

MEMORIA DE PRÁCTICAS RED DE DATOS



**Escola Tècnica Superior
d'Enginyeria**



UNIVERSITAT ROVIRA I VIRGILI

Alberto Flores Pastor

Jorge Pérez Álvarez

2021/2022

Doble grado BIOGEI

ÍNDICE

Práctica 1: Rendimiento.....	3
Práctica 2: Análisis RFC.....	7
Práctica 3: Comunicación aplicaciones.....	10
Práctica 4: Simulación de redes.....	14
Práctica 5: Montaje de LAN.....	19
Práctica 6: Protocolos Wireshark.....	24
Reflexiones.....	34

PRÁCTICA 1:

RENDIMIENTO

En esta práctica analizaremos el rendimiento de una red gracias a la herramienta ping. Esta nos permite determinar si un destino está respondiendo y si es accesible desde el lugar donde se está haciendo.

-Para realizar esta primera parte realizaremos pruebas de ping a diferentes dominios o IP (indistintamente) mediante el envío de paquetes ICMP. Para ello, usaremos el comando ping de Linux, obteniendo los siguientes resultados:

Dominio (IP)	Windows	Linux
Localhost(10.21.1.6)/(10.21.1.7)	<1ms	<1ms
Campusvirtual.urv.cat(84.88.192.218)	Tiempo de espera agotado	Tiempo de espera agotado
Rediris.es(130.206.13.20)	12ms	12,5ms
8.8.8.8(8.8.8.8)	13ms	12ms
Example.com(93.184.216.34)	108ms	108ms

En el localhost se observa un ping tan bajo debido a que se está tratando una red local, por tanto, no tiene que pasar el paquete por ningún rúter.

En campusvirtual.urv.cat creemos que no responde debido a que es un dominio el cual su rúter está configurado de tal forma que no permite la conexión, haciéndolo ver inactivo y por tanto, evitando una posible ruta de retorno al equipo.

En el caso de rediris.es al ser un dominio español no pasará por demasiados rúters para establecer una conexión.

En el de Google, al ser un dominio tan globalizado su acceso es bastante accesible y de ahí un tiempo de espera tan bajo.

Y finalmente para example.com al ser un dominio de fuera de nuestro territorio debe pasar por más rúters para llegar a producirse la conexión, por eso el tiempo es bastante superior.

A continuación, mediante la web pingtestlive.com podremos observar la latencia que encontraremos desde nuestra ubicación a los diferentes servidores repartidos por todo el mundo. A diferencia del comando anterior, esta página no mide el round trip time, sino que las mira mediante la cabecera

HTTP. En la primera imagen se observa el test que hemos realizado para el videojuego Fortnite y en la segunda se observa los resultados obtenidos al realizarlo a Amazon Web Services.

?	Location:	Latency:	Average:	Status:
<input checked="" type="checkbox"/>	Ohio (U.S.)	1144	1144	Done
<input checked="" type="checkbox"/>	Northern Virginia (U.S.)	164	164	Done
<input checked="" type="checkbox"/>	Northern California (U.S.)	177	177	Done
<input checked="" type="checkbox"/>	Oregon (U.S.)	201	201	Done
<input checked="" type="checkbox"/>	Tokyo	259	259	Done
<input checked="" type="checkbox"/>	Seoul, South Korea	271	271	Done
<input checked="" type="checkbox"/>	Mumbai, India	147	147	Done
<input checked="" type="checkbox"/>	Singapore	182	182	Done
<input checked="" type="checkbox"/>	Sydney, Australia	297	297	Done
<input checked="" type="checkbox"/>	Canada	120	120	Done
<input checked="" type="checkbox"/>	Beijing, China	218	218	Done
<input checked="" type="checkbox"/>	Frankfurt, Germany	62	62	Done
<input checked="" type="checkbox"/>	Ireland	77	77	Done
<input checked="" type="checkbox"/>	London	55	55	Done
<input checked="" type="checkbox"/>	Paris, France	54	54	Done
<input checked="" type="checkbox"/>	Sao Paulo, Brazil	226	226	Done

?	Location:	Latency:	Average:	Status:
<input checked="" type="checkbox"/>	Ohio (U.S.)	127	127	Done
<input checked="" type="checkbox"/>	Northern Virginia (U.S.)	106	106	Done
<input checked="" type="checkbox"/>	Northern California (U.S.)	180	180	Done
<input checked="" type="checkbox"/>	Oregon (U.S.)	185	185	Done
<input checked="" type="checkbox"/>	Tokyo	266	266	Done
<input checked="" type="checkbox"/>	Seoul, South Korea	272	272	Done
<input checked="" type="checkbox"/>	Mumbai, India	124	124	Done
<input checked="" type="checkbox"/>	Singapore	195	195	Done
<input checked="" type="checkbox"/>	Sydney, Australia	316	316	Done
<input checked="" type="checkbox"/>	Canada	127	127	Done
<input checked="" type="checkbox"/>	Beijing, China	280	280	Done
<input checked="" type="checkbox"/>	Frankfurt, Germany	50	50	Done
<input checked="" type="checkbox"/>	Ireland	61	61	Done
<input checked="" type="checkbox"/>	London	50	50	Done
<input checked="" type="checkbox"/>	Paris, France	42	42	Done

Como se puede observar al probar un servidor más próximo geográficamente la latencia que se obtiene es menor debido a que debe pasar por menos rúters para completar la conexión.

Para la prueba de velocidad usaremos la página web speedtest.net, la cual nos permite valorar el rendimiento de una conexión a Internet gracias a las medidas en relación con la data rate y a las

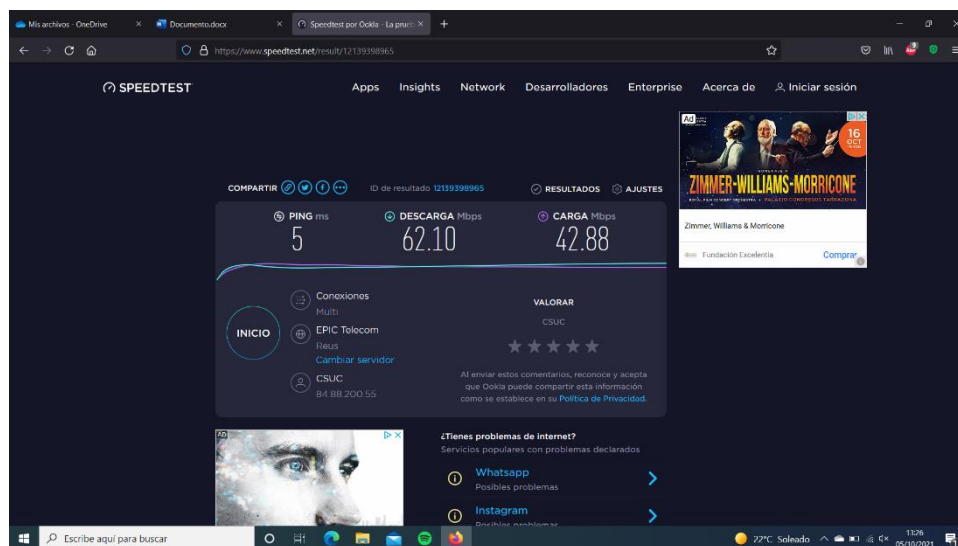
latencias. En un primer momento debemos realizar una prueba de velocidad todo el mundo del laboratorio al mismo tiempo contra el mismo servidor (en nuestro caso Barcelona CSUC) y también realizar esta misma prueba solo desde una máquina. Los resultados obtenidos fueron los siguientes:

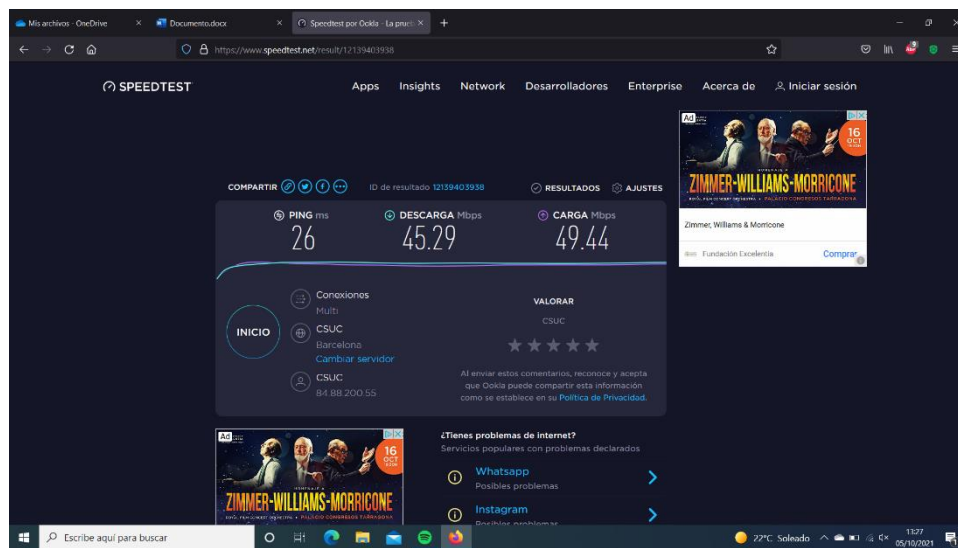
Prueba individual: ping 2 bajada 93.73 subida 94.58

Prueba conjunta: ping 2 bajada 93.17 subida 93.70

Como se puede ver, efectivamente el rendimiento de nuestra red ha bajado. Aun así, no es prácticamente apreciable. Lógicamente, al haber más máquinas intentando establecer conexión con el mismo servidor, el tráfico de datos de este se ve más saturado. Aun así, no creemos que sean suficientes máquinas como para llegar a producir una saturación tan grande a un servidor de alta capacidad de datos como al que estamos realizando la prueba.

