UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Tun-Tap

Trabajo Redes Corporativas

**Grado en Ingeniería Informática**

**Autores**: Adrián Carrillo y Vicente Arnau

**Router**: 17

2019/2020

# Resumen

Tun-Tap es una herramienta de creación de interfaces de red virtuales mediante software. Si bien apareció hace casi 20 años, sigue siendo empleada por multitud de soluciones comerciales de VPN. En este caso la vamos a instalar en equipos Linux para conectar de manera virtual dos subredes y ver cómo encapsula el tráfico para transmitirlo vía paquetes TCP/IP.

**Palabras clave:** tun, interfaz, VPN, enrutamiento.

# Abstract

Tun-Tap is a tool for creating virtual network interfaces using software. Although it appeared almost 20 years ago, it is still used by many commercial VPN solutions. In this case we are going to install it on Linux computers to connect two subnets in a virtual way and see how the traffic is encapsulated to transmit it via TCP / IP packets.

**Keywords :** tun, interface, VPN, routing.



# Tabla de contenidos

# 1. Introducción

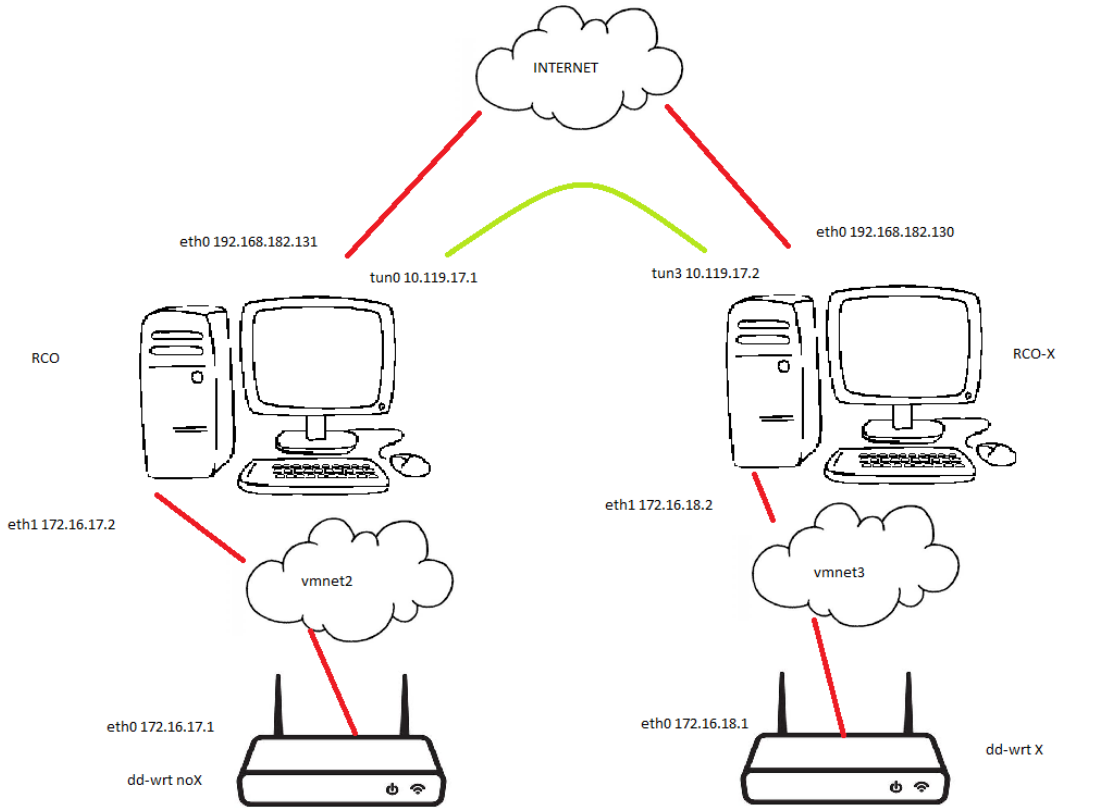
El uso de redes privadas virtuales (VPN[1]) es muy extendido actualmente por todo el globo debido a la facilidad de su uso y a su flexibilidad. El poder acceder a una red completamente externa a la que tenemos en casa se ha convertido en muchos lugares en algo completamente habitual y necesario si se requiere de conexión a una red ajena.

Como vimos anteriormente con los túneles PPTP, fácilmente se pueden conectar dos subredes para que virtualmente se conciba que ambas estén interconectadas. Ahora con Tun-Tap vamos a ver cómo funciona este tipo de conexiones y cómo se encaminan los paquetes de una red a otra. Si bien requiere algo más de configuración que otras soluciones, el hecho de que el código fuente sea público y modificable ya que se encuentra el programa en C en internet simplifica el entender su funcionamiento.

