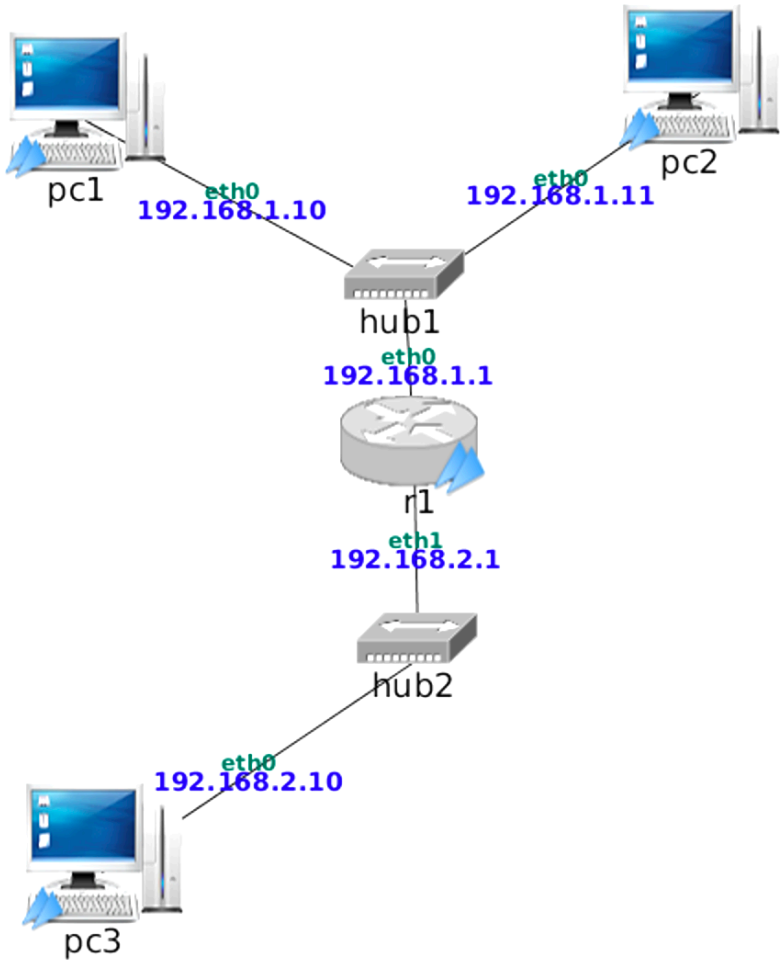
Emulación de redes con NetGui.

Redes de Computadores PL1.