-Para finalizar esta práctica, repetiremos las pruebas desde casa, Una la haremos contra el mismo servidor del apartado anterior y la otra contra el servidor que según la página es óptimo para nosotros. Los resultados obtenidos muestran que efectivamente la latencia es considerablemente mejor en el más cercano, pero que los valores de descarga y carga tampoco son tan dispares el uno del otro.





PRÁCTICA 2:

ANÁLISIS DE RFC

En esta segunda práctica analizaremos los diferentes protocolos relacionados con Internet. Estos estarán descritos en forma RFC (Request For Comments). Una RFC es un documento numérico en el que se describen y definen protocolos, conceptos, métodos y programas de Internet. La gestión de estos se realiza a través de IETF.

Cada RFC tiene un título y un número asignado, que no puede repetirse ni eliminarse, aunque el documento se quede obsoleto. Además, las RFC se redactan en inglés según una estructura específica y en formato de texto ASCII. El estado de una RFC se clasifica de dos formas independientes. La primera establece el grado de madurez de la especificación y la propuesta puede ser considerada respecto a ella como por ejemplo estándar, experimental, informativo... Y La segunda forma de clasificación define el grado de cumplimiento necesario de la norma dentro de Internet.

-Como parte a tratar en esta práctica debemos seleccionar una RFC cualquiera y responder una serie de preguntas. Nosotros hemos escogido el RFC 6409.

-¿Quién es el autor o grupo de autores del documento?

R. Gellens: este científico ha participado y participa activamente en la elaboración de RFC desde el 1997, incluyendo 32 participaciones como autor o coeditor de estos.

J. Klensin: John C. Klensin es un científico político y un profesional de las ciencias de la computación que participa activamente en temas relacionados con Internet. Su carrera incluye 30 años como científico investigador principal en el MIT. Klensin es autor o coeditor de más de 40 RFC y se ha desempeñado como director del Área de Aplicaciones de IETF entre 1993-1995.

-¿En qué año se ha publicado?

Este RFC fue publicado en noviembre del 2011.

-¿Qué documentos actualiza / por qué documento está actualizado?

Actualiza: RFC 4409

Está actualizado por: RFC 8314

-De qué apartados se compone (los más importantes, no es necesario ser exhaustivos)

Este RFC separa el envío de mensajes de la retransmisión de mensajes, lo que permite que cada servicio pueda operar de acuerdo con sus propias reglas (por seguridad, política, etc.), y especifica qué acciones deben tomarse por un servidor de envío. El RFC está compuesto por una introducción donde se explica en que se basa este nuevo protocolo y una estructura central donde detalla varias partes. Entre ellas destacan el proceso de envío de mensajes y quienes estarían autorizados para realizarlo; acciones que son obligatorias, recomendadas o opcionales para beneficiarse de este servicio; como se realiza la interacción con extensiones SMTP; las modificaciones que se hicieron a los mensajes y finalmente algunas consideraciones.

-¿Hay algún esquema o dibujo?

En la siguiente tabla, se enumeran las extensiones de Standard Track y Experimental SMTP cuyos documentos no especifican explícitamente su aplicabilidad a este protocolo. Se enumeran la palabra clave EHLO, el nombre, una indicación del uso de la extensión en el puerto de envío y una referencia.

Como se puede observar el esquema al igual que los dibujos están configurados en formato texto plano.

Keyword	Name	Sub- mission	Reference
PIPELINING	Pipelining	SHOULD	[PIPELINING]
ENHANCEDSTATUSCODES	Enhanced Status Codes	SHOULD	[CODES-EXTENSION]
ETRN	Extended Turn	MUST NOT	[ETRN]
...	Extended Codes	SHOULD	[SMTP-CODES]
DSN	Delivery Status Notification	SHOULD	[DSN]
SIZE	Message size	MAY	[SIZE]
...	521 reply code	MUST NOT	[REPLY-521]
CHECKPOINT	Checkpoint/Restart	MAY	[CHECKPOINT]
BINARYMIME	Binary MIME	MAY	[CHUNKING]
CHUNKING	Chunking	MAY	[CHUNKING]
8BITMIME	Use 8-bit data	SHOULD	[RFC6152]
AUTH	Authentication	MUST	[SMTP-AUTH]
STARTTLS	Start TLS	MAY	[START-TLS]
NO-SOLICITING	Notification of no soliciting	MAY	[RFC3865]
MTRK	Message Tracking	MAY	[MSG-TRACK]
ATRN	On-Demand Relay	MUST NOT	[RFC2645]
DELIVERBY	Deliver By	MAY	[RFC2852]
CONPERM	Content Conversion Permission	MAY	[RFC4141]
CONNEN	Content Conversion Negotiation	MAY	[RFC4141]

Table 1

PRÁCTICA 3:

COMUNICACIÓN DE

APLICACIONES

Definición, funcionalidad y tipos de sockets

El objetivo de esta práctica es entender la comunicación de aplicaciones mediante los sockets y también implementar una aplicación cliente y un servidor mediante el lenguaje Java.

Un socket se puede definir como un tipo de software que actúa como un punto de comunicación de red bidireccional entre dos agentes (servidor y cliente) por el cual se puede emitir o recibir información. Este se define con la unión de una dirección IP con un puerto, como se puede ver a continuación en el ejemplo de nuestra implementación:

```
Socket s = new Socket("127.0.0.1", 8888);
```

La comunicación entre procesos a través de sockets se basa en la filosofía CLIENTE-SERVIDOR. Esto permite a los programadores estar más tranquilos, ya que simplifican el funcionamiento de un programa ya que puede fiarse de que el transporte de mensajes se da correctamente gracias al sistema operativo y ellos solo tendrán que manipular las funciones del socket.

Existen varios tipos:

- Sockets de datagramas: proporciona un punto sin conexión para enviar y recibir paquetes de datos.
- Tomas de corriente crudas: esta toma permite el acceso al proveedor de transporte subyacente.
- Sockets secuenciados: este tipo de socket permite a los usuarios manipular las cabeceras del protocolo de paquetes de secuencia (SPP) o del protocolo de datagramas de Internet (IDP) en un paquete o incluso en un grupo de paquetes.
- Tomas de corriente: este tipo de zócalo se basa en TCP para la transmisión de datos.

Nuestra implementación

Nuestro programa se basará en un juego tradicional con canicas. El juego se resume en que comenzaremos con 10 canicas al igual que la máquina. Si nos toca jugar, tendremos que apostar que cantidad de canicas ganará o perderá en función de si predices bien o no el número de canicas que sacará el adversario (par o impar) y la predicción de lo que crees que sacará. Por lo contrario, en el turno siguiente juega la máquina y tú deberás elegir si sacas un numero par o impar para que adivine la máquina. Se va jugando hasta que uno de los dos se quede sin canicas.

Como característica, al hacerlo concurrente lo hemos realizado con threads. Estos threads, creados por el servidor, tendrán el control del juego, recibiendo y enviando mensajes al cliente. Otra particularidad, es que cuando le toca jugar a la máquina lo hacemos mediante números aleatorios. En

nuestro caso, el cliente tendrá que escribir todos los parámetros que le pidamos en función del turno (controlado mediante una variable). Ellos serán la apuesta y la predicción si le toca jugar o el número de canicas que sacará si le toca jugar a la máquina. El thread analizará y aplicará estos campos y dará el número de canicas actual del jugador y una frase que sirve para información para el cliente o como centinela de que el juego ha acabado.

Nosotros hemos implementado un servidor concurrente TCP. Hemos decidido realizarlo orientado a la conexión porque necesitamos recibir los paquetes en orden, ya que lo que suceda en un turno anterior acaba afectando al turno actual. Y también lo hemos decidido hacer concurrente porque así podremos atender a varios clientes a la vez. También cabe destacar que asignamos como IP 127.0.0.1 (localhost) y de puerto hemos usado uno arbitrario de los registrados, el 8888.

