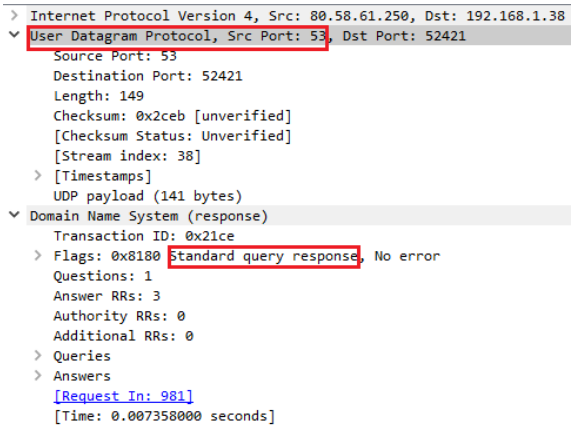


Figura 13: Respuesta DNS.

pañándola de una captura de pantalla de Wireshark.

La dirección IP de mi servidor DNS es la 80.58.61.250 ya que es la dirección a la que se envían las peticiones DNS y que responde a ellas. Esto se puede observar en las Figuras 12 y 13.

2.3. Comportamiento del protocolo SIP (Session Initiation Protocol)

SIP (Session Initiation Protocol) es un protocolo del nivel de aplicación que permite inicializar, modificar y finalizar sesiones interactivas que impliquen elementos multimedia como vídeo, voz, mensajería instantánea, juegos en línea, realidad virtual, etc.

Las operadoras de telecomunicación usan el protocolo SIP para señalizar el tráfico en sus redes más modernas de telefonía (ya sean fijas o móviles) y lo acompañan de flujos RTP (Real-Time Protocol) que contienen la voz de los usuarios. Lo que sucede durante una llamada (quien ha llamado a quien, durante cuanto tiempo, si ha habido problemas, cuál ha sido la calidad del audio, etc) se reporta en los llamados CDR (Call Detailed Records) generando un alto volumen de datos. Esta información la utilizan las operadoras para tomar decisiones sobre cuestiones operativas (dimensionamientos y capacidades de red, identificar problemas, cambiar lógicas de enrutamiento) y para extraer conclusiones sobre el comportamiento de los usuarios.

Veamos ahora como funciona este protocolo utilizando la captura de una llamada de Voz sobre IP (VoIP). Abre con Wireshark el fichero 'sip-rtp-g711.pcap' que puedes encontrar en el Campus (junto con este enunciado) y aplica un filtro para que solo te muestre el protocolo SIP. En Wireshark, ves al menú 'Telefonía' y selecciona la opción 'Flujos SIP'. Selecciona el primer flujo (lo puedes identificar, ya que su estado está marcado como 'COMPLETED') y clica en la opción 'Flow Sequence'.

Ejercicio 16 [0,25p]: *¿Cuántos mensajes SIP puedes ver en el diagrama mostrado? Indica cuáles son.*

En el diagrama de la Figura 14 se pueden observar 6 mensajes SIP. Primero vemos una petición INVITE en la que el teléfono 'origen' indica al teléfono 'destino' que quiere iniciar una llamada, a continuación el teléfono 'destino' responde con una respuesta provisional '100 Trying' para indicar que la petición está en curso. Seguido, cuando el usuario en el teléfono 'destino' acepta la llamada en su terminal, este manda una respuesta final ('200 OK') indicando que la llamada ha sido aceptada. Con esto, el teléfono 'origen' manda un mensaje ACK confirmando el establecimiento de la llamada. Aquí podemos ver que se inicia un intercambio de paquetes RTP (que no son parte del protocolo SIP) que contienen el audio (esto es, la voz) de la llamada. Finalmente, al cabo de unos segundos, uno de los usuarios decide finalizar la llamada por lo que el teléfono 'destino' manda un mensaje 'BYE'. El teléfono 'origen' confirma ('200 OK') que la llamada ha sido finalizada con éxito.