Juan Casado Ballesteros
DNI: 09108762A

GII Mañana 8:00 a 10:00 am

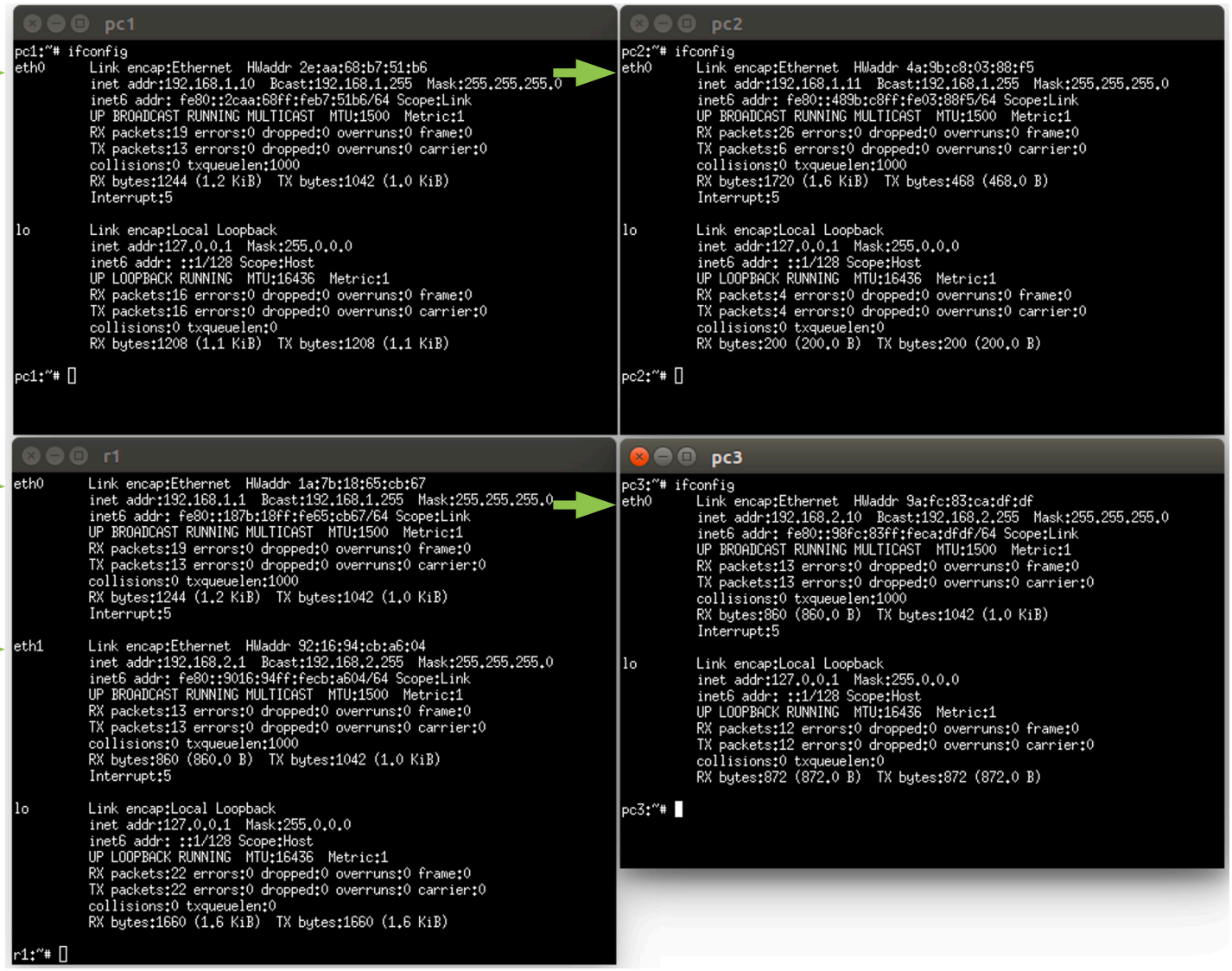