Juego de pruebas

Probamos de que el servidor funciona correctamente cuando conectamos 1 cliente:

```
¡Servidor funcionando...  
1 clientes conectados  
Recibiendo del cliente 1
```

Probamos de que el servidor funciona correctamente cuando conectamos 3 cliente:

```
¡Servidor funcionando...  
1 clientes conectados  
Recibiendo del cliente 1  
2 clientes conectados  
Recibiendo del cliente 2  
3 clientes conectados  
Recibiendo del cliente 3
```

Jugamos con 1 cliente:

```
¡Bienvenido al juego de las canicas!  
En este juego comenzaras con 10 canicas al igual que la maquina, por turnos  
tendras que apostar que cantidad de canicas ganaras o perderas en funcion de si predecies bien que sacara el adversario (par o impar)  
o por lo contrario deberas elegir si sacas un numero par o impar para que adivine la maquina, asi hasta que uno de los dos se quede sin canicas  
Te toca predecir y apostar:|  
¿Crees que tu rival va a sacar par o impar?Inserte P para par o I para impar
```

Escribimos letra incorrecta:

```
¿Crees que tu rival va a sacar par o impar?Inserte P para par o I para impar  
e  
No ha insertado un valor correcto, escriba P o I
```

Insertamos correctamente la predicción, pero no la apuesta:

```
p
Inserta la apuesta(entre 1 y 10 canicas):
0
Error, escribe una apuesta correcta entre 1 y 10 canicas
```

Escribimos una apuesta válida:

```
Error, escribe una apuesta correcta entre 1 y 10 canicas
5
La partida no ha acabado, el jugador tiene 15 canicas y la maquina tiene 5 canicas
Te toca sacar un numero de canicas para que la maquina adivine:
Inserta el numero de canicas que quieres sacar(entre 1 y 15 canicas):
```

Como hemos visto, hemos acertado la predicción y por tanto, le quitamos a la máquina las canicas que habíamos apostado. Ahora es el turno de la máquina y debemos escribir en función de las que tenemos si queremos sacar un número de canicas par o impar. Ponemos un valor incorrecto:

```
Inserta el numero de canicas que quieres sacar(entre 1 y 15 canicas):
16
Error, escribe un numero correcto entre 1 y 15 canicas
```

Insertamos un valor correcto:

```
Error, escribe un numero correcto entre 1 y 15 canicas
13
La partida no ha acabado, el jugador tiene 13 canicas y la maquina tiene 7 canicas
Te toca predecir y apostar:
¿Crees que tu rival va a sacar par o impar?Inserte P para par o I para impar
```

Como se puede intuir, la máquina ha acertado y por tanto perdemos la apuesta que ha realizado la máquina. Ahora continuaremos ejecutando hasta que uno de los dos gane:

```
¿Crees que tu rival va a sacar par o impar?Inserte P para par o I para impar
p
Inserta la apuesta(entre 1 y 13 canicas):
13
Enhorabuena has ganado!!!
GRACIAS POR JUGAR!!!
```

Como se puede ver al quedarse la máquina sin canicas, nos sale que hemos ganado y la comunicación con este cliente se interrumpe.

Se ha realizado la prueba con varios clientes jugando simultáneamente y no ha habido ningún tipo de error en la comunicación, ya que estamos usando un servidor TCP.

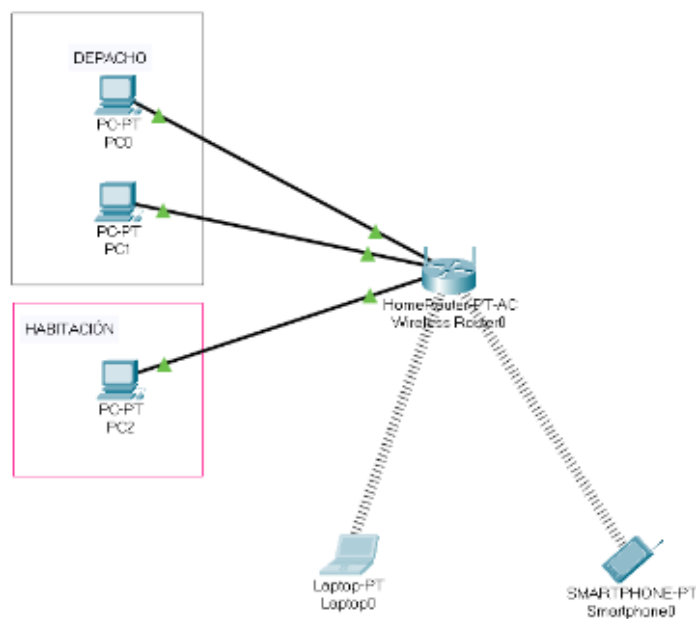
PRÁCTICA 4:

SIMULACIÓN DE REDES

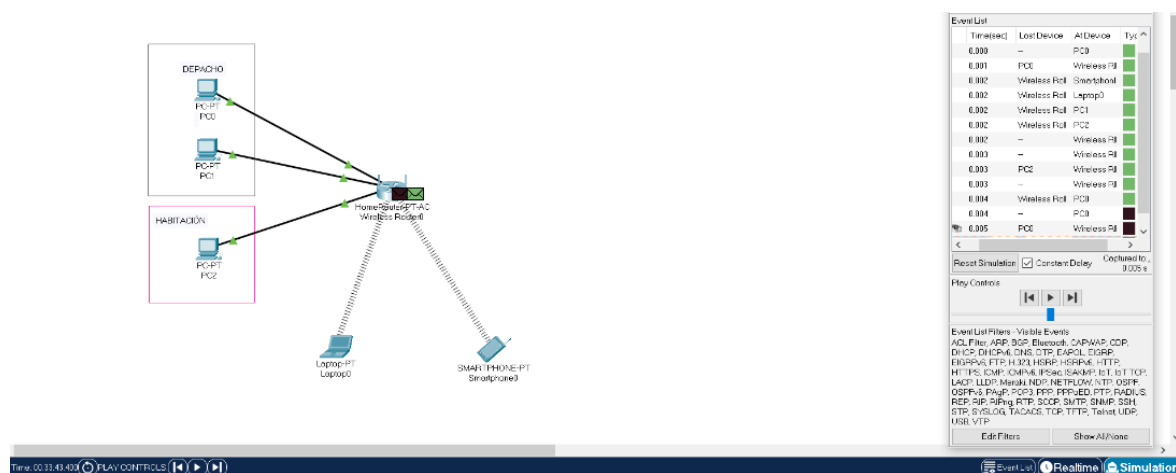
CON PACKET TRACER

Para iniciar la práctica 4 debemos, empleando Packet Tracer, simulamos una red local de una casa. Para ello disponemos de 1 router, 2 ordenadores conectados con IPs asignadas por DHCP, 1 ordenador conectado a la dirección IP estática 192.168.3.31 todos estos conectados por cables y, finalmente, 2 dispositivos, un portátil y un smartphone ambos conectados por wifi y con IP dinámica.

Una vez implementada y modificadas cada una de las características de los dispositivos conectados a la red, nos quedaría un resultado como el que se observa en la siguiente captura:



Tras realizar la instalación adecuada, mediante una simulación del envío de un paquete PCU comprobamos que la instalación esté correctamente realizada.



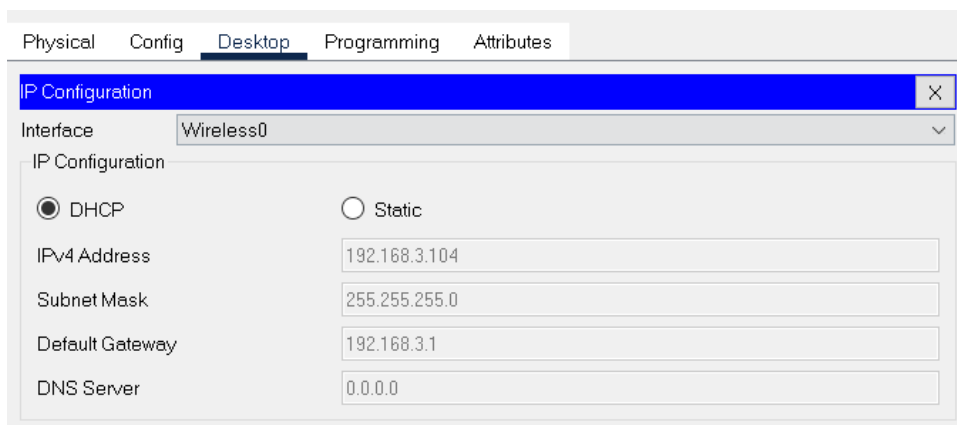
A partir de aquí podremos realizar las diferentes tareas que nos solicita la práctica:

Tasca1:

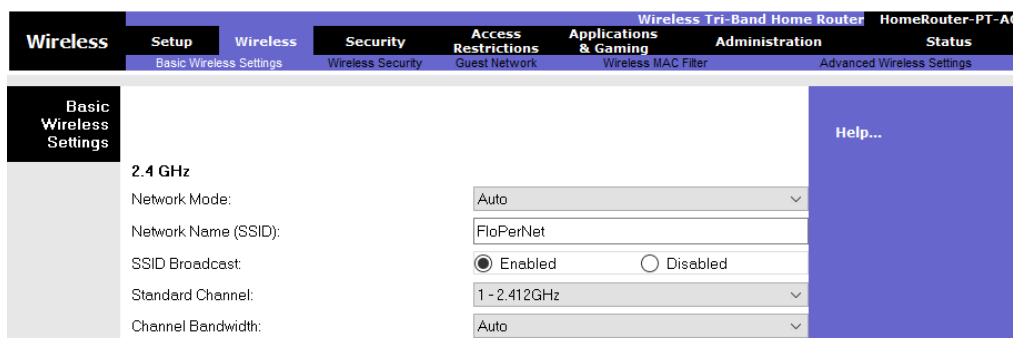
Nos piden configurar un escenario en el que enviamos un paquete desde el PC0 a el portátil y observemos los resultados obtenidos.