Ejercicio 17 [0,25p]: *Identifica cuáles son las direcciones IP de los dos teléfonos que participan en esta llamada.*

Tal y como se puede ver en la Figura 14, las direcciones IP de los dos equipos que participan en el intercambio (los teléfonos) son 10.0.2.20 y 10.0.2.15

Ejercicio 18 [0,25p]: *¿Cuánto tiempo dura el intercambio de paquetes de audio en esta llamada? Indica como lo has calculado.*

Tal y como se puede ver en la Figura 14, el intercambio de paquetes de RTP con audio tiene una duración de 8.48s.

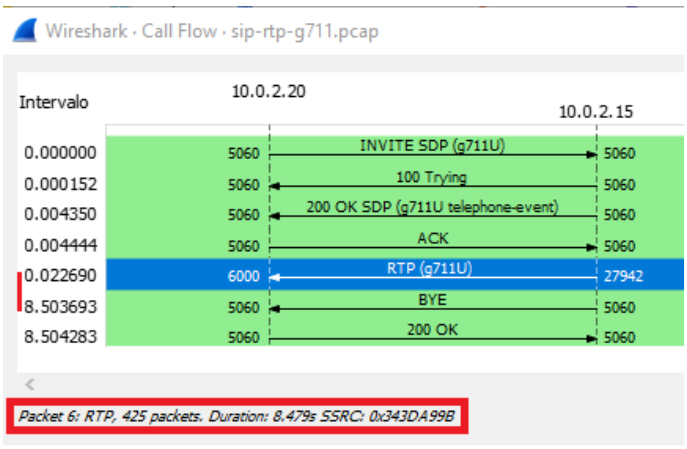


Figura 14: Captura SIP y RTP

Ejercicio 19 [0,5p]: *Imaginemos ahora que, usando una aplicación basada en SIP, queremos iniciar un stream de video desde nuestra Webcam para conectarnos a la plataforma lanzada recientemente por Facebook, Meta. La Webcam genera una resolución de 640x480 píxeles por frame, donde cada píxel se representa con 3 colores, y cada color con 8 bits. El video resultante tiene una frame rate de 20 frames por segundo.*

1. ¿Cuál es el ancho de banda necesario para transmitir el video sin compresión? ¿Cuál es si aplicamos un algoritmo de compresión con un ratio 40:1?
2. ¿Qué ratio de compresión sería necesario si quisiéramos transmitir el video sobre un enlace de 512 kbps?

1. El ancho de banda necesario sin compresión es

$$r = [640x480]\frac{px}{frame} \cdot 3 \cdot [8]\frac{bit}{px} \cdot [20]fps = [147456000]\frac{bit}{s} = [147{,}5]Mbps$$

Con una compresión de 40:1, se reduce a:

$$r_{comp} = \frac{[147456000]\frac{bit}{s}}{40} = [3686400]\frac{bit}{s} = [3{,}7]Mbps$$

2. Para un enlace de [512]kbps, el video debe comprimirse a un ratio de

$$c = \frac{[147456000]\frac{bit}{s}}{[512000]\frac{bit}{s}} = 288$$

3. Extracción de datos desde una API

API es el acrónimo en inglés de *application programming interface*. La API es básicamente una interfaz que habla por nosotros con un programa. Cuando un programador decide poner a disposición pública algún conjunto de datos generados por sus programas, lo puede hacer a través de una interfaz de este tipo. Otros programadores extraen estos datos de la aplicación construyendo URLs o a través de clientes HTTP. Una API está caracterizada por tres elementos:

- el **usuario**, que es la persona realizando la petición;
- el **cliente**, que es el ordenador que envía la petición al servidor;
- el **servidor**, que es el ordenador que responde a la petición.

Los datos se almacenan en un servidor y la documentación correspondiente se pone a disposición de los programadores para que puedan acceder. Un usuario externo entonces puede realizar peticiones de datos a este servidor. Ejemplos de estos sistemas son las redes sociales comunes como Twitter, Facebook, YouTube, Linkedin, etc. Incluso portales como la NASA o el New York Times ponen sus datos a disposición para que los desarrolladores puedan acceder a ellos y manipularlos.