A modo de resumen muy simple, al crear una nueva interfaz virtual redirige el tráfico que antes saldría directamente a internet mediante nuestras conexiones habituales como `eth0` en Linux para mandarlo por sus propias interfaces `tun` y que llegue a la red destino que habremos configurado.

[1]Siglas en inglés de *virtual prívate network*.

La red sobre la que partimos es la que se indica en la siguiente figura:



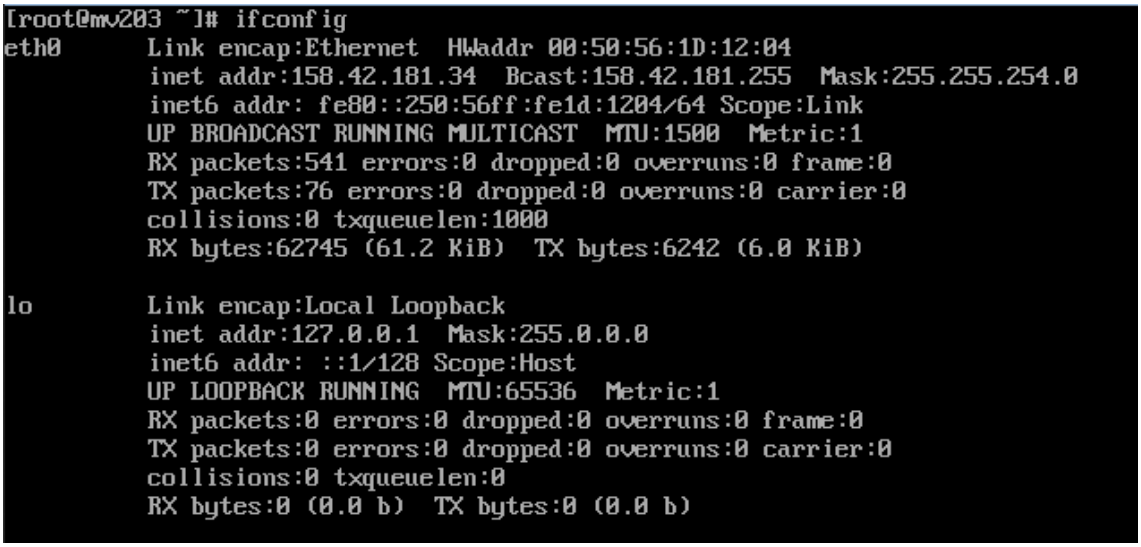

Aclaración sobre la figura: en las primeras capturas, las interfaces `eth0` de ambas máquinas virtuales `RCO` pueden tener distintas direcciones ya que las máquinas virtuales no reciben las mismas direcciones en la UPV que en la red doméstica[2]. Aquí ya hemos gastado las direcciones de casa.

[2]El rango de direcciones IP que reparte la UPV en el laboratorio es `158.42.181.XX`, mientras que haciendo NAT con VMware las obtenemos en el rango `192.168.182.13X`.

# 2. Configuración

Para el objetivo uno especificado, la configuración de las máquinas virtuales fue similar a la que usamos en la práctica. A modo de resumen, no especificaremos los cambios realizados a las máquinas virtuales del trabajo de PPTP a la práctica de Tun-Tap, pero sí destacaremos aquellos cambios que difieren de esta última práctica a la hora de hacer este nuevo entregable.

La configuración de la que partimos es la que sigue. Para `RCO`:




Para `RCO-X`:

```
[root@mv104 ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:50:56:1D:11:04
          inet addr:158.42.181.164  Bcast:158.42.181.255  Mask:255.255.254.0
          inet6 addr: fe80::250:56ff:fe1d:1104/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1744 errors:0 dropped:0 overruns:0 frame:0
          TX packets:328 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:175556 (171.4 KiB)  TX bytes:27259 (26.6 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:720 (720.0 b)  TX bytes:720 (720.0 b)
```

Como podemos comprobar en ambos `ifconfig (1)`, las dos máquinas se encuentran dentro de la misma red, la de la UPV, y son visibles entre ellas. Para levantar el túnel escribimos las siguientes órdenes. En `RCO`:

```
[root@mv203 ~]# tunctl -n -t tun0
Set 'tun0' persistent and owned by uid 0
[root@mv203 ~]# ip link set tun0 up
[root@mv203 ~]# ip addr add 10.119.17.1/30 dev tun0
```

Con estas órdenes creamos el túnel `tun0`, levantamos la respectiva interfaz y le asignamos una IP, en nuestro caso, `10.119.17.1`. Hay que tener en cuenta que, a no ser que hagamos ajustes de manera más específica, esta configuración se perderá cuando se reinicie la máquina. Una manera de que esto no ocurra es haciéndolo a través de `/etc/network/interfaces`:

```
iface tap1 inet manual
pre-up ip tuntap add tap1 mode tap user root
pre-up ip addr add 192.168.1.121/24 dev tap1
up ip link set dev tap1 up
post-up ip route del 192.168.1.0/24 dev tap1
post-up ip route add 192.168.1.121/32 dev tap1
post-down ip link del dev tap1
```