Al hacerlo observamos que el paquete no se ha transmitido de forma correcta, este ha viajado desde el PC0 al servidor de la red local pero no ha llegado hasta el portátil. Esto se debe a que el portátil y el PC0 están en subredes distintas, ya que al cambiar las IP estática del router, no se cambiaron las IPs dinámicas de ambos equipos.

Para poder conectarlas, hemos modificado la ip del portátil y del smartphone con las nuevas características del router. Para ello hemos pasado la configuración del IP de ambos dispositivos a estática y luego nuevamente a automática, obteniendo la dirección que se observa en la captura.



Ahora ya puede comunicarse el PC0, pero de una forma no segura, así que el siguiente paso será configurar el SSID y establecer una contraseña a nuestra red. En nuestro caso nombraremos a la red como "FloPerNet" y a la contraseña como "AlbertoJorge".





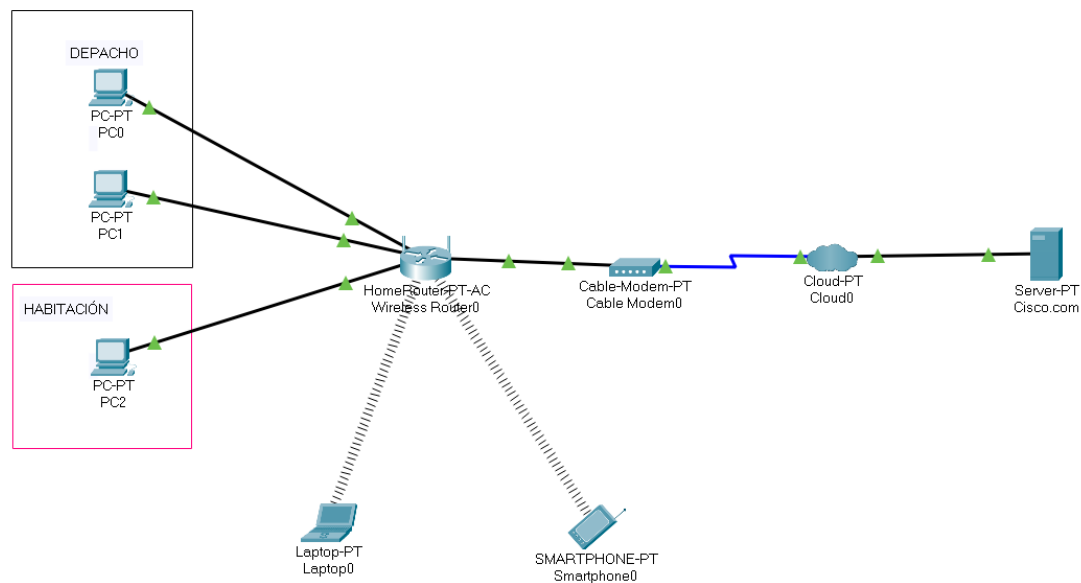
Una vez establecido esto, realizamos la simulación y comprobamos que el transporte del paquete UCP se realiza correctamente dentro de la Red entre PC0 y el portátil.

Vis.	Time(sec)	Last Device	At Device	Type
	0.000	—	PC0	
	0.001	PC0	Wireless Router	
	0.002	Wireless Router	Laptop0	
	0.002	Wireless Router	Smartphone0	
	0.006	—	Laptop0	
	0.007	Laptop0	Wireless Router	
	0.008	Wireless Router	PC0	

En la simulación se observó el correcto transporte

Tasca2:

Para esta segunda tarea vamos a conectarnos un servidor remoto de Cisco.com mediante internet, para ello deberemos añadir un cable-modem, la conexión a la nube y un servidor. Todos estos los configuraremos con las características necesarias para establecer conexión con la nube y guardar en el servidor las características del dominio Cisco.com, obteniendo como resultado el que se observa en la captura.



Una vez configurado, realizamos desde el PC0 un ping hacia Cisco.com para comprobar que se ha realizado de forma correcta, obteniendo la siguiente respuesta y observando la correcta conexión.

```
C:\>ping Cisco.com

Pinging 208.67.220.220 with 32 bytes of data:

Reply from 208.67.220.220: bytes=32 time=2ms TTL=128
Reply from 208.67.220.220: bytes=32 time=70ms TTL=128
Reply from 208.67.220.220: bytes=32 time=1ms TTL=128
Reply from 208.67.220.220: bytes=32 time=66ms TTL=128

Ping statistics for 208.67.220.220:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 70ms, Average = 34ms

C:\>
```

Ahora bien, si intentamos realizar este ping desde el dispositivo PC2, el cual está configurado con una IP estática, el resultado varía y obtenemos el siguiente.

```
Packet Tracer PC Command Line 1.0
C:\>ping Cisco.com
Ping request could not find host Cisco.com. Please check the name and try again.
C:\>
```

PRÁCTICA 5:

MONTAJE DE UNA LAN

Grupo completo que hemos realizado esta práctica: Jorge Pérez Álvarez, Alberto Flores Pastor, Alexia Vidal Pons, Ismael Prieto Sánchez y Nerea Agud Lombarte.

1. Poned una fotografía del montaje realizado.

Se cogieron un router, un switcher, un adaptador cable ethernet y los diferentes cables para unir todos los componentes. Primero se unió el router al switcher, el adaptador en el ordenador que funcionará de router y con los cables se unieron todos los componentes de la LAN:

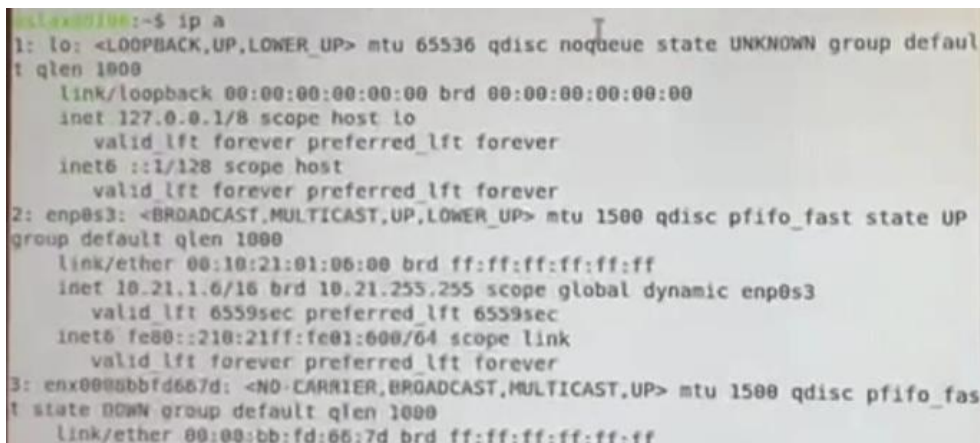
- **Router – LINUX:** Cable negro
- **Servidor Web - LINUX:** Cable blanco
- **Cliente - WINDOWS:** Cable azul
- **Del switch al SSID:** Cable gris
- Cable negro que sale del router es el que se conecta a la corriente eléctrica.



2. Especificad cuales de los retos habéis alcanzado.

- Para cada uno, describe brevemente (usando 50 entre 50 y 100 palabras) qué acciones habéis llevado a cabo.
- **Configurar el punto de acceso:** Debe dar direcciones a los nodos que se conecten con DHCP.
 - Primero reiniciamos el router para borrar las configuraciones anteriores.
 - Con el teléfono móvil nos conectamos a la red wifi (linksys) y accedemos al navegador donde ponemos la IP en la barra del navegador (192.168.1.1).
 - Ponemos el usuario (d6lyinkssys) y la contraseña (admin).
 - Una vez en la configuración del router cambiamos el nombre de la red por Grupo0 y el rango de direcciones IP (192.168.1.100 - 192.168.1.124).
 - Guardamos los cambios y procedemos con el resto de la configuración.

- **Configurar el Linux que hace de router.** Debe configurar las IP de cada interfaz y establecer las reglas de direccionamiento necesarias.
 - El ordenador que hace de router será el único dispositivo que, además, de estar conectado a la LAN que se ha montado, tendrá una toma de internet.
 - Desde la consola de comando, por medio de la instrucción *ip -a* podemos observar los dispositivos conectados con sus correspondientes IPs.



