Trabajo de Laboratorio 3

HTTP. HTML. CGI. AJAX. DFS. DNS

1. Se debe desarrollar una aplicación que presente al usuario una página web con una opción de carga de datos y una opción de modificación de datos. La opción de carga consistirá en mostrar un formulario con un conjunto de datos a completar y que luego almacene estos datos en un archivo de texto plano. Los datos son:

- Nombre y Apellido (hasta 70 caracteres)
- Número de Alumno/ Legajo (cantidad de dígitos limitada)
- Sexo (lista desplegable para elegir entre Masculino o Femenino)
- Edad (hasta dos dígitos)
- Contraseña (que no se muestre en pantalla)

Para el caso de "modificación", se deberá requerir Numero de Alumno y la Contraseña, si este par es encontrado en el archivo de texto plano, se muestra lo que ya se tiene para ser modificado como si se estuvieran cargando los datos, pero con los valores iniciales/default cargados del archivo. Utilizar cookies para permitir sucesivas modificaciones sobre los datos sin volver a pedir el legajo y palabra clave.

Se deben desarrollar las páginas estáticas HTML, el o los programas CGI para procesar los datos y la distribución de estos componentes en el sistema de archivos.
Disponer el servidor Web (apache) para este punto en un contenedor docker
Justifique si este ejercicio puede ser considerado una aplicación web.

Se desarrolló una Aplicación Web que cumple con lo especificado utilizando el lenguaje de Programación/Scripting Python versión 3.

Como servidor Web se utilizó la versión 2 de Apache, el mismo permite ejecutar programas CGI escritos en Python, para esto se procedió a descargar la imagen Httpd:2.4 de docker desde el repositorio Docker Hub, que contiene una distribución de Linux con un servidor Apache2.

A partir de dicha imagen, configurando un DockerFile generamos una nueva imagen que instala en la distribución de Linux el gestor de comandos bash y valiéndonos de este gestor, inyectamos python3 en la imagen.

Con esta nueva imagen configuramos un docker-compose.yml mapeando dos volúmenes, el primero de estos, mapea la ruta donde residen las páginas estáticas de nuestro desarrollo en el filesystem local, con la ruta donde se alojan las páginas estáticas en nuestra nueva imagen de docker, para que las levante el apache2.

El segundo volumen mapea la carpeta donde se alojan los archivos .py que conforman nuestro programa CGI, con la ruta dentro de la imagen creada, donde se deben alojar los script cgi que luego ejecutara Apache.

Por último se mapeo el puerto 80 dentro del contenedor de docker al puerto 8080 de nuestra PC para acceder al sitio Web desde nuestro navegador con la siguiente URL:

http://localhost:8080/index.html

En cuanto a la aplicación se creó la página estática index.html que ofrece las funcionalidades de cargar un nuevo alumno y modificar un alumno, esta es la única página estática, el resto de las páginas se generan dinámicamente utilizando Python cgi scripts. Toda la funcionalidad CGI, ya sea la generación dinámica de contenido, como la funcionalidad de guardar un nuevo alumno y recuperar alumnos y guardar las modificaciones, guardar y manejar cookies en los siguientes Python scripts:

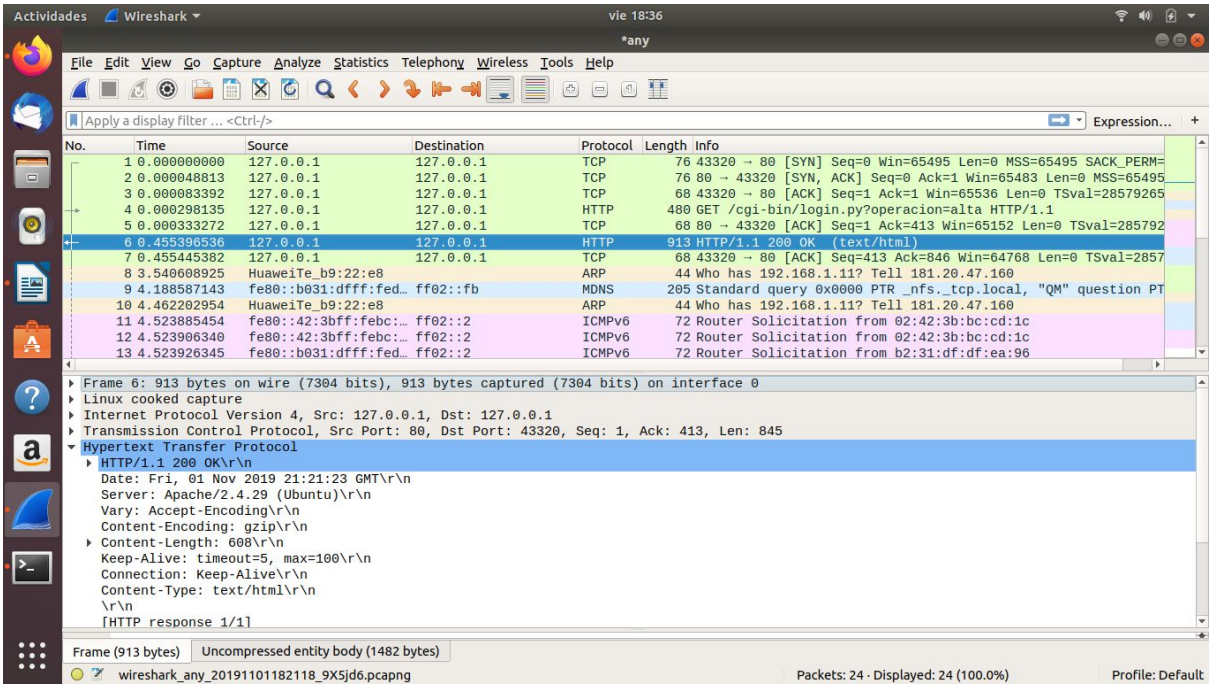
- HttpRequest.py
- HttpResponse.py
- RepoAlumnos.py
- RepoCookies.py

Para almacenar y recuperar los alumnos y las cookies se utilizan archivos delimitados por coma CSV.

2. Desarrolle un experimento (mediante analizador de paquetes) que muestre si el servidor de HTTP agrega o quita información a la que genera un programa CGI. Deberá utilizar un programa CGI conocido para saber exactamente lo que genera y debería mostrar la información que llega del lado del cliente, a nivel de HTTP. Contrastar eso con la salida que produce la invocación local del mismo programa CGI

Para el desarrollo del experimento se utilizó el programa CGI desarrollado en el punto anterior, con la ayuda del analizador de paquetes Wireshark se pudo constatar que el servidor agrega cabeceras HTTP al mensaje de respuesta que envía al cliente Http. Debemos hacer notar aquí, que las únicas cabeceras Http que introducimos voluntariamente en la generación dinámica de páginas, fueron el Content-Type y para el caso de las modificación de alumnos logueados, el seteo de la cookie de session. Pero utilizando el analizador de paquetes se hace evidente que el servidor Http agrega más cabeceras.

En la siguiente captura de pantalla, podemos dar cuenta de la respuesta obtenida por el servidor Http luego de hacer una petición a la funcionalidad de cargar un nuevo alumno, donde del lado del programa cgi renderizamos el formulario de alta de nuevo alumno, seteando únicamente la cabecera Http Content-type y donde podemos ver evidentemente que el servidor agrego Content-Encoding, Content-Length, Keep-Alive, etc.



3. Agregue a la aplicación del ejercicio 1 una reporte básico con un filtro por nombre (buscar los que comiencen con el criterio cargado). Los resultados que resulten del filtro deben mostrarse visualmente diferenciados. ej:

Nombre 1 Núm 1 Sexo 1 Edad 1 ----------
Nombre 2 Núm 2 Sexo 2 Edad 2 ----------
...etc.
COMENTARIO: este ejercicio es para que se vea que el programa CGI debería combinar la generación de HMTL con el resultado de lo que se procesa localmente en el servidor. Y que el HTML generado puede ser "largo". El programa CGI puede estar en el lenguaje que se desee.