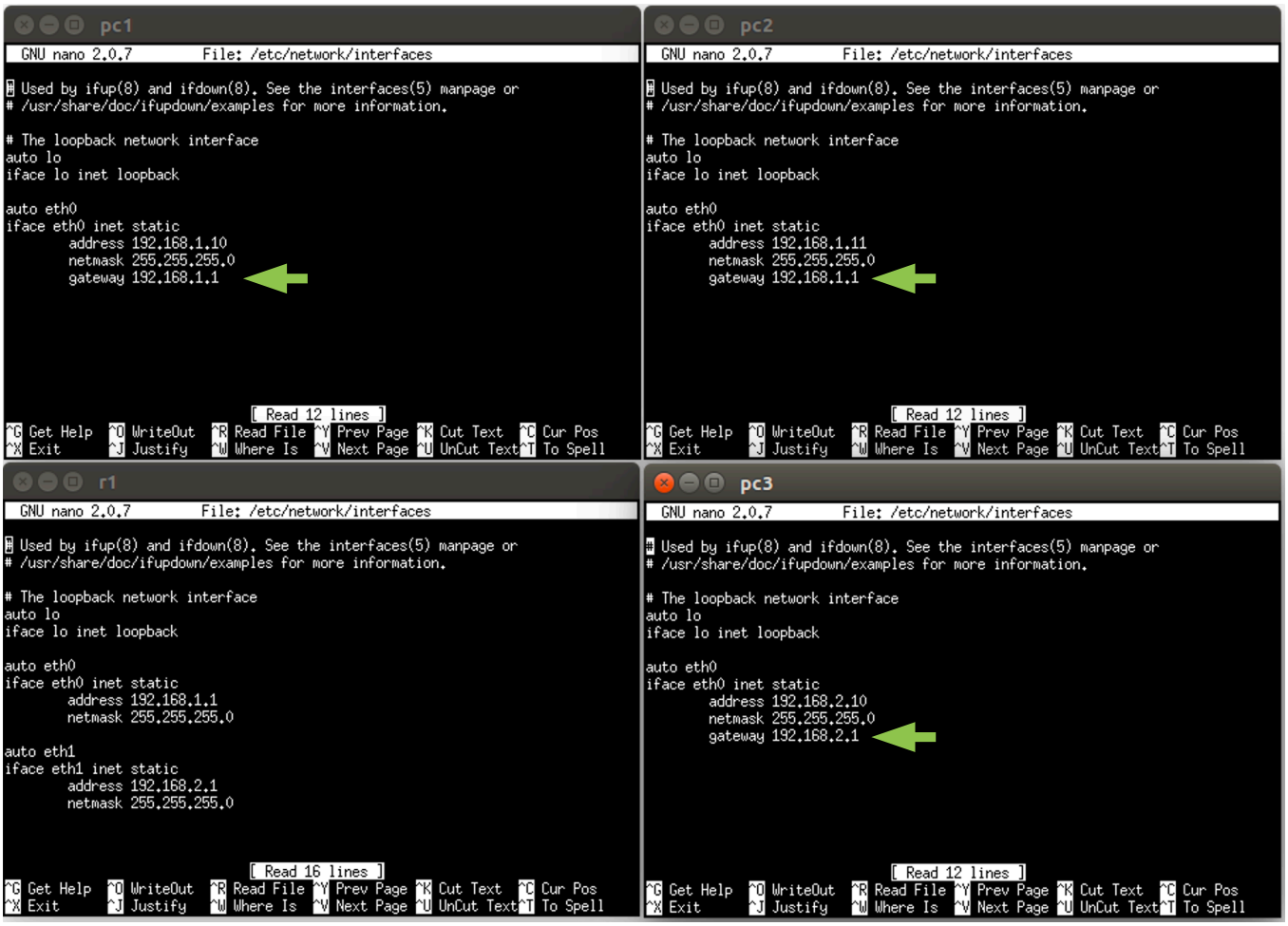
Configuración de la red.