A través de una API conocida como RESTful API, podemos realizar peticiones (*queries*) utilizando el protocolo HTTP. ¿Cómo? Lo haremos realizando *HTTP-requests* y *HTTP-responses*. Concretamente, las APIs pueden realizar un número limitado de acciones entre las que encontramos:

- **GET**: solicita datos a un servidor;
- **POST**: envía cambios desde el cliente al servidor;

- **PUT**: revisa o añade a la información existente;
- **DELETE**: destruye información existente.

¿A qué tipos de datos podemos acceder? Por ejemplo, con Facebook podríamos obtener la lista de amigos entre un amigo nuestro y nosotros mismos, o el número de likes obtenidos en ciertas páginas. A Twitter le podríamos pedir la lista de temas y argumentos más tratados en una cierta área geográfica, por ejemplo en España. A través de la Google Direction API podemos obtener rutas para llegar de una dirección a otra. Por último, desde la Google Book API, podríamos obtener la lista de libros de un autor, etc.

La mayoría de aplicaciones mencionadas requieren gestionar una autorización que se tiene que pedir, concretándose en una contraseña o clave de la API. Las API públicas también tienen limitaciones en cuanto a cantidad de datos disponibles y número de peticiones por segundo posibles. Los datos que obtenemos se representan a través de un formato específico conocido como JSON (Java Script Object Notation). JSON es un formato estándar que se usa para transmitir datos entre un servidor y una aplicación web como alternativa al XML.

Vamos a utilizar *Fruityvice*, una API abierta y muy sencilla que nos permite obtener datos sobre distintos tipos de frutas. Iniciaremos seguidamente una nueva captura en Wireshark. Antes de empezar, es recomendable vaciar la caché del DNS (ver Sección 2.2).

Realicemos ahora una petición a la API que nos devuelva información sobre todas las frutas existentes en la base de datos: introduce `https://www.fruityvice.com/api/fruit/all`. Una vez el navegador devuelva los datos solicitados podemos parar la captura Wireshark.

Vamos a ver primero como el protocolo DNS resuelve la dirección IP del servidor de la API FruityVice. Filtramos los paquetes capturados por DNS y buscamos la petición al servidor DNS local de la dirección IP del servidor `www.fruityvice.com`.

Ejercicio 20 [0,25p]: *¿Cuál es la dirección IP que resuelve el dominio `www.fruityvice.com`? Adjunta una captura de pantalla indicando el paquete donde se encuentra esta respuesta.*

Tal y como se puede ver en la Figura 15, la dirección IP que resuelve el dominio `www.fruityvice.com` es la que se puede ver en la respuesta a la petición DNS: la 217.160.142.194.

Ejercicio 21 [0,5p]: *Teniendo en cuenta la información que has derivado en el ejercicio anterior (sobre la dirección IP que resuelve el dominio de interés), estima cuánto tiempo ha durado la descarga (en milisegundos) de los datos sobre las frutas desde esa dirección IP. Adjunta una captura de pantalla de Wireshark para justificar tu respuesta.*

Tal y como se muestra en la Figura 16, si filtramos por los paquetes que tienen la IP 217.160.142.194