Se modifico el desarrollo del punto 1 para crear una lista del repositorio de alumnos, a la misma se la dotó de un campo para filtrar la lista de los alumnos que coinciden con un patrón del nombre.

Para ello modificamos los Python script para dar respuesta a esta nueva funcionalidad.

4. Agregue a la aplicación del ejercicio 1 otra consulta de valores totales por rango de edad (0-20, 20-40, mas de 40). Mostrar el rango y la cantidad de alumnos que hay en ese rango.
COMENTARIO: este ejercicio es para que se vea que la mayor parte del procesamiento en el lado del servidor puede ser solamente para computar valores simples a mostrar, pero que pueden ser resultado de procesamiento relativamente complejo.

Se modifico el desarrollo del punto 1 para crear una lista del repositorio de alumnos, a la misma se la dotó de dos campos para filtrar la lista de los alumnos que coinciden con un rango de edades dado.

Para ello modificamos los Python script para dar respuesta a esta nueva funcionalidad.

5. Tome el código que se adjunta como Anexo 5 del apunte de AJAX y agregue las siguientes mejora de funcionalidad:

a) Agregar un botón a la ventana principal de chat, a efectos que un usuario ingresante pueda registrarse en el sitio. El usuario ingresará su "Nick" y al pulsar el botón, se registrará en un archivo data/usuarios.txt en el servidor.
Hecho.

b) Producido el registro, se devolverá al usuario una lista total de usuarios registrados en una pequeña ventana a la derecha de la ventana común de mensajes y también se devolverá toda la conversación entre usuarios registrada hasta ese momento.
Hecho.

c) Intentar establecer la actualización incremental del chat, mediante el almacenamiento en cada cliente de la última línea de chat enviada.
En el cliente se mantiene una cookie con el valor de la última línea de chat enviada. Al servidor se le pide la diferencia entre los mensajes.

d) Disponer el servidor Web (apache) para este punto en un contenedor docker.

Se generó un Dockerfile para la imagen a montar, luego se generó un .yml para levantar con docker compose el servicio.

```
# Docker file for python simple webservice build

FROM ubuntu:18.04

RUN apt-get update
RUN apt-get -y install apache2

# Python2.7
RUN apt-get -y install python2.7
RUN apt-get -y install vim
RUN apt-get -y install python-pip

# Http settings
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
ENV APACHE_PID_FILE /var/run/apache2.pid
ENV APACHE_RUN_DIR /var/run/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
RUN mkdir -p $APACHE_RUN_DIR $APACHE_LOCK_DIR $APACHE_LOG_DIR

RUN mkdir -p /production/www/chat/cgi-bin
RUN mkdir -p /production/www/chat/public_html

RUN mkdir -p /production/www/alumnos/cgi-bin
RUN mkdir -p /production/www/alumnos/public_html

COPY apache2 /etc/apache2
RUN ln -s /etc/apache2/mods-available/cgi.load /etc/apache2/mods-enabled/cgi.load

EXPOSE 80

ENTRYPOINT [ "/usr/sbin/apache2" ]
CMD ["-D", "FOREGROUND"]
```

docker-compose.yml

```
version: "2"

services:
  cgi-chat:
    container_name: cgi-chat
    build:
      context: .

    volumes:
      - ./chat/public_html:/production/www/chat/public_html
      - ./chat/cgi-bin:/production/www/chat/cgi-bin

    ports:
      - "80:80"
    restart: always
```

El chat tiene un funcionamiento típico, donde un usuario se registra, y puede empezar a intercambiar mensajes con el resto de los usuarios. También posee un botón de logOut para cuando el usuario decida salir del chat. En la izquierda tenemos los usuarios logueados en el sistema, y en la derecha observamos los mensajes que se van registrando por el webservice.

Los mensajes enviados por el usuario logueado serán de color azul, y el resto de los mensajes de color verde.

Internamente, cuando un usuario se registra, el servidor le asigna un ID que se transmite al navegador por medio de una cookie junto al nombre de usuario, y se registra en una base el usuario y el ID que le corresponde.

Una vez logueado podrá empezar a escribir en el área de mensajes, y enviarlos. Cuando el mensaje llegue al servidor, se guarda en una base de datos, quien envió el mensaje, gracias a la cookie con el id, la hora, y el mensaje.