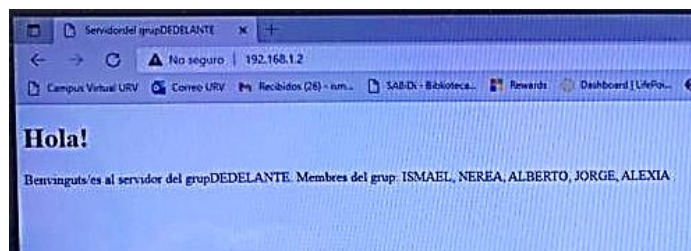
```
root@ubuntu:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:10:21:01:00:00 brd ff:ff:ff:ff:ff:ff
    inet 10.21.1.6/16 brd 10.21.255.255 scope global dynamic enp0s3
        valid_lft 6559sec preferred_lft 6559sec
    inet6 fe80::210:21ff:fe01:000/64 scope link
        valid_lft forever preferred_lft forever
3: enx0000bbfd667d: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000
    link/ether 00:00:bb:fd:66:7d brd ff:ff:ff:ff:ff:ff
```

- En la conexión 3 que se observa en la captura, debemos modificar la que se encuentra por defecto por la que correspondería a la IP de este dispositivo conectado a la red. Nosotros la establecimos como 192.168.1.126 con máscara 24 ya que, se encuentra fuera del rango de direcciones IP establecidos en la configuración del punto de acceso.
 - Finalmente, tendremos que establecer el bit del *ip_forward* a 1 y, de esta manera, poder hacer el ping desde cualquier dispositivo de la red a este de forma correcta.
- **Configurar el cliente Windows.** Debe configurar la IP y tener en cuenta que la salida a Internet será a través de la IPint de Linux que hace de router.
 - Encendemos el cliente Windows
 - Una vez realizado el montaje le hemos cambiado la red y no tiene acceso a internet por lo que reiniciamos el cliente
 - Una vez está montado el servidor Web y el router les hacemos ping y comprobamos el acceso a estos desde el cliente, el cual es correcto.

- **Activar un servidor web Apache en la máquina Linux que hace de servidor web.** Sustituir la web por defecto por el código HTML que le proporcionamos.
 - Actualizar: `sudo apt update`
 - Instalar Apache 2. `sudo apt install apache2`
 - Instalar `sudo apt install ufw`
 - `sudo ufw app list`: Comprobamos que no aparece Apache como disponible así que lo creamos manualmente → `gedit /etc/ufw/applications.d/apache2-utils.ufw.profile` añadimos texto
 - Damos permisos a Apache: `sudo ufw allow "Apache"`
 - Con la instrucción: `sudo ufw status` nos aparece como no disponible, por lo que lo activamos → `sudo ufw enable`
 - Conseguimos activar el servidor: `sudo iptables -L`
 - Modificamos el código html para tener nuestra página: `sudo gedit /var/www/html`
 - `Ip addr add 192.168.1.2/24 dev enp0s3` para poder tener ip en la red.
 - Los otros ordenadores pueden hacer ping y acceder a la página.
 - Cambiamos la ruta IP por defecto para contactar con el router: `Ip route add default via 192.168.1.126`

```
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2021-11-23 13:25:23 CET; 10min ago
     Docs: https://httpd.apache.org/docs/2.4/
  Main PID: 2532 (apache2)
    Tasks: 55 (limit: 4915)
   Memory: 18.0M
    CGroup: /system.slice/apache2.service
            └─2532 /usr/sbin/apache2 -k start
              └─2534 /usr/sbin/apache2 -k start
                └─2535 /usr/sbin/apache2 -k start
```

```
de nov. 23 13:25:22 d108 systemd[1]: Starting The Apache HTTP Server...
de nov. 23 13:25:23 d108 systemd[1]: Started The Apache HTTP Server.
```



- **Manipular los dispositivos**, comprobar el funcionamiento de los cables, cuidar dejarlo aseado, etc.
 - Todo fue montado correctamente (foto anterior) y todos los cables y material funcionaba. Los recogimos de donde los cogimos.

- b. **Poner las reglas o instrucciones que habéis utilizado para configurar el ordenador que hace de router.**
 - i. *ip a*: permite ver todos los dispositivos conectados a la red
 - ii. *ip address add 192.168.1.126/24 dev enx0008bbfd667d*: nos añadirá la nueva ip del dispositivo conectado a la red.
 - iii. *ip address del 192.168.1.106/24 dev enx0008bbfd667d*: elimina la que estaba establecida por defecto.
 - iv. *apt-get remove network-manager*: borrar el network-manager
 - v. *echo > 1 /proc/sys/net/ipv4/ip_forward*: pasar el bit del forward a 1 y permitir hacer ping desde otros dispositivos conectados a la red a nuestro dispositivo.
 - vi. *cat /proc/net/ipv4/ip_forward*: observar si realmente la instrucción anterior había realizado el cambio.
 - vii. *ping 192.168.1.126*: desde cualquiera de los dispositivos conectados a la red para comprobar que, efectivamente se ha podido realizar el ping.

3. **Poner una fotografía conforme desde el cliente y desde el dispositivo sin cables podéis acceder a la web del servidor.**

No conseguimos llegar hasta el punto final. Nos faltó que el router fuese capaz de redireccionar las peticiones que se hacían a él mismo, pero eran para otras IP de la red.

PRÁCTICA 6:

PROTOCOLOS WIRESHARK

En esta práctica usaremos el analizador de red WireShark para estudiar alguno de los protocolos que hemos estado estudiando en teoría. La función de estos analizadores es capturar y analizar los paquetes de nuestro dominio de colisión dentro de nuestra red local. Este tipo de herramientas se utiliza ampliamente para depurar protocolos, comprender el funcionamiento de estos o monitorizar una red para detectar errores, entre otras funciones. Actualmente con el modelo switch lo que nos permite es capturar el tráfico entrante o saliente de la estación que ejecuta el analizador o simplemente el tráfico de difusión conocido como broadcast. Todas nuestras capturas se realizaron sobre la máquina enp0s3.

Reto 1 análisis de un ping:

Nosotros en el laboratorio estableceremos el ping a la máquina 10.21.1.7 de la red interna de la clase por medio del comando 'ping' en la consola.

```
milax@d106:~$ ping 10.21.1.7
PING 10.21.1.7 (10.21.1.7) 56(84) bytes of data.
64 bytes from 10.21.1.7: icmp_seq=1 ttl=64 time=0.714 ms
64 bytes from 10.21.1.7: icmp_seq=2 ttl=64 time=0.211 ms
64 bytes from 10.21.1.7: icmp_seq=3 ttl=64 time=0.323 ms
64 bytes from 10.21.1.7: icmp_seq=4 ttl=64 time=0.290 ms
64 bytes from 10.21.1.7: icmp_seq=5 ttl=64 time=0.317 ms
64 bytes from 10.21.1.7: icmp_seq=6 ttl=64 time=0.415 ms
64 bytes from 10.21.1.7: icmp_seq=7 ttl=64 time=0.359 ms
64 bytes from 10.21.1.7: icmp_seq=8 ttl=64 time=0.286 ms
```

Tras observar el ping recibido, mediante el analizador WireShark, aplicaremos el filtro ICMP, ya que este protocolo es el empleado en las redes TCP/IP para intercambiar datos de estado y con el fin de comprobar si la conexión entre dispositivos es correcta enviando el paquete a la dirección IP del dispositivo.

Mediante este filtro, podemos observar los paquetes ICMP de nuestra red local enviados y recibidos entre el ordenador con el que estábamos trabajando (10.21.1.6) y con el que establecimos la comunicación (10.21.1.7).

Filter: icmp		Expression...	Clear	Apply	Desa
o.		Time	Source	Destination	Protocol Length Info
176	8.830691134	10.21.1.6	10.21.1.7	ICMP	98 Echo (ping) request id=0x4764, seq=1/256, ttl=64 (reply in 177)
177	8.831389205	10.21.1.7	10.21.1.6	ICMP	98 Echo (ping) reply id=0x4764, seq=1/256, ttl=64 (request in 176)
190	9.860361066	10.21.1.6	10.21.1.7	ICMP	98 Echo (ping) request id=0x4764, seq=2/512, ttl=64 (reply in 191)
191	9.860551671	10.21.1.7	10.21.1.6	ICMP	98 Echo (ping) reply id=0x4764, seq=2/512, ttl=64 (request in 190)
204	10.884654398	10.21.1.6	10.21.1.7	ICMP	98 Echo (ping) request id=0x4764, seq=3/768, ttl=64 (reply in 205)
205	10.884954449	10.21.1.7	10.21.1.6	ICMP	98 Echo (ping) reply id=0x4764, seq=3/768, ttl=64 (request in 204)
218	11.908210791	10.21.1.6	10.21.1.7	ICMP	98 Echo (ping) request id=0x4764, seq=4/1024, ttl=64 (reply in 219)
219	11.908472341	10.21.1.7	10.21.1.6	ICMP	98 Echo (ping) reply id=0x4764, seq=4/1024, ttl=64 (request in 218)
258	12.932195278	10.21.1.6	10.21.1.7	ICMP	98 Echo (ping) request id=0x4764, seq=5/1280, ttl=64 (reply in 259)
259	12.932488121	10.21.1.7	10.21.1.6	ICMP	98 Echo (ping) reply id=0x4764, seq=5/1280, ttl=64 (request in 258)
272	13.958711381	10.21.1.6	10.21.1.7	ICMP	98 Echo (ping) request id=0x4764, seq=6/1536, ttl=64 (reply in 273)
273	13.959033091	10.21.1.7	10.21.1.6	ICMP	98 Echo (ping) reply id=0x4764, seq=6/1536, ttl=64 (request in 272)
290	14.982660281	10.21.1.6	10.21.1.7	ICMP	98 Echo (ping) request id=0x4764, seq=7/1792, ttl=64 (reply in 291)
291	14.982336494	10.21.1.7	10.21.1.6	ICMP	98 Echo (ping) reply id=0x4764, seq=7/1792, ttl=64 (request in 290)
303	16.004191874	10.21.1.6	10.21.1.7	ICMP	98 Echo (ping) request id=0x4764, seq=8/2048, ttl=64 (reply in 304)
304	16.004461381	10.21.1.7	10.21.1.6	ICMP	98 Echo (ping) reply id=0x4764, seq=8/2048, ttl=64 (request in 303)