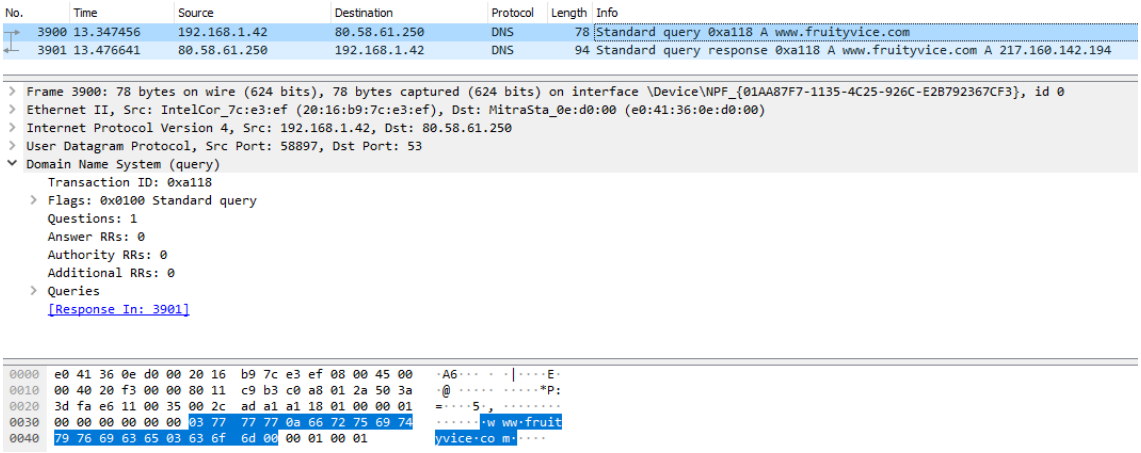
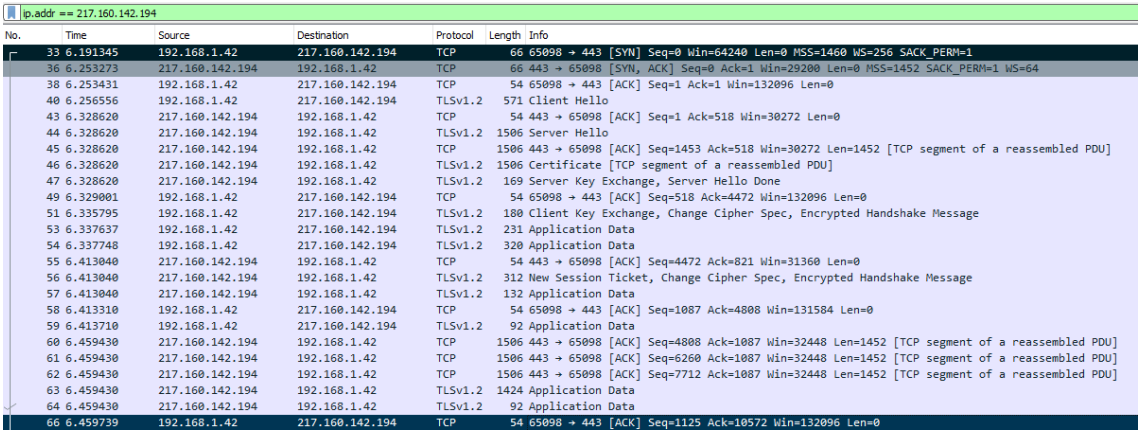


Figura 15: Resolución DNS del dominio `www.fruityvice.com`.

como origen o destino (filtro ip.addr == 217.160.142.194), podemos observar el intercambio de paquetes entre nuestro ordenador y el servidor API. En nuestro caso particular, la conexión TCP con el servidor de la API empieza en el paquete #33 (en el instante 6.191345 s). Una vez establecida esta conexión, da comienzo la transferencia de datos mediante el protocolo HTTPS y podemos observar una serie de paquetes TLS versión 1.2. En nuestro caso, el último TCP ACK se recibe en el mensaje #66 (6.459739 s). Con lo cual, podemos estimar que la duración de la descarga de datos ha sido de $6{,}459739 - 6{,}191345 \approx 268$ ms.

Ejercicio 22 [0,25p]: *Explorando la captura del ejercicio anterior, indica si, teniendo visibilidad de lo que está sucediendo en la red (es decir, lo que vemos en Wireshark), es posible leer el contenido de la información que se está transmitiendo: en este caso la lista de las frutas con sus características. Justifica tu respuesta.*