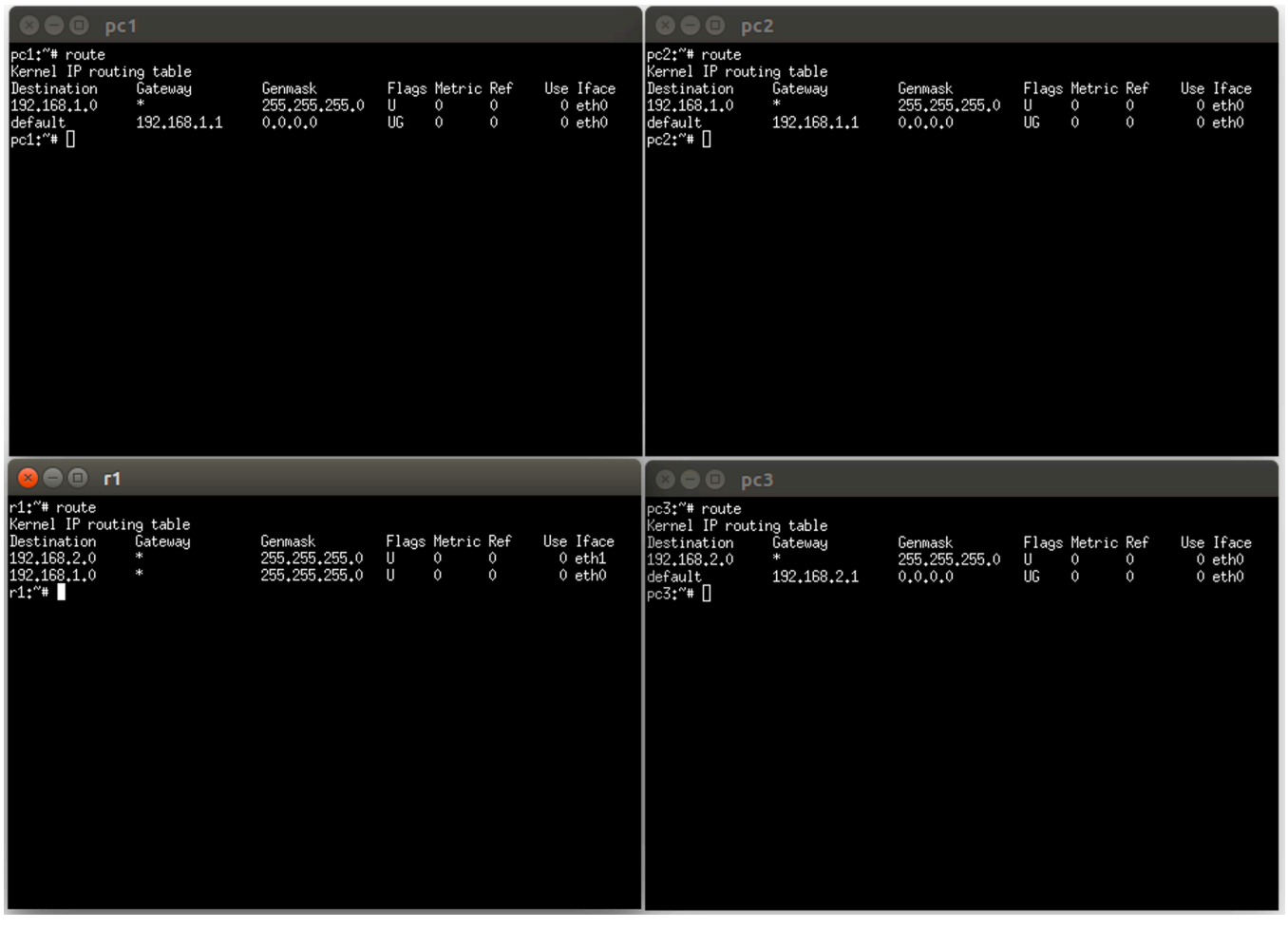
En los apartados 5, 6 y 7 se observa como los equipos que están en la misma subred pueden comunicarse entre si, pero no pueden comunicarse con los equipos de otras subredes. PC1, PC2, y R1 a través de eth0 forman una subred conectada mediante HUB1; PC3 y R1 a través de eth1 forman otra subred conectada mediante HUB2.

Para solucionar esto debemos indicar el router por defecto al que enviar los datagramas cuando el equipo receptor al que queremos comunicarnos no está en ella, creando de este modo las tablas de enrolamiento de los equipos. Una vez configurado el router por defecto para cada equipo de la subred todos los puntos de nuestra red podrán comunicarse entre sí. Se adjuntan las fotos correspondientes a:

- Direcciones IP de las interfaces de red de cada maquina (visualizado con ifconfig).
- Archivo de configuración de las interfaces de red (visualizando con nano /etc/network/interfaces).
- Tabla de enrolamiento de cada equipo (visualizado con route).




pc1
GNU nano 2.0.7 File: /etc/network/interfaces
Used by ifup(8) and ifdown(8). See the interfaces(5) manpage or
/usr/share/doc/ifupdown/examples for more information.
The loopback network interface
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.1.10
netmask 255.255.255.0
gateway 192.168.1.1
[Read 12 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
pc2
GNU nano 2.0.7 File: /etc/network/interfaces
Used by ifup(8) and ifdown(8). See the interfaces(5) manpage or
/usr/share/doc/ifupdown/examples for more information.
The loopback network interface
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.1.11
netmask 255.255.255.0
gateway 192.168.1.1
[Read 12 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
r1
GNU nano 2.0.7 File: /etc/network/interfaces
Used by ifup(8) and ifdown(8). See the interfaces(5) manpage or
/usr/share/doc/ifupdown/examples for more information.
The loopback network interface
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.1.1
netmask 255.255.255.0
auto eth1
iface eth1 inet static
address 192.168.2.1
netmask 255.255.255.0
[Read 16 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
pc3
GNU nano 2.0.7 File: /etc/network/interfaces
Used by ifup(8) and ifdown(8). See the interfaces(5) manpage or
/usr/share/doc/ifupdown/examples for more information.
The loopback network interface
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.2.10
netmask 255.255.255.0
gateway 192.168.2.1
[Read 12 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell


pc1
pc1:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.1.0 * 255.255.255.0 U 0 0 0 eth0
default 192.168.1.1 0.0.0.0 UG 0 0 0 eth0
pc1:~#
pc2
pc2:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.1.0 * 255.255.255.0 U 0 0 0 eth0
default 192.168.1.1 0.0.0.0 UG 0 0 0 eth0
pc2:~#
r1
r1:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.2.0 * 255.255.255.0 U 0 0 0 eth1
192.168.1.0 * 255.255.255.0 U 0 0 0 eth0
r1:~#
pc3
pc3:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.2.0 * 255.255.255.0 U 0 0 0 eth0
default 192.168.2.1 0.0.0.0 UG 0 0 0 eth0
pc3:~#

Análisis del tráfico.

Realizamos un ping de PC3 a PC1 y capturamos los paquetes de la red en R1 (eth1). Vemos que los primeros paquetes capturados se corresponden a paquetes ARP (Address Resolution Protocol), en ellos PC3 pregunta por la MAC correspondiente a la IP del router, el router le contesta proporcionándole dicha información.

Posteriormente se produce el intercambio de reply, request de los paquetes ICMP (Internet Control Message Protocol) que produce la orden ping. Tras unos cuantos paquetes la caché ARP del router se renueva preguntando por la MAC de PC3. Después continuan los paquetes ICMP de forma normal.

Debido a que estamos capturando datos en eth1 del router podemos ver las comunicaciones ARP entre el router y PC3 para actualizar las caché ARP mientras PC3 se comunica con PC1. Si probamos a capturar datos entre PC1 y R1 sobre eth0 también vemos este mismo tipo de comportamiento, no es así entre PC2 y R1 ya que PC2 no está recibiendo paquetes, deducimos así que la actualización de la caché ARP se da solo cuando se está produciendo una comunicación.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9a:fc:83:ca:df:df	Broadcast	ARP	42	Who has 192.168.2.1? Tell 192.168.2.10
2	0.000111	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	192.168.2.1 is at 92:16:94:cb:a6:04
3	0.000271	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=1/256, ttl=64 (reply in 4)
4	0.011830	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=1/256, ttl=63 (request in 3)
5	0.996774	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=2/512, ttl=64 (reply in 6)
6	0.997424	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=2/512, ttl=63 (request in 5)
7	1.998141	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=3/768, ttl=64 (reply in 8)
8	1.998523	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=3/768, ttl=63 (request in 7)
9	2.997082	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=4/1024, ttl=64 (reply in 10)
10	2.997391	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=4/1024, ttl=63 (request in 9)
11	3.997740	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=5/1280, ttl=64 (reply in 12)
12	3.998287	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=5/1280, ttl=63 (request in 11)
13	4.999050	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=6/1536, ttl=64 (reply in 14)
14	4.999538	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=6/1536, ttl=63 (request in 13)
15	5.009837	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	Who has 192.168.2.10? Tell 192.168.2.1
16	5.010240	9a:fc:83:ca:df:df	92:16:94:cb:a6:…	ARP	42	192.168.2.10 is at 9a:fc:83:ca:df:df
17	5.998281	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=7/1792, ttl=64 (reply in 18)
18	5.998474	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=7/1792, ttl=63 (request in 17)
19	6.997361	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=8/2048, ttl=64 (reply in 20)
20	6.997623	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=8/2048, ttl=63 (request in 19)
21	7.996388	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=9/2304, ttl=64 (reply in 22)
22	7.996727	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=9/2304, ttl=63 (request in 21)
23	8.995652	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=10/2560, ttl=64 (reply in 24)
24	8.996027	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=10/2560, ttl=63 (request in 23)
25	9.999935	192.168.2.10	192.168.1.10	ICMP	98	Echo (ping) request id=0x7902, seq=11/2816, ttl=64 (reply in 26)
26	10.000354	192.168.1.10	192.168.2.10	ICMP	98	Echo (ping) reply id=0x7902, seq=11/2816, ttl=63 (request in 25)

Ahora procedemos a enviar un ping cuya cantidad de datos no cabe en el datagrama, en concreto enviamos 2000B de datos, por lo que este se ve forzado a fragmentarse. Vemos que para cada request y para cada reply son necesarios dos datagramas (se envían dos dategramas por ICMP) porque en uno solo no caben los datos a enviar.