Una vez aplicado el filtro se puede observar los paquetes que ha solicitado nuestro dispositivo y la contestación por parte del otro dispositivo de la red.

Análisis de un ARP:

La memoria ARP (**Address Resolution Protocol**) es necesaria para las comunicaciones de red ya que esta guarda la relación entre las MAC y las direcciones IP de la red. El protocolo ARP entra en funcionamiento en el momento en el que se necesita transmitir un paquete IP destinado a una dirección IP cuyo mapeado MAC se desconoce.

En este reto debíamos eliminar esta memoria y hacer un ping a una de las máquinas de la red del laboratorio. Primero hemos instalado el comando para consultar la memoria caché ARP, la hemos mostrado mediante *sudo arp -a* y finalmente eliminado al completo mediante *sudo ip -s -s neigh flush all*.

```
milax@d106:~$ sudo arp -a
ou1.lab.deim (10.21.0.10) at 00:10:21:00:10:00 [ether] on enp0s3
ou2.lab.deim (10.21.0.18) at 00:10:21:00:18:00 [ether] on enp0s3
? (10.21.0.1) at 00:00:5e:00:01:01 [ether] on enp0s3
d107.lab.deim (10.21.1.7) at 00:10:21:01:07:00 [ether] on enp0s3

milax@d106:~$ sudo ip -s -s neigh flush all
10.21.0.10 dev enp0s3 lladdr 00:10:21:00:10:00 used 107/101/67 probes 4 STALE
10.21.0.18 dev enp0s3 lladdr 00:10:21:00:18:00 used 107/103/69 probes 4 STALE
10.21.0.1 dev enp0s3 lladdr 00:00:5e:00:01:01 ref 1 used 2589/3/2589 probes 4 RE
ACHABLE
10.21.1.7 dev enp0s3 lladdr 00:10:21:01:07:00 used 463/462/423 probes 1 STALE

*** Round 1, deleting 4 entries ***
*** Flush is complete after 1 round ***
```

Una vez hemos borrado la memoria ARP, realizamos un ping a una de las máquinas del laboratorio (10.21.1.7) y observamos el protocolo ARP mediante el analizador Wireshark.

```
milax@d106:~$ sudo arp -a
? (10.21.0.1) at 00:00:5e:00:01:01 [ether] on enp0s3
milax@d106:~$ ping 10.21.1.7
PING 10.21.1.7 (10.21.1.7) 56(84) bytes of data.
54 bytes from 10.21.1.7: icmp_seq=1 ttl=64 time=0.882 ms
54 bytes from 10.21.1.7: icmp_seq=2 ttl=64 time=0.364 ms
54 bytes from 10.21.1.7: icmp_seq=3 ttl=64 time=0.590 ms
54 bytes from 10.21.1.7: icmp_seq=4 ttl=64 time=0.400 ms
54 bytes from 10.21.1.7: icmp_seq=5 ttl=64 time=0.320 ms
54 bytes from 10.21.1.7: icmp_seq=6 ttl=64 time=0.299 ms
54 bytes from 10.21.1.7: icmp_seq=7 ttl=64 time=0.180 ms
54 bytes from 10.21.1.7: icmp_seq=8 ttl=64 time=0.292 ms
^C
--- 10.21.1.7 ping statistics ---
3 packets transmitted, 8 received, 0% packet loss, time 153ms
rtt min/avg/max/mdev = 0.180/0.415/0.882/0.209 ms
```

207	8.54324555	EncantoN_01:06:00	Broadcast	ARP	42 Who has 10.21.1.7? Tell 10.21.1.6
208	8.543684176	EncantoN_01:07:00	EncantoN_01:06:00	ARP	60 10.21.1.7 is at 00:10:21:01:07:00

Como se puede ver en este caso, ha entrado el protocolo ARP ya que, se necesita transmitir un paquete IP destinado a una dirección IP cuyo mapeado MAC se desconoce. Para ello el emisor utiliza la dirección de broadcast de ARP como dirección del destinatario. Va comparando con cada una de las estaciones de su red su dirección IP y en caso de coincidencia transmite un paquete de respuesta de ARP al emisor. Finalmente, esta respuesta la utilizará para crear una entrada en la memoria ARP y así tenerla mapeada para próximas veces.

Análisis de un TCP:

En este reto debemos analizar el three-way handshake. El three-way handshake es un método utilizado en una red TCP / IP para crear una conexión entre un host / cliente local y un servidor. Cuando se establece una conexión TCP, el cliente y el servidor deben enviar un total de 3 paquetes para confirmar el establecimiento de la conexión. En un primer momento el cliente pone el flag SYN en 1, genera aleatoriamente un valor seq = J y envía el paquete de datos al servidor. Ahora el cliente espera que el servidor confirme. Con esto podemos saber que la capacidad de envío del cliente y la capacidad de recepción del servidor funcionan correctamente. En el segundo apretón, al tener el flag SYN=1, el servidor sabe que el cliente quiere establecer la conexión. Para ello, pone el flag ACK=1 y una variable ack en J+1. También, vuelve a generar otro valor seq=K y vuelve a enviar el paquete de datos. De esta forma confirmamos la solicitud de conexión y el cliente puede concluir que las capacidades de recepción y envío del servidor y las capacidades de recepción y envío del cliente son correctas. En el último apretón, el cliente verifica si ack es J+1 y si el flag ACK es 1. Si es correcto, configura ack en K+1 y vuelve a enviar el paquete al servidor, en caso de el servidor confirmar el ack correcto, se establece la conexión entre el cliente y el servidor. De esta forma se puede concluir que las capacidades de envío y recepción del cliente son normales, y las capacidades de envío y recepción del propio servidor también son normales.

Para inicializar el análisis de un TCP primero realizamos la búsqueda en incógnito al dominio "example.com".

Filter:	http	Expression...	Clear	Apply	Desa
No.	Time	Source	Destination	Protocol	Length Info
95	8.541504979	10.21.1.6	93.184.216.34	HTTP	405 GET / HTTP/1.1
97	8.687683502	93.184.216.34	10.21.1.6	HTTP	1093 HTTP/1.1 200 OK (text/html)
99	8.700507315	10.21.1.6	93.184.216.34	HTTP	326 GET /favicon.ico HTTP/1.1
102	8.806800650	93.184.216.34	10.21.1.6	HTTP	1079 HTTP/1.1 404 Not Found (text/html)

Ahora analizaremos el trace de TCP.