Tal y como podemos ver en la Figura 16, la transferencia de los propios datos de la aplicación (Application Data) se realiza de forma encriptada y autenticada usando TLSv1.2. Esto significa que, aunque alguien pudiera acceder a un dispositivo de red, como puede ser un router, y obtener así visibilidad del tráfico intercambiado, el valor de la información interceptada sería muy bajo, ya que esta sería ininteligible dado el uso de algoritmos de encriptación. Esta misma propiedad, supone a menudo un reto para aquellos que desean sacar conclusiones sobre los datos que transitan sus redes. Un ejemplo puede ser un operador de telecomunicación: ofrecen la conectividad para que nosotros, como usuarios, podamos acceder a un gran número de aplicaciones y servicios ofrecidos por empresas de Internet. Mientras que este operador puede ver a qué aplicaciones se conectan sus usuarios (las identifican según dirección IP del servidor), a qué horas, con qué volumen de


ip.addr == 217.160.142.194

No.	Time	Source	Destination	Protocol	Length	Info
33	6.191345	192.168.1.42	217.160.142.194	TCP	66	65098 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
36	6.253273	217.160.142.194	192.168.1.42	TCP	66	443 → 65098 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1452 SACK_PERM=1 WS=64
38	6.253431	192.168.1.42	217.160.142.194	TCP	54	65098 → 443 [ACK] Seq=1 Ack=1 Win=132096 Len=0
40	6.256556	192.168.1.42	217.160.142.194	TLSv1.2	571	Client Hello
43	6.328620	217.160.142.194	192.168.1.42	TCP	54	443 → 65098 [ACK] Seq=1 Ack=518 Win=30272 Len=0
44	6.328620	217.160.142.194	192.168.1.42	TLSv1.2	1506	Server Hello
45	6.328620	217.160.142.194	192.168.1.42	TCP	1506	443 → 65098 [ACK] Seq=1453 Ack=518 Win=30272 Len=1452 [TCP segment of a reassembled PDU]
46	6.328620	217.160.142.194	192.168.1.42	TLSv1.2	1506	Certificate [TCP segment of a reassembled PDU]
47	6.328620	217.160.142.194	192.168.1.42	TLSv1.2	169	Server Key Exchange, Server Hello Done
49	6.329001	192.168.1.42	217.160.142.194	TCP	54	65098 → 443 [ACK] Seq=518 Ack=4472 Win=132096 Len=0
51	6.335795	192.168.1.42	217.160.142.194	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
53	6.337637	192.168.1.42	217.160.142.194	TLSv1.2	231	Application Data
54	6.337748	192.168.1.42	217.160.142.194	TLSv1.2	320	Application Data
55	6.413040	217.160.142.194	192.168.1.42	TCP	54	443 → 65098 [ACK] Seq=4472 Ack=821 Win=31360 Len=0
56	6.413040	217.160.142.194	192.168.1.42	TLSv1.2	312	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
57	6.413040	217.160.142.194	192.168.1.42	TLSv1.2	132	Application Data
58	6.413310	192.168.1.42	217.160.142.194	TCP	54	65098 → 443 [ACK] Seq=1087 Ack=4808 Win=131584 Len=0
59	6.413710	192.168.1.42	217.160.142.194	TLSv1.2	92	Application Data
60	6.459430	217.160.142.194	192.168.1.42	TCP	1506	443 → 65098 [ACK] Seq=4808 Ack=1087 Win=32448 Len=1452 [TCP segment of a reassembled PDU]
61	6.459430	217.160.142.194	192.168.1.42	TCP	1506	443 → 65098 [ACK] Seq=6260 Ack=1087 Win=32448 Len=1452 [TCP segment of a reassembled PDU]
62	6.459430	217.160.142.194	192.168.1.42	TCP	1506	443 → 65098 [ACK] Seq=7712 Ack=1087 Win=32448 Len=1452 [TCP segment of a reassembled PDU]
63	6.459430	217.160.142.194	192.168.1.42	TLSv1.2	1424	Application Data
64	6.459430	217.160.142.194	192.168.1.42	TLSv1.2	92	Application Data
66	6.459739	192.168.1.42	217.160.142.194	TCP	54	65098 → 443 [ACK] Seq=1125 Ack=10572 Win=132096 Len=0

Figura 16: Flujo de datos API.

tráfico, etc. no pueden acceder al propio contenido de las aplicaciones cuando este va encriptado. Aunque, a nivel de usuario, esto es un valor por lo que a la privacidad y confidencialidad se refiere, esto limita su capacidad para extraer conclusiones sobre los datos intercambiados (por ejemplo, presentar publicidad en función del contenido de una conversación de chat).