Tanto el registro de usuarios logueados, como el registro de mensajes enviados, son mostrados en pantallas, a través de un set interval, que consta de programar eventos cada un determinado tiempo.

Se envía al servidor los mensajes que se escribieron del lado del cliente a través de una cookie, cuando el servidor recibe el mensaje, compara los mensajes que se escribieron en el cliente, y los mensajes que tiene el como servidor, la diferencia resultante de la comparación, se le manda al cliente, para ser concatenados con los mensajes escritos previamente.

La lista de usuarios logueados, se refresca completa debido a que podríamos tener la misma cantidad de usuarios, pero que sean distintos. Se le solicita al servidor la lista completa de usuarios y se reescribe en pantalla.

Todas las consultas al servidor están escritas en un index.js, através de ajax se logran las peticiones al servidor asincrónicamente, y en un segundo plano, para evitar que la pantalla sea redireccionada, y poder actualizar la informacion presentada.

Escribimos 2 cgi’s:

- users.py: Encargado de las operaciones de usuarios.
- messages.py: Encargado de las operaciones de mensajes.

6. DFS:

a) Configurar NFS y efectuar un compartimiento de archivos entre un Servidor y un Cliente Linux. Describir la operatoria en el informe.

Para llevar acabo este punto, previamente descargamos e instalamos la herramienta VirtualBox con la cual ejecutamos la imagen de un Ubuntu Versión 12.10 para poder ejecutar las pruebas del lado del cliente. Como servidor se utilizó la PC de Federico ejecutando la versión 18.04d Ubuntu, en la que a su vez se ejecutó la Virtual Machine.

Instalación de NFS en la PC Servidor

El primer paso fue instalar NFS en una PC con Ubuntu 18.04 para compartir recursos con los clientes, esto lo hicimos instalando un paquete llamado nfs-kernel-server con el siguiente comando:

sudo apt install nfs-kernel-server

Creamos la carpeta compartir en la ruta /home/federico usando el siguiente comando:

sudo mkdir /compartir

sudo chown nobody:nogroup /compartir

Habilitar exports para NFS

Los 'exports' son los recursos que queremos compartir por medio de NFS y se definen en el archivo /etc/exports, estos recursos son rutas o directorios que podemos compartir por medio del servidor NFS hacia otros equipos Linux.

Para agregar una ruta y compartirla por medio de NFS se tiene que editar ese archivo y agregar parámetros , los parámetros puede ser:

- ro: Limita el acceso a solo lectura por los clientes.
- rw: Los clientes tendrán permiso de lectura y escritura al directorio
- no_root_squash: De forma predeterminada, cualquier solicitud de archivo realizada por el usuario root en la máquina cliente se trata como si fuera realizada por el usuario nobody en el servidor. Si se selecciona no_root_squash, la raíz en la máquina cliente tendrá el mismo nivel de acceso a los archivos en el sistema como root en el servidor. Esto puede tener serias implicaciones de seguridad, aunque puede ser necesario si desea realizar algún trabajo administrativo en la máquina cliente que involucre a los directorios exportados. No se debe especificar esta opción sin una buena justificación.
- no_subtree_check: Si solo se exporta una parte de un volumen, una rutina llamada comprobación de subárbol verifica que un archivo que se solicita al cliente se encuentre en la parte apropiada del volumen. Si se exporta todo el volumen, deshabilitar esta comprobación acelerará las transferencias.

- sync: Los cambios en los datos de los archivos se realizan en el disco de forma inmediata, lo que tiene un impacto en el rendimiento, pero es menos probable que provoque una pérdida de datos si el sistema se reinicia o se apaga.
- async: Los cambios en los datos del archivo se hacen inicialmente en la memoria. Esto acelera el rendimiento, pero es más probable que provoque la pérdida de datos en caso de falla o apagado repentino del sistema.