Paralelamente, en `RCO-X` procedimos de manera similar:

```
[root@mv104 ~]# tunctl -n -t tun3
Set 'tun3' persistent and owned by uid 0
[root@mv104 ~]# ip link set tun3 up
[root@mv104 ~]# ip addr add 10.119.17.2/30 dev tun3
```

En este caso, queremos que `RCO-X` sea el cliente y asignamos la IP correspondiente a él, `10.119.17.2`. Para comprobar que ambos interfaces se hayan creado de manera correcta, usamos `ifconfig` para visualizarlos. Para `RCO`:

```
[root@mv203 simpletun]# ifconfig tun0
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.119.17.1  P-t-P:10.119.17.1  Mask:255.255.255.252
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:16448 (16.0 KiB)  TX bytes:16448 (16.0 KiB)
```

Para RCO-X:

```
[root@mv104 ~]# ifconfig tun3
tun3      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.119.17.2  P-t-P:10.119.17.2  Mask:255.255.255.252
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:16448 (16.0 KiB)  TX bytes:16448 (16.0 KiB)
```

Antes de lanzar ambos túneles fue necesario desactivar `iptables (2)` en el servidor, ya que de otra manera el túnel nos puede dar problemas o directamente imposibilitar la conexión.

```
[root@mv203 simpletun]# iptables -F
```

Tras esto, pudimos empezar a abrir los túneles. Primeramente fue necesario abrir el servidor y posteriormente, el cliente. Como hemos especificado anteriormente, RCO servirá como servidor y RCO-X como cliente. Para abrir el túnel en RCO usamos la siguiente orden, tras acceder al directorio donde esté compilado `simpletun.c`:

```
[root@mv203 simpletun]# ./simpletun  -i tun0 -s
```

Tras esta orden, el servidor quedará a la espera de que un cliente se conecte en el puerto por defecto 55555. Hay que destacar que el programa, en la modalidad servidor, no requiere de ningún argumento adicional, simplemente hay que definir la interfaz que queramos utilizar. Para abrir el túnel del cliente utilizamos un orden similar:

```
[root@mv104 simpletun]# ./simpletun -i tun3 -c 158.42.181.34
```

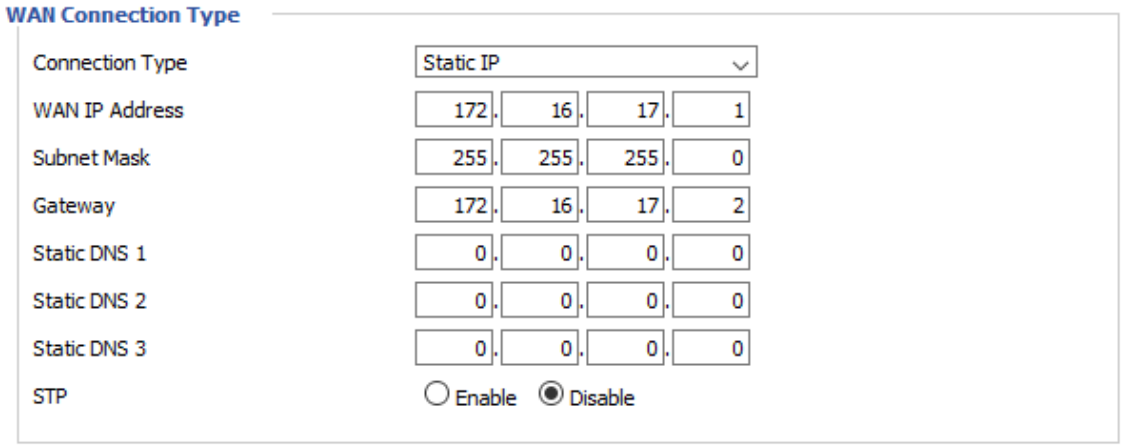
En este caso, el cliente sí requiere de una IP a la que conectarse. Será necesaria la IP pública del servidor, además del interfaz que ha de usarse. Si todo ha ido bien y no se ha lanzado el proceso en segundo plano, el túnel se quedará en bucle. Esto significa que la conexión se ha realizado de manera correcta y ya podremos utilizar el túnel.

Para el objetivo dos mantuvimos una estructura similar a la anterior. En este caso, fue necesario crear dos subredes para RCO y RCO-X y en estas dos subredes alojamos dos enrutadores, los mismos enrutadores que hemos usado en prácticas anteriores. La misión de este objetivo es que ambos enrutadores se puedan comunicar entre ellos a pesar de estar en dos redes distintas, gracias al túnel creado entre RCO y RCO-X.