Imaginemos ahora que hemos fundado nuestra propia startup para ofrecer, a través de Internet, una API REST que expone datos sobre la evolución, cotización y transacciones de distintos tipos de criptomoneda. Esta API está diseñada para que la consuman aplicaciones de trading que, con la información que extraigan de la API, van a generar recomendaciones de inversión personalizadas para cada uno de sus clientes. Llevamos poco tiempo con este proyecto y, aunque tenemos a muchos clientes que están probando nuestro servicio de forma gratuita, son pocos los que han dado el paso al servicio premium de pago. Con esto, tenemos que vigilar muy bien cuáles son nuestros costes de operación. Por este motivo, como se indica en la Figura 18, vamos a exponer nuestra API a través de un único servidor que va a disponer de una conexión a Internet limitada a un ancho de banda de subida de $r_S = 100\,\text{Mb/s}$[3](sentido servidor-clientes). El contenido completo de la información que expondremos vía nuestra API tiene un tamaño de $S = 200\,\text{MB}$[4]. Imaginemos que Elon Musk, fundador de varias compañías entre las que se encuentra Tesla, publica un Tweet anunciando que a partir del próximo año Tesla va a dejar de aceptar moneda tradicional y los pagos deberán hacerse únicamente con Bitcoin. Esto, automáticamente, genera un alto volumen de transacciones de gente intentando comprar Bitcoin en un espacio de tiempo muy corto. Todos nuestros clientes necesitan actualizar sus directrices de inversión inmediatamente, por lo que acceden de forma concurrente a

[3] $1\,\text{Mb/s} = 10^6$ bits/s.
[4] $1\,\text{MB} = 10^6$ bytes $= 8 \cdot 10^6$ bits.

Figura 17: Tweet imaginario de Elon Musk

nuestro servidor para descargar la información actualizada. Las computadoras de nuestros clientes se encuentran distribuidas a través de $N_A = 150$ redes de acceso, de forma que cada red tiene $N_C = 10$ computadoras.

Ejercicio 23 [0,5p]: *Considera que cada computadora de cliente descarga la información actualizada una única vez y que cada petición de actualización devuelve el contenido completo de la API. Calcula:*

1. El volumen total de tráfico enviado desde nuestro servidor, $T_{cs,ul}$
2. El volumen de tráfico recibido por cada una de las redes de acceso, $T_{cs,a}$
3. La velocidad de descarga para cada cliente, $r_{cs,dl}$
4. El tiempo total de descarga para cada cliente, t_{dl}

El volumen total de tráfico enviado desde nuestro servidor es el resultado de mandar una copia del fichero a cada una de las computadoras cliente, por lo que tenemos:

$$T_{cs,ul} = S \cdot N_A \cdot N_C = 200\,\text{MB} \cdot 1500 = 300\,\text{GB}. \tag{1}$$

El volumen de tráfico recibido por cada una de las redes de acceso es:

$$T_{cs,a} = S \cdot N_C = 200\,\text{MB} \cdot 10 = 2\,\text{GB}. \tag{2}$$

Asumiendo un escenario ideal, en el que el ancho de banda se divide de forma uniforme entre todos los clientes, obtenemos:

$$r_{cs,dl} = \frac{r_S}{N_A \cdot N_C} = \frac{100\,\text{Mb/s}}{1500} = 66{,}7\,\text{kb/s}. \tag{3}$$

De forma que el tiempo de descarga para cada cliente es:

$$t_{cs,dl} = \frac{S}{r_{cs,dl}} = \frac{200\,\text{MB}}{66{,}7\,\text{kb/s}} = 6\,\text{h}\,40\,\text{min}. \tag{4}$$