No.	Time	Source	Destination	Protocol	Length	Info
90	8.435198904	10.21.1.6	93.184.216.34	TCP	74	57830 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2301282390 TSecr=0 WS=128
93	8.541266240	93.184.216.34	10.21.1.6	TCP	74	80 → 57830 [SYN, ACK] Seq=0 Ack=1 Wln=65535 Len=0 MSS=1460 SACK_PERM=1 TSval=2677176719 TSecr=2301282390 WS=512
94	8.541268141	10.21.1.6	93.184.216.34	TCP	66	57830 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2301282496 TSecr=2677176719
95	8.541504979	10.21.1.6	93.184.216.34	HTTP	409	GET / HTTP/1.1
96	8.645802443	93.184.216.34	10.21.1.6	TCP	66	80 → 57830 [ACK] Seq=1 Ack=349 Win=67072 Len=0 TSval=2677176824 TSecr=2301282496
97	8.687683582	93.184.216.34	10.21.1.6	HTTP	1093	HTTP/1.1 200 OK (text/html)
98	8.687762962	10.21.1.6	93.184.216.34	TCP	66	57830 → 80 [ACK] Seq=340 Ack=1028 Win=64128 Len=0 TSval=2301282642 TSecr=2677176825
99	8.780507315	10.21.1.6	93.184.216.34	HTTP	326	GET /favicon.ico HTTP/1.1
101	8.805483218	93.184.216.34	10.21.1.6	TCP	66	80 → 57830 [ACK] Seq=1028 Ack=600 Win=68096 Len=0 TSval=2677176983 TSecr=2301282655
102	8.806809650	93.184.216.34	10.21.1.6	HTTP	1079	HTTP/1.1 404 Not Found (text/html)
103	8.847505606	10.21.1.6	93.184.216.34	TCP	66	57830 → 80 [ACK] Seq=600 Ack=2841 Win=64128 Len=0 TSval=2301282802 TSecr=2677176984

Como se ve al principio, se encunetran los tres apretones explicados al principio del reto. Se puede observar como los valores de seq y ark van coincidiendo, corroborando así el envío y recepción de paquetes de datos. La siguiente imagen es la correspondiente al primer apretón:

The image shows a Wireshark capture of a network packet. The packet list on the left shows Frame 90: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0. The packet details pane on the right shows the following information:

- Ethernet II, Src: Enterasy, a1:40:77 (08:1f:45:a1:40:77), Dst: Encanto, 01:06:00 (08:10:21:01:06:00)
- Internet Protocol Version 4, Src: 93.184.216.34, Dst: 10.21.1.6
- Transmission Control Protocol, Src Port: 80, Dst Port: 57830, Seq: 0, Ack: 1, Len: 0
- TCP, Seq=0, Win=64240, Len=0, MSS=1460, SACK_PERM=1, TSval=2301282390, TSecr=0, WS=128

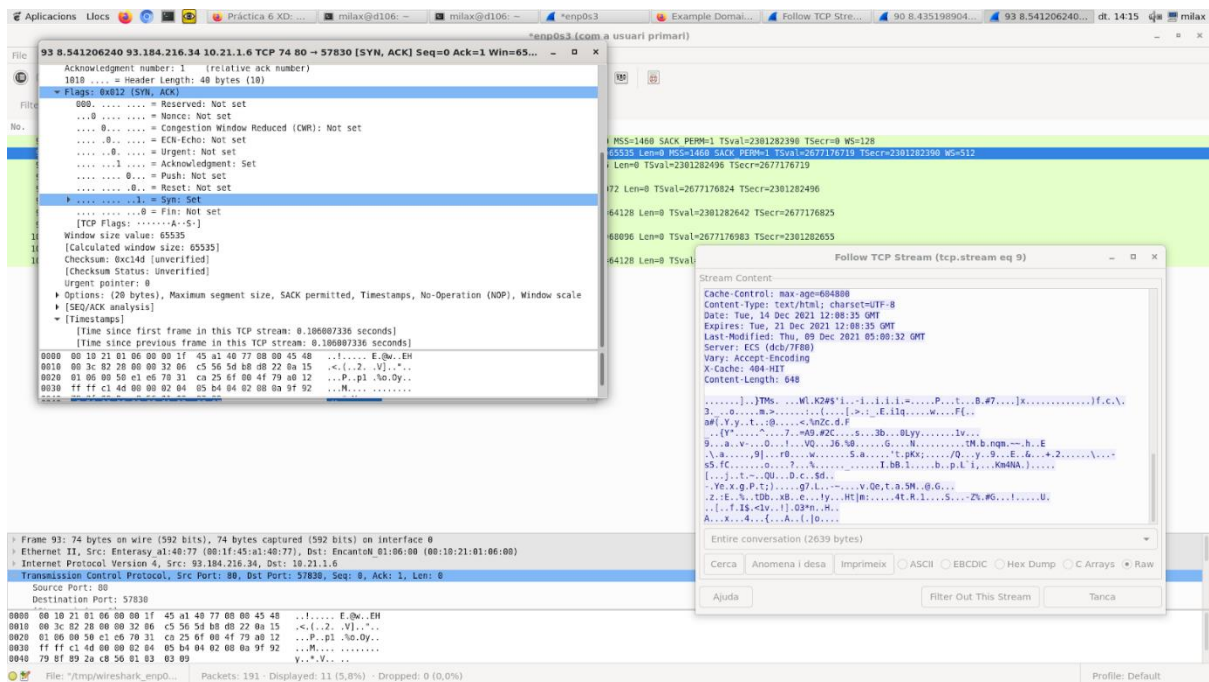
The packet bytes are displayed in hexadecimal and ASCII. The ASCII part shows the following text:

```

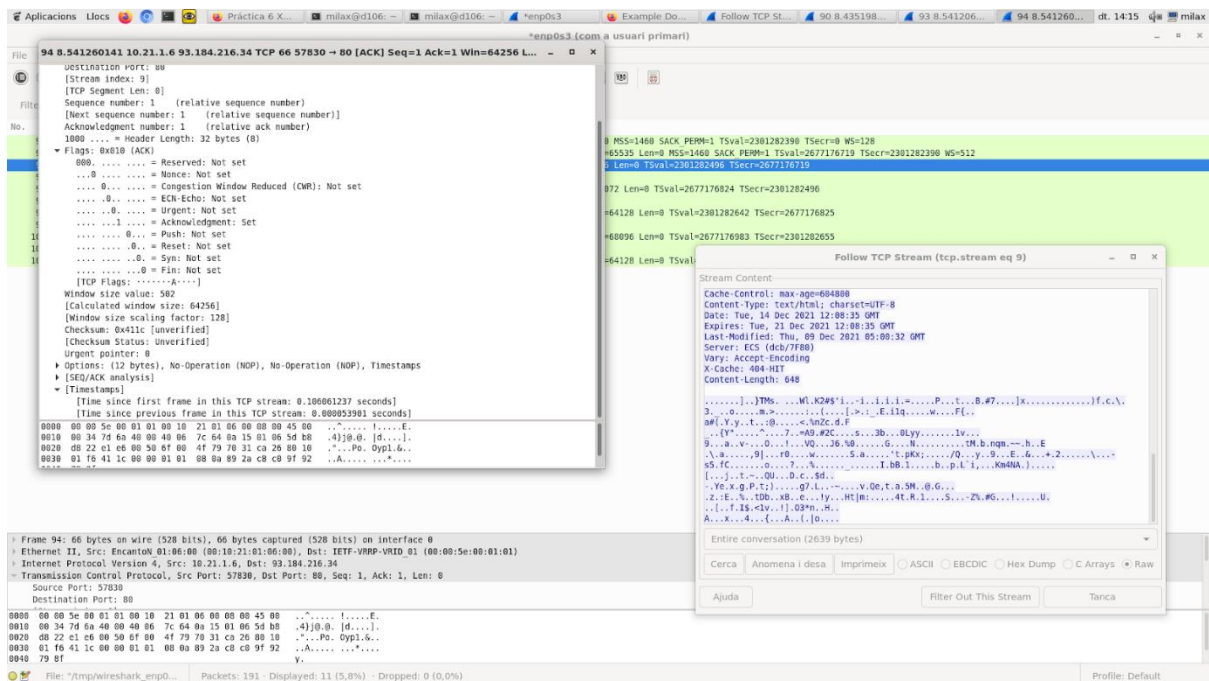
0000  00 10 21 01 06 00 00 1f 45 a1 40 77 00 00 45 00  ..E...EH
0010  00 3c 7d 60 40 00 40 06 7c 5d 0a 15 01 06 5d 00  <10.0.1.1...
0020  00 22 c1 45 00 30 4f 00 4f 70 00 00 00 00 20 00  ..P..No...
0030  ff ff c1 4d 00 02 04 05 b4 04 02 00 0a 9f 92  ..M.....
0040  79 8f 2a c8 56 01 83 03 09  ..Y..V...

```

En este paquete se puede ver el flag SYN activado y que el tiempo que ha tardado es 0. En la siguiente imagen se ve el segundo apretón:



En esta imagen se ve como el flag ACK está activa, ya que así se sabe que el cliente quiere establecer conexión con el servidor. También se puede comprobar que, para establecer esta conexión, el tiempo ya no es 0. En la última imagen se ve el último apretón:



En esta última se corrobora y establece la conexión entre el servidor y el cliente. Ya no necesita el flag SYN, pero si el ACK para comprobar los parámetros. Al igual que en la imagen anterior si consume tiempo el realizar esta conexión.

Análisis de un DNS 1:

La memoria caché DNS es empleada por los navegadores y por el sistema operativo para facilitar la búsqueda a un sitio web frecuente.

El sistema operativo Linux de nuestro dispositivo no tenía información almacenada en esta caché ya que es gestionada directamente desde la caché perteneciente al navegador, por tanto, nosotros para comprobar que realmente se encontraba vacía, activamos el systemd y comprobamos el espacio empleado por este tipo de memoria.

```
milax@dl06:~$ sudo systemd-resolve --statistics
DNSSEC supported by current servers: yes

Transactions
Current Transactions: 0
Total Transactions: 2

Cache
Current Cache Size: 0
Cache Hits: 0
Cache Misses: 0

DNSSEC Verdicts
Secure: 0
Insecure: 0
Bogus: 0
Indeterminate: 0
```

Como se puede ver en la imagen en current cache size sale 0. De esta forma, comprobamos que esta memoria cache está vacía. En la siguiente imagen se observará el resultado de los paquetes UDP al buscar en un dominio conocido por el servidor (www.urv.cat).