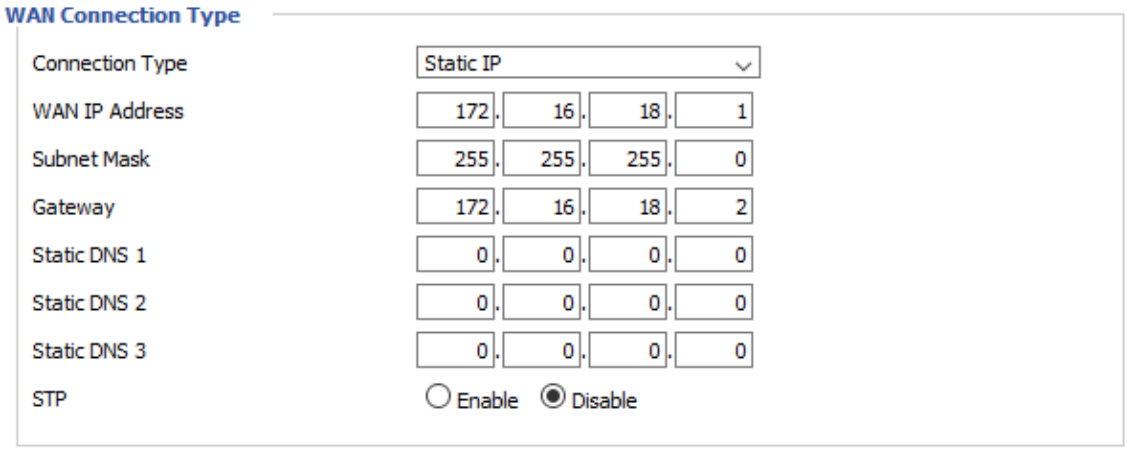
Hay que tener en cuenta que tanto `ddwrt-X` como `ddwrt-noX` deben estar en la subred correspondiente a su máquina virtual para que la red que hemos especificado tenga sentido y pueda haber conexión entre las distintas partes. Para conseguirlo, desde la configuración de VMware seleccionamos las distintas subredes (`VMnet2` y `VMnet3`) para cada uno de los encaminadores (`ddwrt-noX` y `ddwrt-X`, respectivamente).

En estos momentos, los enrutadores no tenían una IP congruente con la red que les corresponde. Por tanto, la configuramos manualmente desde la interfaz gráfica que disponen.

Dentro de esta interfaz, nos dirigimos a la pestaña *Setup*, a la subpestaña *Basic Setup* y al apartado *WAN Setup*. Una vez aquí, rellenamos los huecos como muestra la siguiente imagen. En el caso de `ddwrt-noX`:




Para `ddwrt-X`:




*WAN IP Address* corresponde a la nueva IP que poseerá el enrutador al pulsar *Apply*. Es importante destacar que en *Gateway* fue necesario poner la IP correspondiente a las

máquinas RCO y RCO-X, según convenga, ya que serán estas las que redirijan el tráfico de los enrutadores.

En RCO y RCO-X procedimos a hacer algunos cambios, ya que no poseen las IP configuradas para estar en la misma subred que los enrutadores. Para conseguir esto, debemos acceder al fichero de configuración de interfaces de red que se encuentra en /etc/sysconfig/network-scripts/ifcfg-eth1. En el caso de RCO, el fichero quedó así:


Para RCO-X:

Es importante, además de configurar la IP correspondiente, añadir la máscara adecuada. Esto lo hicimos añadiendo al fichero el campo `NETMASK`. En el caso de `RCO-X`, el fichero completo quedó de la siguiente manera:

```
  GNU nano 2.0.9 Fichero: ...c/sysconfig/network-scripts/ifcfg-eth1

DEVICE=eth1
BOOTPROTO=none
ONBOOT=yes
TYPE=Ehernet
NM_CONTROLED=yes
DEFROUTE=yes
PEERDNS=yes
IPV4_FAILURE_FATAL=yes
IPV6INIT=no
NAME="System eth1"
NETWORK=172.16.18.0
IPADDR=172.16.18.2
NETMASK=255.255.255.0
```

De manera análoga, realizamos el mismo procedimiento en `RCO`. Destacamos que tras guardar este fichero, la configuración no estará aplicada por completo. Para que esto sea así, debimos tumbar el interfaz `eth1` y volverlo a levantar usando las órdenes `ifdown` e `ifup`.