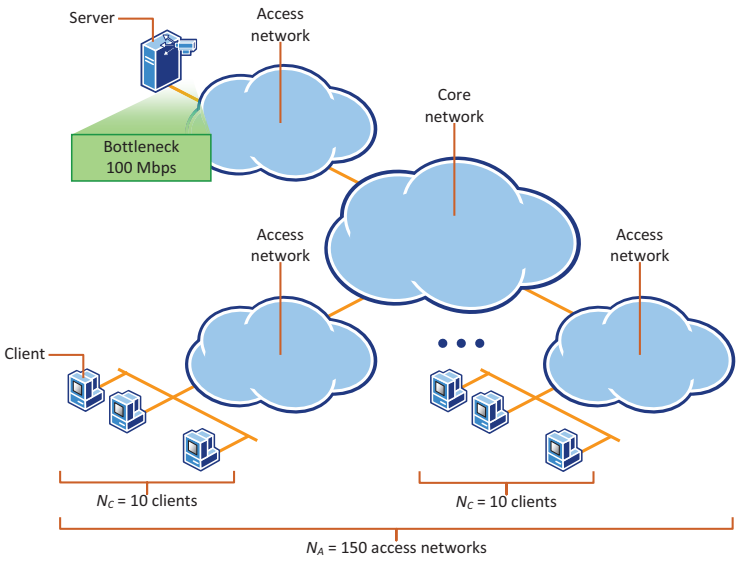


Figura 18: Arquitectura Cliente/Servidor.

Dada la experiencia en el ejercicio anterior, hemos recibido quejas de algunos clientes indicando que el tiempo de descarga de actualizaciones durante el incidente 'Elon Musk' ha sido demasiado largo y no han podido ofrecer recomendaciones de inversión en tiempo real. Para ello, hablamos con nuestro equipo de ingeniería y decidimos que, para poder dar un buen nivel de servicio durante este tipo de eventos, debemos redefinir la arquitectura de red de nuestro servicio. Después de considerar distintas alternativas tecnológicas y sus costes asociados, decidimos llegar a un acuerdo con una empresa de distribución de contenidos, también conocidas como CDN (Content Distribution Networks), tal y como se puede ver en la Figura 19. Esta empresa dispone de un servidor CDN en cada red de acceso, por lo que estos, en todo momento, van a mantener una copia actualizada del contenido de nuestra API. De este modo, las computadoras de nuestros clientes se van a conectar a estos servidores 'espejo' que residen en la red de acceso para descargar la información. La velocidad máxima de subida para estos servidores de la CND es de $r_S = 100\,\text{Mb/s}$ (sentido servidorCDN-clientes).

Ejercicio 24 [0,5p]: *calcula las mismas variables que en el ejercicio anterior, ahora denotadas como* $T_{cdn,ul}$, $T_{cdn,a}$, $r_{cdn,dl}$, $t_{cdn,dl}$.

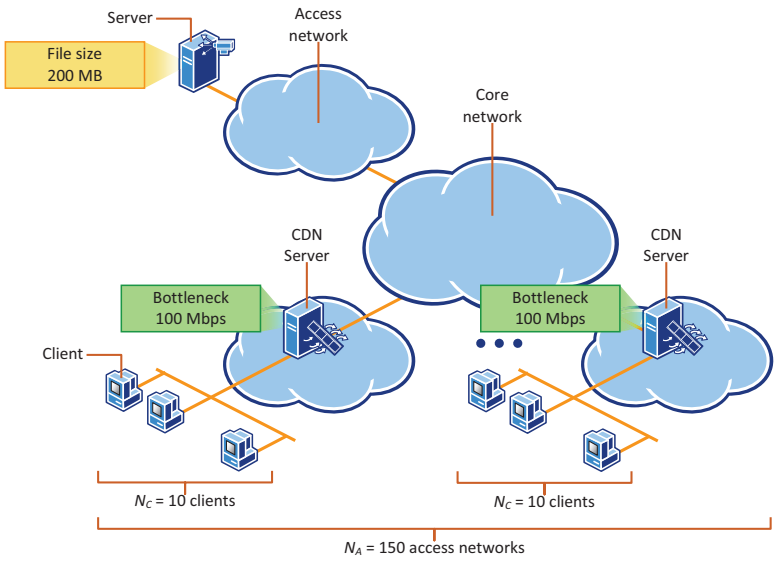


Figura 19: Arquitectura CDN.