Acto seguido editamos el archivo /etc/exports y agregamos el directorio y las opciones para compartir la ruta /media01/videos/accion en lectura y escritura con permisos para un rango de IPs que estén entre 10.107.31.0 y 10.107.31.255 :

/compartir 10.0.2.0/24(rw, sync, no_subtree_check)ss

Modificado el archivo, el mismo se debió procesar y luego reiniciar el Kernel de NFS utilizando los siguientes comandos:

sudo exportfs -a

sudo systemctl restart nfs-kernel-server

Y con esto terminamos la configuración del servidor de archivos, si deseamos ver las carpetas compartidas lo podemos hacer con el comando:

sudo exportfs -v

Instalar NFS en el cliente

Para habilitar el soporte para NFS en la PC cliente dolo se debió ejecutar el siguiente comando:

sudo apt-get install nfs-common

Montar la carpeta por NFS en el cliente

Para montar la carpeta compartida por NFS ejecutamos el siguiente comando:

sudo mount -t nfs 10.107.31.2:/home/Federico/Downloads /home/Federico/ Downloads

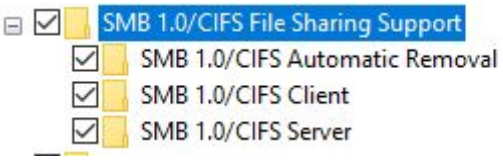
Para verificar que todo funcionaba ok fuimos a la ruta donde montamos el recurso NFS y listamos los archivos ejecutando el comando ls.

b) Configurar Samba y efectuar un compartimiento de archivos entre un Windows y un Cliente Linux. Describir la operatoria en el informe.

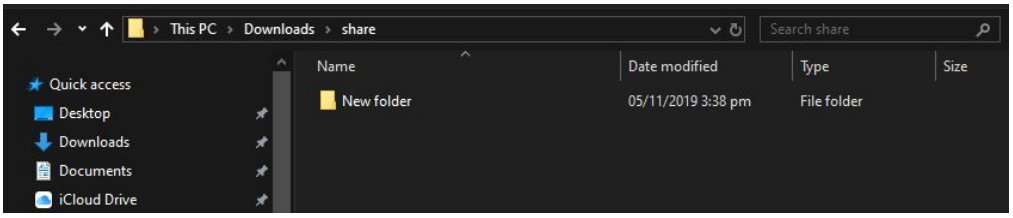
Samba utiliza el protocolo SMB para lograr la interacción usamos el cliente de MacOS.

Documentacion: https://es.wikipedia.org/wiki/Server_Message_Block

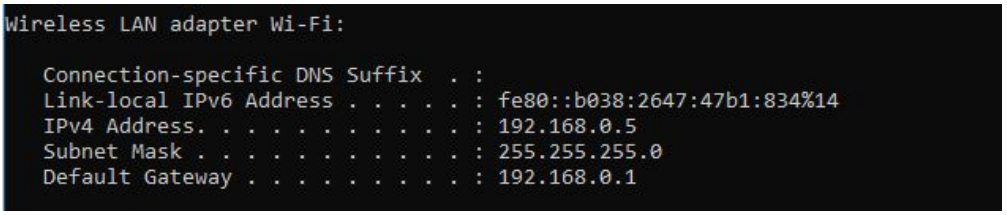
En windows: Samba viene por defecto en windows 10, pero se debe activar la opción en características de windows.



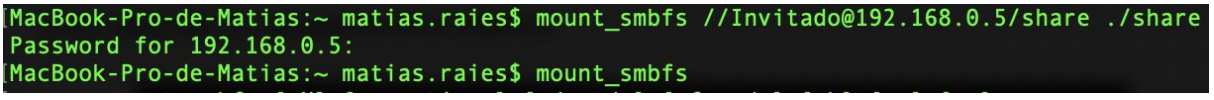
Desde windows se compartió la carpeta share y se creo un usuario invitado para darle permisos a la carpeta y que pueda ser accedido por protocolo smb.



Verificamos la dirección IP de windows para saber donde se aloja el servidor samba.



Con la dirección IP del servidor samba en windows, logramos acceder desde MacOS con el cliente smb.