Tras esto, podemos probar a enviar `ping` entre los componentes de una misma red para comprobar su funcionamiento. Desde `ddwrt-noX` a `RCO`:

```
root@DD-WRT:~# ping 172.16.17.2
PING 172.16.17.2 (172.16.17.2): 56 data bytes
64 bytes from 172.16.17.2: seq=0 ttl=64 time=0.315 ms
64 bytes from 172.16.17.2: seq=1 ttl=64 time=0.766 ms
64 bytes from 172.16.17.2: seq=2 ttl=64 time=0.965 ms

--- 172.16.17.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.315/0.682/0.965 ms
root@DD-WRT:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.17.2     0.0.0.0         255.255.255.255 UH    0      0        0 eth0
192.168.17.0    0.0.0.0         255.255.255.0   U     0      0        0 br0
172.16.17.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     0      0        0 br0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         172.16.17.2     0.0.0.0         UG    0      0        0 eth0
root@DD-WRT:~# _
```

Desde `ddrwrt-X` a `RCO-X`:

```
root@DD-WRT:~# ping 172.16.18.2
PING 172.16.18.2 (172.16.18.2): 56 data bytes
64 bytes from 172.16.18.2: seq=0 ttl=64 time=0.322 ms
64 bytes from 172.16.18.2: seq=1 ttl=64 time=0.912 ms
64 bytes from 172.16.18.2: seq=2 ttl=64 time=0.947 ms

--- 172.16.18.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.322/0.727/0.947 ms
root@DD-WRT:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.18.2     0.0.0.0         255.255.255.255 UH    0      0        0 eth0
172.16.18.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.17.0    0.0.0.0         255.255.255.0   U     0      0        0 br0
169.254.0.0     0.0.0.0         255.255.0.0     U     0      0        0 br0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         172.16.18.2     0.0.0.0         UG    0      0        0 eth0
root@DD-WRT:~#
```

Obsérvese las tablas de enrutamiento, sobre todo la última, ya que será la que nos muestre que hemos realizado bien la configuración del *Gateway* en el interfaz gráfico del enrutador.

Una vez hecho esto, levantamos de nuevo los túneles como hicimos en el objetivo anterior y nos dispusimos a añadir entradas a la tabla de enrutamiento. Estas reglas en las tablas de enrutamiento de las máquinas virtuales `RCO` y `RCO-X` sirvieron para que supieran por qué interfaz enviar los paquetes que tuvieran como destino la red contigua. Para ello usamos la orden `route add` (3):

```
[root@mv203 ~]# route add -net 172.16.18.0 netmask 255.255.255.0 gw 10.119.17.1
[root@mv203 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.119.17.0     0.0.0.0         255.255.255.252 U     0      0        0 tun0
172.16.18.0     10.119.17.1     255.255.255.0   UG    0      0        0 tun0
192.168.182.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1003   0        0 eth0
0.0.0.0         192.168.182.2   0.0.0.0         UG    0      0        0 eth0
[root@mv203 ~]# _
```

Al comprobar la tabla de enrutamiento, nos aseguramos que se ha añadido de manera correcta. Esta entrada consigue que los paquetes que vayan dirigidos a la red `172.16.18.0` sean dirigidos por el túnel que tenemos en `tun0`. Añadir esta regla en `RCO-X` se realiza de la misma manera, aunque esta vez, cambiando la red destino por `172.16.17.0` y el *Gateway* por `10.119.17.2`:

```
[root@mv104 ~]# route add -net 172.16.17.0 netmask 255.255.255.0 gw 10.119.17.2
[root@mv104 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.119.17.0     0.0.0.0         255.255.255.252 U     0      0        0 tun3
192.168.182.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.16.18.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.16.17.0     10.119.17.2     255.255.255.0   UG    0      0        0 tun3
169.254.0.0     0.0.0.0         255.255.0.0     U     1002   0        0 eth1
169.254.0.0     0.0.0.0         255.255.0.0     U     1003   0        0 eth0
0.0.0.0         192.168.182.2   0.0.0.0         UG    0      0        0 eth0
```

Tras esto, fue necesario desactivar el firewall en todos los dispositivos de las redes con `iptables -F` y asegurarnos que las máquinas virtuales `RCO` y `RCO-X` tuvieran reenvío IP activado. Para ello, tuvimos que editar el fichero de configuración `ip_forward` en `/proc/sys/net/ipv4` sustituyendo el cero por un uno. Para `RCO`:

```
 GNU nano 2.0.9                                  Fichero: /proc/sys/net/ipv4/ip_forward

1
```

Para `RCO-X`:

```
 GNU nano 2.0.9      Fichero: /proc/sys/net/ipv4/ip_forward

1
```

De manera parecida a lo que ocurre con los túneles Tun-Tap, esta configuración de reenvío no es permanente, se reinicializa cada vez que la máquina se vuelve a encender. Para hacerlo permanente deberemos editar el fichero `sysctl.conf`, en el directorio `/etc/`:

```
 GNU nano 2.0.9          Fichero: /etc/sysctl.conf                    Modificado

kernel.msgmnb = 65536

# Controls the maximum size of a message, in bytes
kernel.msgmax = 65536

# Controls the maximum shared segment size, in bytes
kernel.shmmax = 68719476736

# Controls the maximum number of shared memory segments, in pages
kernel.shmall = 4294967296

#Enable IP Forwarind permanent
net.ipv4.ip_forward = 1
```

Una vez todas estas configuraciones fueron realizadas, solo nos quedaba probar la conexión entre los dos enrutadores.