El volumen total de tráfico enviado desde nuestro servidor es el resultado de mandar una copia del fichero a cada uno de los servidores CDN, por lo que tenemos:

$$T_{cdn,ul} = S \cdot N_A = 200\,\text{MB} \cdot 150 = 30\,\text{GB}. \tag{5}$$

El volumen de tráfico recibido por cada una de las redes de acceso es:

$$T_{cdn,a} = 200\,\text{MB}. \tag{6}$$

La velocidad de descarga de cada cliente es:

$$r_{cdn,dl} = \frac{r_S}{N_C} = \frac{100\,\text{Mb/s}}{10} = 10\,\text{Mb/s}. \tag{7}$$

De forma que el tiempo de descarga para cada cliente es:

$$t_{cdn,dl} = \frac{S}{r_{cdn,dl}} = \frac{200\,\text{MB}}{10\,\text{Mb/s}} = 2\,\text{min}\,40\,\text{s}. \tag{8}$$

Ejercicio 25 [0,5p]: *Considera el caso de uso de los dos ejercicios anteriores. ¿Qué formato de datos seleccionarías para entregar los datos vía nuestra API REST? Justifica tu respuesta y proporciona un ejemplo de mensaje para este caso de uso.*

Tal y como se puede ver en la Sección 2.1.1. del documento 'Las redes como medio para obtener datos', la arquitectura REST permite a los proveedores de APIs entregar datos en múltiples formatos, como texto plano, HTML, XML y JSON, entre otros. De todos ellos, el formato de intercambio de datos más utilizado en REST es JSON (JavaScript Object Notation), dado su formato ligero y de fácil lectura, así como su rapidez y sencillez a la hora de intercambiar datos. Un sencillo JSON de ejemplo para el caso de uso en cuestión podría ser el de la Figura 20, que podría utilizarse para mostrar al usuario final un dashboard como el representado en la Figura 21.


```
{
  "Name": "CryptoAPI",
  "Description": "Today's Cryptocurrency Prices by MarketCap",
  "Values": [
    {
      "Name": "BitCoin",
      "Price": "$62,131.75",
      "24h%": "+0.18%",
      "7d%": "+1.24%",
      "MarketCap": "$1,172,817,829,191",
      "Volume(24h)": "$34,987,363,137",
      "Circulating Supply": "18,863,556 BTC"
    },
    {
      "Name": "Etherum",
      "Price": "$4,562.01",
      "24h%": "+1.29%",
      "7d%": "+9.36%",
      "MarketCap": "$538,542,865,156",
      "Volume(24h)": "$19,891,878,573",
      "Circulating Supply": "118,204,314 ETH"
    }
  ]
}
```


Figura 20: Ejemplo JSON.

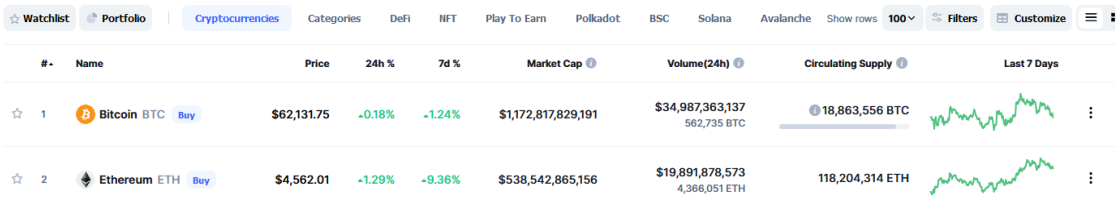


Figura 21: Ejemplo Aplicación.