The image shows a Wireshark packet capture of a successful DNS query. The filter is set to 'udp'. The packet list shows a standard query request from 10.0.0.100 to 10.0.0.1. The packet details pane shows the query for 'milax.d106.com' with a response from 10.0.0.1. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.100	10.0.0.1	DNS	58	Standard query 0x1000000000000000 milax.d106.com
2	0.00176201	10.0.0.1	10.0.0.100	DNS	80	Standard query response 0x1000000000000000 milax.d106.com

Frame 1: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
Ethernet II, Src: Encant0n 01:00:00:00:00:00, Dst: IETF-VRRP-VRID_01 08:00:5e:00:01:01
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.100
User Datagram Protocol, Src Port: 40114, Dst Port: 53
Domain Name System (query)

En esta imagen se ve como se establece una conexión con ese dominio, ya que es una de las direcciones que tiene guardada la cache DNS del servidor. En la siguiente imagen observaremos el conjunto de paquetes UDP correspondientes a un dominio que no existe:

The image shows a Wireshark packet capture of multiple failed DNS queries. The filter is set to 'udp'. The packet list shows a series of standard query requests from 10.0.0.100 to 10.0.0.1 for a non-existent domain. The packet details pane shows the query for 'milax.d106.com' with a response from 10.0.0.1. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
7441	12.18008289	10.0.0.1	10.0.0.1	DNS	153	Standard query response 0xbf40 no such name AAAA fwgwege.com lab.dein SOA dns.urf.cat
7442	12.17299554	10.0.0.1	10.0.0.1	DNS	75	Standard query 0xc3a1 A www.fwgwege.com
7443	12.22071796	10.0.0.1	10.0.0.1	DNS	148	Standard query response 0xc3a1 no such name A www.fwgwege.com SOA a.gtld-servers.net

Frame 1: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
Ethernet II, Src: Encant0n 01:00:00:00:00:00, Dst: IETF-VRRP-VRID_01 08:00:5e:00:01:01
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.100
User Datagram Protocol, Src Port: 40114, Dst Port: 53
Domain Name System (query)

Como se puede ver sale que en la memoria cache DNS del servidor, no se reconoce ningún dominio con ese nombre (No such name).

Análisis de un DNS 2:

En este último reto intentaremos forzar la fragmentación de un datagrama. Para ello pondremos la medida del ping en 1800 bytes, siendo así superior a la MTU de la capa de enlace (1500 bytes). La MTU nos indica el tamaño en bytes de la unidad de datos más grande que puede enviarse usando un protocolo de comunicaciones. Si establecemos un valor de MTU demasiado alto causaremos fragmentación y pérdidas en los paquetes enviados. También, si es demasiado bajo no llegamos a aprovechar de forma adecuada la capacidad de la red.

En la siguiente imagen vemos que sucede al realizar el comando **ping 192.168.1.100 -s 1800**:

No.	Time	Source	Destination	Protocol	Length	Info
286	19.46826843	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=468c) [Reassembled in #287]
287	19.46836595	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=1/256, ttl=64 (no response found)
315	20.48205104	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=46a9) [Reassembled in #316]
316	20.48209659	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=2/512, ttl=64 (no response found)
347	21.50462914	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4744) [Reassembled in #348]
348	21.50464845	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=3/768, ttl=64 (no response found)
373	22.52850265	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=47ad) [Reassembled in #374]
374	22.52856451	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=4/1024, ttl=64 (no response found)
400	23.56803232	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4833) [Reassembled in #401]
401	23.56804257	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=5/1280, ttl=64 (no response found)
434	24.57860368	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=48ab) [Reassembled in #435]
435	24.57861393	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=6/1536, ttl=64 (no response found)
451	25.60157360	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4903) [Reassembled in #452]
452	25.60159161	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=7/1792, ttl=64 (no response found)
466	26.62639940	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=49b1) [Reassembled in #467]
467	26.62631912	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=8/2048, ttl=64 (no response found)
484	27.64802386	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=49ff) [Reassembled in #485]
485	27.64803972	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=9/2304, ttl=64 (no response found)
546	28.67580335	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4a56) [Reassembled in #547]
547	28.67582578	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=10/2560, ttl=64 (no response found)
562	29.70239913	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4a80) [Reassembled in #563]
563	29.70239882	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=11/2816, ttl=64 (no response found)
581	30.72279729	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4acb) [Reassembled in #582]
582	30.72281371	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=12/3072, ttl=64 (no response found)
599	31.74796728	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4b88) [Reassembled in #600]
600	31.74799505	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=13/3328, ttl=64 (no response found)
612	32.77078298	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4c72) [Reassembled in #613]
613	32.77080475	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=14/3584, ttl=64 (no response found)
623	33.79265408	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4c82) [Reassembled in #624]
624	33.79267241	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=15/3840, ttl=64 (no response found)
635	34.81005315	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4d1d) [Reassembled in #636]
636	34.81007184	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=16/4096, ttl=64 (no response found)
658	35.84082820	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4d86) [Reassembled in #659]
659	35.84083597	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=17/4352, ttl=64 (no response found)
678	36.86415489	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4e45) [Reassembled in #679]
679	36.86416923	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=18/4608, ttl=64 (no response found)
691	37.89397813	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4e87) [Reassembled in #692]
692	37.89398835	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=19/4864, ttl=64 (no response found)
729	39.91598886	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4eae) [Reassembled in #730]
730	39.91600458	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=20/5120, ttl=64 (no response found)
740	39.93658751	192.168.1.1	192.168.1.100	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=4f62) [Reassembled in #741]
741	39.93654271	192.168.1.100	192.168.1.1	ICMP	362	Echo (ping) request id=0x0ae0, seq=21/5376, ttl=64 (no response found)

Para llegar a este punto hemos filtrado los resultados según la dirección destino mediante el filtro `ip.addr==192.168.1.100`. Como se puede ir observando el paquete se ha ido fragmentando y también para comprobar la correcta fragmentación, se observa que en todos los paquetes fragmentados tenemos la misma identificación: `0x0ae0`. Podemos comprobar también que estos paquetes tienen la misma longitud de 362 bytes.

REFLEXIONES

Reflexión conjunta

Las prácticas de la asignatura las consideramos bastante apropiadas. Nos gusta sobre todo el hecho de que va profundizando cada vez más. Es decir, las primeras son más introductorias y las finales profundiza más en conceptos concretos de redes de datos. Las prácticas más interesantes que nos han parecido han sido la 3 y la 5. De la 3 nos ha gustado la funcionalidad que tiene y que hemos sido nosotros los que hemos decidido que código queríamos implementar. Y de la 5 nos gustó el trabajar juntamente con otro grupo, pudiendo poner ideas en común. Aun así, creemos que algunas prácticas se podrían desarrollar un poco más en el guion, ya que a veces íbamos un poco perdidos (como en la 5). No obstante, el profesor en el laboratorio iba dando claras instrucciones para poder avanzar en caso de que tuviésemos dudas o no supiésemos continuar.

Reflexión Alberto Flores

Al ser un grupo de dos personas el trabajo se repartía bastante fácil, ya que, realizar la división de forma equitativa entre dos es relativamente sencillo. Primero hemos realizado las prácticas en el laboratorio, escribiendo un esbozo en el informe. Un par de semanas antes de la entrega, nos pusimos los dos a ir comentando y desarrollando el informe. En clase, nos poníamos los dos en un mismo ordenador y mientras uno controlaba el ordenador, el otro iba dando indicaciones y escribiendo en el documento lo que íbamos haciendo. Este rol nos lo íbamos intercambiando en cada sesión. En general, creo que he contribuido de forma equitativa tanto en la realización de las prácticas como en la escritura de este informe. Gracias a estos laboratorios he conseguido profundizar en concepto de redes que simplemente me sonaban, ya que en informática no se suele tratar estos conceptos.

Reflexión Jorge Pérez

A la hora de realizar las prácticas hemos tenido que repartir el trabajo de una forma bastante equitativa para que ninguno tuviera que hacer demasiado por su parte al ser solo 2 personas. Aun así, lo hemos repartido de tal manera de que ambos hayamos trabajado en todas las prácticas, tanto en las horas de clases como en el tiempo que le hemos tenido que dedicar fuera de estas. Considero que, aunque no hayamos trabajado por igual todas las prácticas, ya que dependía de cómo nos hubiéramos repartido el trabajo, la labor de ambos ha sido necesaria para poder realizar cada una de las prácticas.

La asignatura nos ha aportado unos conocimientos fundamentales en muchos campos de la ingeniería como telecomunicaciones y que considero necesarios para nuestro aprendizaje, ya que el funcionamiento de las redes está estrechamente relacionado con la informática.