# 3. Funcionamiento del programa simpletun

Para probar que el primer objetivo era funcional, enviamos un ping desde `RCO` a `RCO-X` usando la orden `ping -s 1000 10.119.17.2` (4) y capturamos el intercambio de paquetes con Wireshark[3].

Escuchamos tanto la interfaz `eth0` como la interfaz `tun0` para poder hacer un seguimiento más exhaustivo y acertado, y seguir de manera real el viaje de un ping usando túneles Tun-Tap. Usando los filtros (5) que Wireshark ofrece, fuimos capaces de separar las distintas interfaces debido a que en cada una de ellas se observa un encapsulado distinto, gracias al inherente funcionamiento de `simpletun`.

En el caso de la interfaz `eth0` se observa:

| | | | | | |
|---|---|---|---|---|---|
| 305 | 11.96047720 | 158.42.181.164 | 158.42.181.34 | TCP | 1094 60314 > 55555 [PSH, ACK] Seq=2063 Ack=2061 Win=646 Len=1028 TSval=886527 TSecr=873216 |
| 306 | 11.96107634 | 158.42.181.34 | 158.42.181.164 | TCP | 66 55555 > 60314 [ACK] Seq=2061 Ack=3091 Win=676 Len=0 TSval=873217 TSecr=886527 |
| 307 | 11.96115542 | 158.42.181.34 | 158.42.181.164 | TCP | 68 55555 > 60314 [PSH, ACK] Seq=2061 Ack=3091 Win=676 Len=2 TSval=873217 TSecr=886527 |
| 308 | 11.96117796 | 158.42.181.164 | 158.42.181.34 | TCP | 66 60314 > 55555 [ACK] Seq=3091 Ack=2063 Win=646 Len=0 TSval=886528 TSecr=873217 |
| 310 | 11.96179211 | 158.42.181.34 | 158.42.181.164 | TCP | 1094 55555 > 60314 [PSH, ACK] Seq=2063 Ack=3091 Win=676 Len=1028 TSval=873218 TSecr=886528 |
| 311 | 11.96187401 | 158.42.181.164 | 158.42.181.34 | TCP | 66 60314 > 55555 [ACK] Seq=3091 Ack=3091 Win=678 Len=0 TSval=886529 TSecr=873218 |
| 338 | 12.92217903 | 158.42.181.164 | 158.42.181.34 | TCP | 68 60314 > 55555 [PSH, ACK] Seq=3091 Ack=3091 Win=678 Len=2 TSval=887489 TSecr=873218 |
| 340 | 12.96360369 | 158.42.181.34 | 158.42.181.164 | TCP | 66 55555 > 60314 [ACK] Seq=3091 Ack=3093 Win=676 Len=0 TSval=874219 TSecr=887489 |

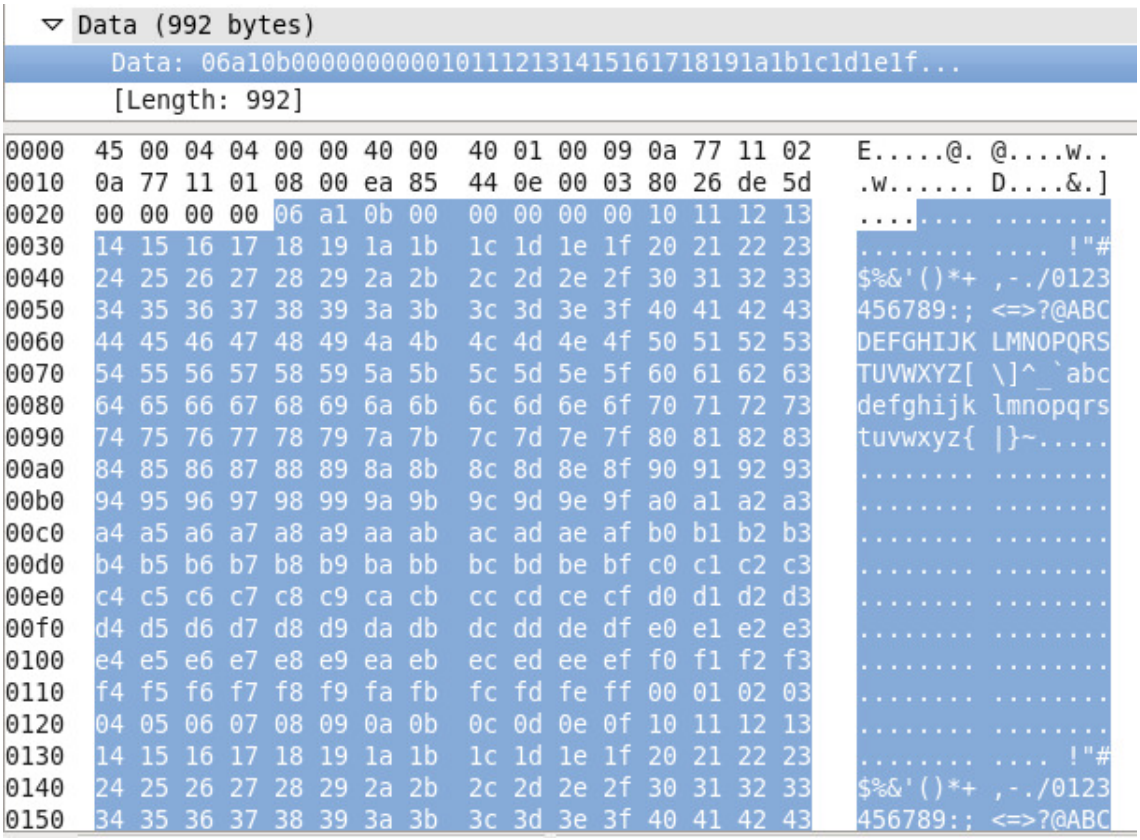
Mientras que en la interfaz `tun0` observamos:

| | | | | | |
|---|---|---|---|---|---|
| 302 | 11.91737267 | 10.119.17.2 | 10.119.17.1 | ICMP | 1028 Echo (ping) request id=0x440e, seq=3/768, ttl=64 |
| 309 | 11.96200910 | 10.119.17.1 | 10.119.17.2 | ICMP | 1028 Echo (ping) reply id=0x440e, seq=3/768, ttl=64 |
| 337 | 12.92205552 | 10.119.17.2 | 10.119.17.1 | ICMP | 1028 Echo (ping) request id=0x440e, seq=4/1024, ttl=64 |
| 345 | 12.96510737 | 10.119.17.1 | 10.119.17.2 | ICMP | 1028 Echo (ping) reply id=0x440e, seq=4/1024, ttl=64 |
| 374 | 13.92685212 | 10.119.17.2 | 10.119.17.1 | ICMP | 1028 Echo (ping) request id=0x440e, seq=5/1280, ttl=64 |
| 381 | 13.97026112 | 10.119.17.1 | 10.119.17.2 | ICMP | 1028 Echo (ping) reply id=0x440e, seq=5/1280, ttl=64 |

En la interfaz respectiva al túnel no es muy notorio debido al poco tráfico que se genera, pero en la `eth0`, gracias al tamaño de 1000 bytes que hemos especificado en la declaración del ping, podemos diferenciar de forma bastante acertada qué paquetes corresponden a los `ping` que estamos mandando entre todo el tráfico que escucha esta interfaz.