Se aprecia como los mensajes de llenado y refresco de la caché ARP se siguen produciendo.

El tamaño máximo de los datagramas enviados es de 1514 Bytes, el otro datagrama es de 562 Bytes por lo que hay una sobrecarga total de 78 Bytes debido a las cabeceras necesarias para transmitir los datos fragmentados.

En el primer datagrams se envían 1480 Bytes de datos y 34 Bytes de cabecera, en el segundo 528 Bytes de datos y 34 Bytes de cabecera. Ambos datos se obtienen de mirar en la cabecera IP el campo total length y restarle el tamaño de la cabecera IP que son 20 Bytes.

>ping

He utilizado este comando para comprobar si los terminales eran accesibles a lo largo de la fase de configuración de la red

>arp

He utilizado este comando para ver el estado de la caché ARP y borrarla con el modificador -d

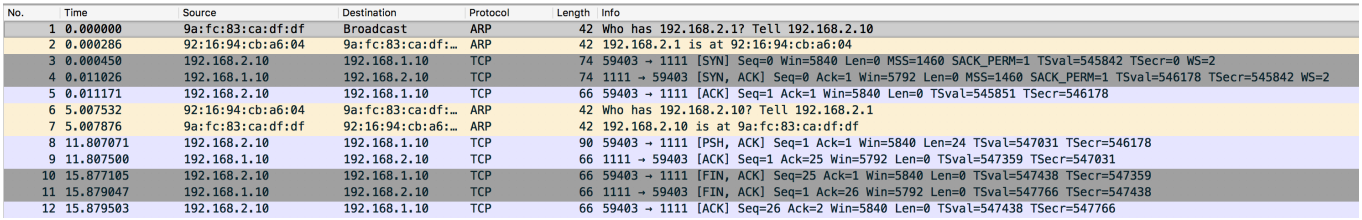
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9a:fc:83:ca:df:df	Broadcast	ARP	42	Who has 192.168.2.1? Tell 192.168.2.10
2	0.000207	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	192.168.2.1 is at 92:16:94:cb:a6:04
3	0.000376	192.168.2.10	192.168.1.10	ICMP	1514	Echo (ping) request id=0xec02, seq=1/256, ttl=64 (reply in 5)
4	0.000547	192.168.2.10	192.168.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=0650)
5	0.012166	192.168.1.10	192.168.2.10	ICMP	1514	Echo (ping) reply id=0xec02, seq=1/256, ttl=63 (request in 3)
6	0.012208	192.168.1.10	192.168.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6131)
7	0.994698	192.168.2.10	192.168.1.10	ICMP	1514	Echo (ping) request id=0xec02, seq=2/512, ttl=64 (reply in 9)
8	0.994702	192.168.2.10	192.168.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=0651)
9	0.995337	192.168.1.10	192.168.2.10	ICMP	1514	Echo (ping) reply id=0xec02, seq=2/512, ttl=63 (request in 7)
10	0.995377	192.168.1.10	192.168.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6132)
11	1.996930	192.168.2.10	192.168.1.10	ICMP	1514	Echo (ping) request id=0xec02, seq=3/768, ttl=64 (reply in 13)
12	1.996935	192.168.2.10	192.168.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=0652)
13	1.997577	192.168.1.10	192.168.2.10	ICMP	1514	Echo (ping) reply id=0xec02, seq=3/768, ttl=63 (request in 11)
14	1.997618	192.168.1.10	192.168.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6133)
15	2.995895	192.168.2.10	192.168.1.10	ICMP	1514	Echo (ping) request id=0xec02, seq=4/1024, ttl=64 (reply in 17)
16	2.995900	192.168.2.10	192.168.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=0653)
17	2.996292	192.168.1.10	192.168.2.10	ICMP	1514	Echo (ping) reply id=0xec02, seq=4/1024, ttl=63 (request in 15)
18	2.996316	192.168.1.10	192.168.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6134)
19	3.994945	192.168.2.10	192.168.1.10	ICMP	1514	Echo (ping) request id=0xec02, seq=5/1280, ttl=64 (reply in 21)
20	3.994951	192.168.2.10	192.168.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=0654)
21	3.995445	192.168.1.10	192.168.2.10	ICMP	1514	Echo (ping) reply id=0xec02, seq=5/1280, ttl=63 (request in 19)
22	3.995475	192.168.1.10	192.168.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6135)
23	4.998583	192.168.2.10	192.168.1.10	ICMP	1514	Echo (ping) request id=0xec02, seq=6/1536, ttl=64 (reply in 25)
24	4.998587	192.168.2.10	192.168.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=0655)
25	4.999081	192.168.1.10	192.168.2.10	ICMP	1514	Echo (ping) reply id=0xec02, seq=6/1536, ttl=63 (request in 23)
26	4.999151	192.168.1.10	192.168.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6136)
27	5.009642	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	Who has 192.168.2.10? Tell 192.168.2.1
28	5.009979	9a:fc:83:ca:df:df	92:16:94:cb:a6:…	ARP	42	192.168.2.10 is at 9a:fc:83:ca:df:df
29	5.998132	192.168.2.10	192.168.1.10	ICMP	1514	Echo (ping) request id=0xec02, seq=7/1792, ttl=64 (reply in 31)
30	5.998138	192.168.2.10	192.168.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=0656)
31	5.998920	192.168.1.10	192.168.2.10	ICMP	1514	Echo (ping) reply id=0xec02, seq=7/1792, ttl=63 (request in 29)
32	5.998960	192.168.1.10	192.168.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6137)
33	6.998552	192.168.2.10	192.168.1.10	ICMP	1514	Echo (ping) request id=0xec02, seq=8/2048, ttl=64 (reply in 35)
34	6.998557	192.168.2.10	192.168.1.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=0657)
35	6.999229	192.168.1.10	192.168.2.10	ICMP	1514	Echo (ping) reply id=0xec02, seq=8/2048, ttl=63 (request in 33)
36	6.999262	192.168.1.10	192.168.2.10	IPv4	562	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6138)