Montamos el recurso compartido y accedemos a él.

```
MacBook-Pro-de-Matias:share matias.raies$ pwd
/Users/matias.raies/share
MacBook-Pro-de-Matias:share matias.raies$ ls
New folder
```

7. DNS:

Objetivo: Observar las llamadas implícitas a DNS y ponderar tiempos de acceso. Efectuar el siguiente experimento, verificando información con la ayuda de un analizador de protocolo:

a) Desde el browser haga una conexión a una página WEB de argentina, que haya utilizado hace mucho tiempo o que no haya utilizado (un diario, un blog, etc.). Obtenga la diferencia temporal entre la consulta que se observa en el analizador y su respuesta.

https://www.blaque.com.ar

790	21.170304	192.168.0.11	1.1.1.1	DNS	77	Standard query 0x7a16 A www.blaque.
791	21.402520	1.1.1.1	192.168.0.11	DNS	159	Standard query response 0x7a16 A ww
792	21.403757	192.168.0.11	104.17.214.236	TCP	78	61323 → 80 [SYN] Seq=0 Win=65535 Le
793	21.436509	104.17.214.236	192.168.0.11	TCP	66	80 → 61323 [SYN, ACK] Seq=0 Ack=1 W
794	21.436568	192.168.0.11	104.17.214.236	TCP	54	61323 → 80 [ACK] Seq=1 Ack=1 Win=26
795	21.436952	192.168.0.11	104.17.214.236	HTTP	413	GET / HTTP/1.1
796	21.472709	104.17.214.236	192.168.0.11	TCP	60	80 → 61323 [ACK] Seq=1 Ack=360 Win=
797	21.482810	104.17.214.236	192.168.0.11	HTTP	362	HTTP/1.1 301 Moved Permanently
798	21.482975	192.168.0.11	104.17.214.236	TCP	54	61323 → 80 [ACK] Seq=360 Ack=309 Wi
799	21.490473	192.168.0.11	104.17.214.236	TCP	78	61324 → 443 [SYN] Seq=0 Win=65535 L
800	21.520243	104.17.214.236	192.168.0.11	TCP	66	443 → 61324 [SYN, ACK] Seq=0 Ack=1
801	21.520302	192.168.0.11	104.17.214.236	TCP	54	61324 → 443 [ACK] Seq=1 Ack=1 Win=2
802	21.520848	192.168.0.11	104.17.214.236	TLSv1…	571	Client Hello
803	21.553142	104.17.214.236	192.168.0.11	TCP	60	443 → 61324 [ACK] Seq=1 Ack=518 Win
804	21.558916	104.17.214.236	192.168.0.11	TLSv1…	1514	Server Hello, Change Cipher Spec

Podemos observar como en el paquete Standard query el navegador le pide al DNS la IP de www.blaque.com.ar, el paquete Standard query response contendrá la IP de esa dirección.

b) Haga lo mismo con una página no frecuentada por Ud. que se supone que está situada en un servidor europeo o de Asia. (la página oficial del grupo Nightwish, o algún diario galés, etc.). Compare sus conclusiones con las obtenidas en el caso a).

https://www.ecured.cu

58	10.346241	192.168.0.11	1.1.1.1	DNS	73	Standard query 0x0d19 A www.ecured.cu
59	10.380050	1.1.1.1	192.168.0.11	DNS	89	Standard query response 0x0d19 A www.ecured.cu
60	10.381440	192.168.0.11	190.92.127.40	TCP	78	61360 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
61	10.751972	190.92.127.40	192.168.0.11	TCP	74	80 → 61360 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len
62	10.752040	192.168.0.11	190.92.127.40	TCP	66	61360 → 80 [ACK] Seq=1 Ack=1 Win=131712 Len=0 T
63	10.752487	192.168.0.11	190.92.127.40	HTTP	421	GET / HTTP/1.1
64	11.059163	190.92.127.40	192.168.0.11	TCP	66	80 → 61360 [ACK] Seq=1 Ack=356 Win=30208 Len=0
65	11.059166	190.92.127.40	192.168.0.11	HTTP	523	HTTP/1.1 301 Moved Permanently
66	11.059217	192.168.0.11	190.92.127.40	TCP	66	61360 → 80 [ACK] Seq=356 Ack=458 Win=131264 Len
67	11.105828	192.168.0.11	190.92.127.40	TCP	78	61361 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=146
68	11.469545	190.92.127.40	192.168.0.11	TCP	74	443 → 61361 [SYN, ACK] Seq=0 Ack=1 Win=28960 Le
69	11.469640	192.168.0.11	190.92.127.40	TCP	66	61361 → 443 [ACK] Seq=1 Ack=1 Win=131712 Len=0
70	11.470501	192.168.0.11	190.92.127.40	TLSv1	583	Client Hello
71	11.776606	190.92.127.40	192.168.0.11	TCP	66	443 → 61361 [ACK] Seq=1 Ack=518 Win=30208 Len=0
72	11.776611	190.92.127.40	192.168.0.11	TLSv1…	1514	Server Hello