Creamos un servidor TCP y un cliente, enviamos un mensaje del cliente al servidor y cerraros la conexión obteniendo la captura adjunta.

Podemos apreciar los mensajes de llenado de la caché ARP en los que PC1 pregunta sobre la MAC del router y donde el router pregunta sobre la MAC de PC1.

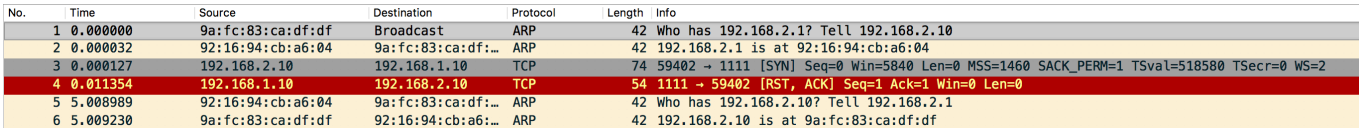
En lo referente a los paquetes TCP vemos los tres paquetes de inicio de conexión, SYN, SYN-ACK y ACK, posteriormente se envía un mensaje de PC3 a PC1 a lo que obtenemos el correspondiente ACK. Finalmente se envían otros tres mensajes de cierre de conexión, FIN-ACK, FIN-ACK y ACK.

TCP es un protocolo orientado a conexión, es por esto que existan unos mensajes de negociación entre cliente y servidor para establecerla. Otra cosa a desatacar es que al cerrar la conexión el en cliente esta se cierra también en el servidor de forma automática.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9a:fc:83:ca:df:df	Broadcast	ARP	42	Who has 192.168.2.1? Tell 192.168.2.10
2	0.000286	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	192.168.2.1 is at 92:16:94:cb:a6:04
3	0.000450	192.168.2.10	192.168.1.10	TCP	74	59403 → 1111 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=545842 TSecr=0 WS=2
4	0.011026	192.168.1.10	192.168.2.10	TCP	74	1111 → 59403 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=546178 TSecr=545842 WS=2
5	0.011171	192.168.2.10	192.168.1.10	TCP	66	59403 → 1111 [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSval=545851 TSecr=546178
6	5.007532	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	Who has 192.168.2.10? Tell 192.168.2.1
7	5.007876	9a:fc:83:ca:df:df	92:16:94:cb:a6:…	ARP	42	192.168.2.10 is at 9a:fc:83:ca:df:df
8	11.807071	192.168.2.10	192.168.1.10	TCP	90	59403 → 1111 [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=24 TSval=547031 TSecr=546178
9	11.807500	192.168.1.10	192.168.2.10	TCP	66	1111 → 59403 [ACK] Seq=1 Ack=25 Win=5792 Len=0 TSval=547359 TSecr=547031
10	15.877105	192.168.2.10	192.168.1.10	TCP	66	59403 → 1111 [FIN, ACK] Seq=25 Ack=1 Win=5840 Len=0 TSval=547438 TSecr=547359
11	15.879047	192.168.1.10	192.168.2.10	TCP	66	1111 → 59403 [FIN, ACK] Seq=1 Ack=26 Win=5792 Len=0 TSval=547766 TSecr=547438
12	15.879503	192.168.2.10	192.168.1.10	TCP	66	59403 → 1111 [ACK] Seq=26 Ack=2 Win=5840 Len=0 TSval=547438 TSecr=547766

En la siguiente captura un cliente TCP intenta conectarse a un puerto en el que no hay un socket TCP abierto, es decir, en el que no hay un cliente escuchando. Lo que podemos observar es que al enviar el paquete con el flag SYN activado recibidos un RST-ACK que nos indica que la conexión no es posible tras lo cual en la terminal obtenemos que no es posible establecer la conexión por lo que el cliente no es creado.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9a:fc:83:ca:df:df	Broadcast	ARP	42	Who has 192.168.2.1? Tell 192.168.2.10
2	0.000032	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	192.168.2.1 is at 92:16:94:cb:a6:04
3	0.000127	192.168.2.10	192.168.1.10	TCP	74	59402 → 1111 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=518580 TSecr=0 WS=2
4	0.011354	192.168.1.10	192.168.2.10	TCP	54	1111 → 59402 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	5.008989	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	Who has 192.168.2.10? Tell 192.168.2.1
6	5.009230	9a:fc:83:ca:df:df	92:16:94:cb:a6:…	ARP	42	192.168.2.10 is at 9a:fc:83:ca:df:df

Creamos un servidor UDP y un cliente y enviamos un mensaje del cliente al servidor obteniendo la captura adjunta.

UDP es un protocolo sin conexión por lo que no son necesarios los mensajes de establecimiento y fin de la misma como si los son en TCP. El mensaje es creado y enviado sin más preámbulos y sin garantía de que el servidor lo vaya a recibir. Esto puede ser una gran ventaja ya que para enviar el mismo mensaje solo hemos necesitado un paquete mientras que con TCP nos hicieron falta un total de 8 lo cual implica mayor lentitud y una sobrecarga de la red.

Con UDP al enviar un mensaje el servidor lo recibe como le llegue si es que le llega y sin dar ninguna información al cliente de cómo ha ido la comunicación.

Ya que no hay ninguna conexión establecida si cerramos el cliente el servidor permanecerá abierto y a la escucha de cualquier otro cliente que se quiera comunicar con él, a diferencia de en TCP este no se cierra automáticamente ya que no está vinculado a ningún cliente concreto.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9a:fc:83:ca:df:df	Broadcast	ARP	42	Who has 192.168.2.1? Tell 192.168.2.10
2	0.000144	92:16:94:cb:a6:04	9a:fc:83:ca:df:…	ARP	42	192.168.2.1 is at 92:16:94:cb:a6:04
3	0.000122	192.168.2.10	192.168.1.10	UDP	66	32768 → 1111 Len=24