A pesar de estar más lejos el servidor de la página .cu, solo se observa una consulta al DNS, y con un tiempo de demora muy similar al del primer request.

c) Vuelva nuevamente a invocar la página seleccionada en a), (que se supone ya reside en su caché). Compare resultados.

1	0.000000	192.168.0.11	104.17.215.236	TLSv1…	519	Application Data
2	0.037426	104.17.215.236	192.168.0.11	TCP	60	443 → 61522 [ACK] Seq=1 Ack=466 Win=38 Len=0
3	0.061197	104.17.215.236	192.168.0.11	TCP	1514	443 → 61522 [ACK] Seq=1 Ack=466 Win=38 Len=1460 [TCP se
4	0.061529	104.17.215.236	192.168.0.11	TCP	1514	443 → 61522 [ACK] Seq=1461 Ack=466 Win=38 Len=1460 [TCP
5	0.061535	104.17.215.236	192.168.0.11	TLSv1…	1385	Application Data
6	0.061609	192.168.0.11	104.17.215.236	TCP	54	61522 → 443 [ACK] Seq=466 Ack=2921 Win=4073 Len=0
7	0.061610	192.168.0.11	104.17.215.236	TCP	54	61522 → 443 [ACK] Seq=466 Ack=4252 Win=4052 Len=0
8	0.062591	104.17.215.236	192.168.0.11	TCP	1514	443 → 61522 [ACK] Seq=4252 Ack=466 Win=38 Len=1460 [TCP
9	0.062738	192.168.0.11	104.17.215.236	TCP	54	61522 → 443 [ACK] Seq=466 Ack=5712 Win=4096 Len=0
10	0.063120	104.17.215.236	192.168.0.11	TCP	1514	443 → 61522 [ACK] Seq=5712 Ack=466 Win=38 Len=1460 [TCP
11	0.064033	104.17.215.236	192.168.0.11	TLSv1…	1385	Application Data
12	0.064118	192.168.0.11	104.17.215.236	TCP	54	61522 → 443 [ACK] Seq=466 Ack=8503 Win=4075 Len=0
13	0.065512	104.17.215.236	192.168.0.11	TCP	1514	443 → 61522 [ACK] Seq=8503 Ack=466 Win=38 Len=1460 [TCP
14	0.065991	104.17.215.236	192.168.0.11	TCP	1514	443 → 61522 [ACK] Seq=9963 Ack=466 Win=38 Len=1460 [TCP
15	0.066031	192.168.0.11	104.17.215.236	TCP	54	61522 → 443 [ACK] Seq=466 Ack=11423 Win=4073 Len=0
16	0.066912	104.17.215.236	192.168.0.11	TLSv1…	1385	Application Data

Al escribir la dirección nuevamente en el navegador, no se observaron paquetes DNS, directamente comenzó a interactuar con el sitio. En la primera interacción también se vio el intercambio de paquetes para el protocolo TLS.

d) Aporte las conclusiones que puede sacar del experimento.

Del experimento se observa que, al cargar cualquier sitio primero se debe consultar la IP del sitio al servidor DNS. Una vez se tenga la IP se podrán enviar request al sitio.

En caso de contener el sitio en la caché, primero no se deberá consultar al servidor DNS y al menos en este caso, no se deberá volver a intercambiar los mensajes de protocolos de seguridad.