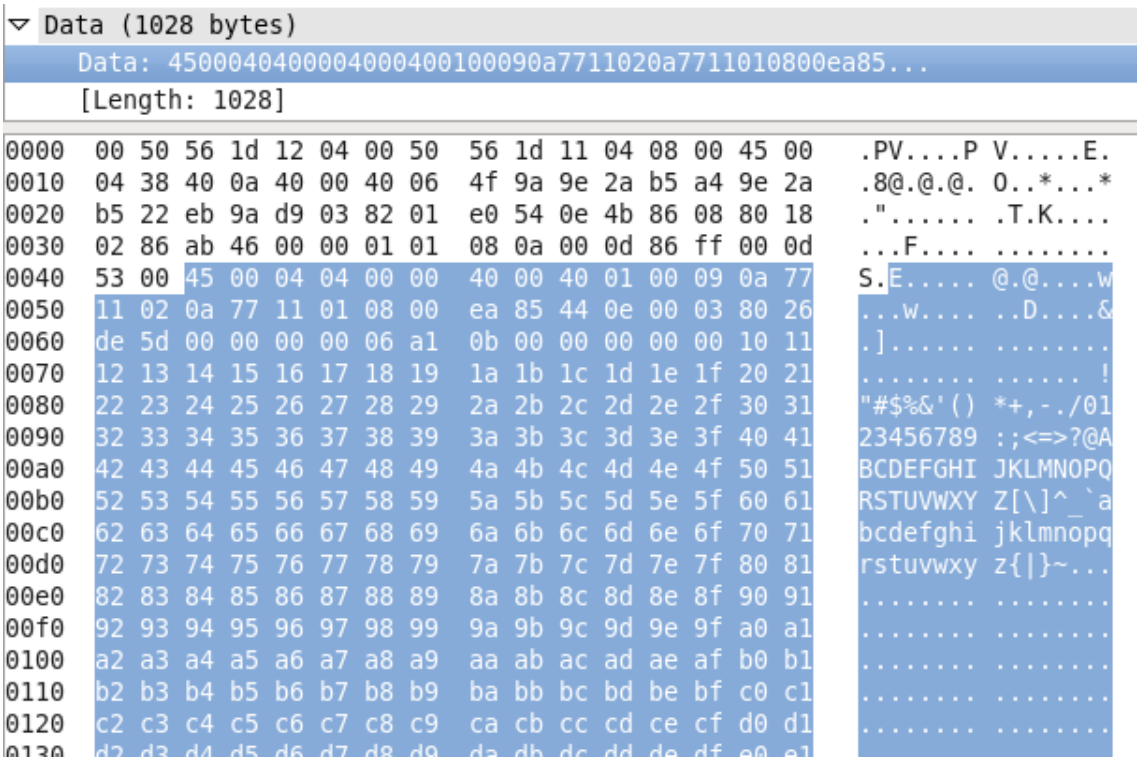
[3] www.wireshark.com

No obstante y para asegurarnos, vamos a comprobar el contenido de los paquetes, empezando por los de la interfaz `tun0`:



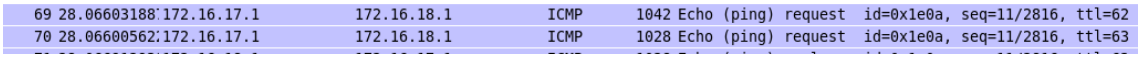

En la captura se ha seleccionado el texto correspondiente al ping. Los 28 bytes de datos anteriores a este coinciden con el tamaño mínimo de la cabecera IP (20 bytes (6)) junto con los de la cabecera ICMP (8 bytes (7)).

Si comprobamos los datos escuchados por la interfaz `eth0`, podemos ver algo similar:

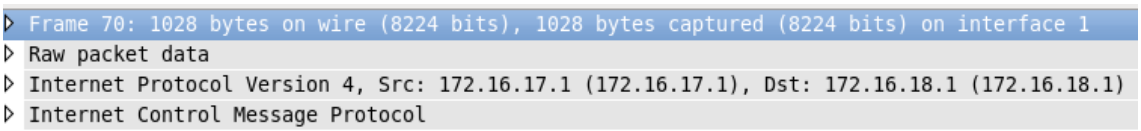



En este caso vemos que la longitud de los datos son 1028 bytes. Esta longitud es igual a la anterior ya que se trata del mismo paquete encapsulado dentro de TCP, de ahí que el tamaño total del paquete sea 1094 bytes. Por si la similitud de tamaños no fuera suficiente, el contenido del datagrama es idéntico al anterior, y es debido a eso por lo que podemos asegurar que se trata del ping que hemos enviado.

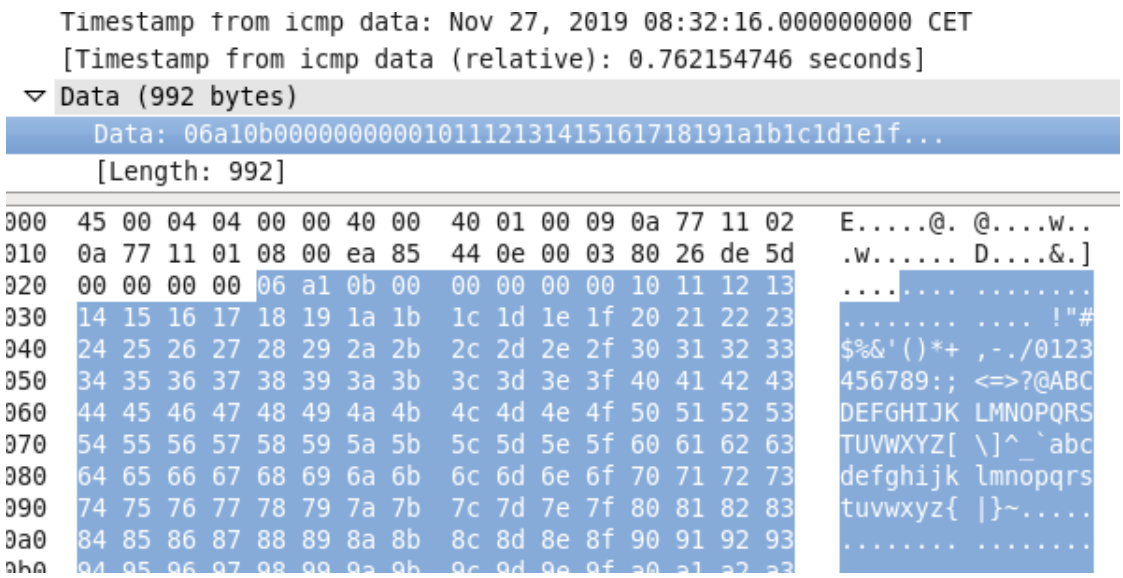
En el objetivo dos trata de conectar dos dispositivos de dos redes separadas mediante un túnel similar al primer objetivo. La metodología que hemos usado en este caso es igual al anterior. Usando un ping de un tamaño dado, seguiremos su camino de enrutador a enrutador; en el interfaz `tun0` podemos observar el cambio de dirección destino.



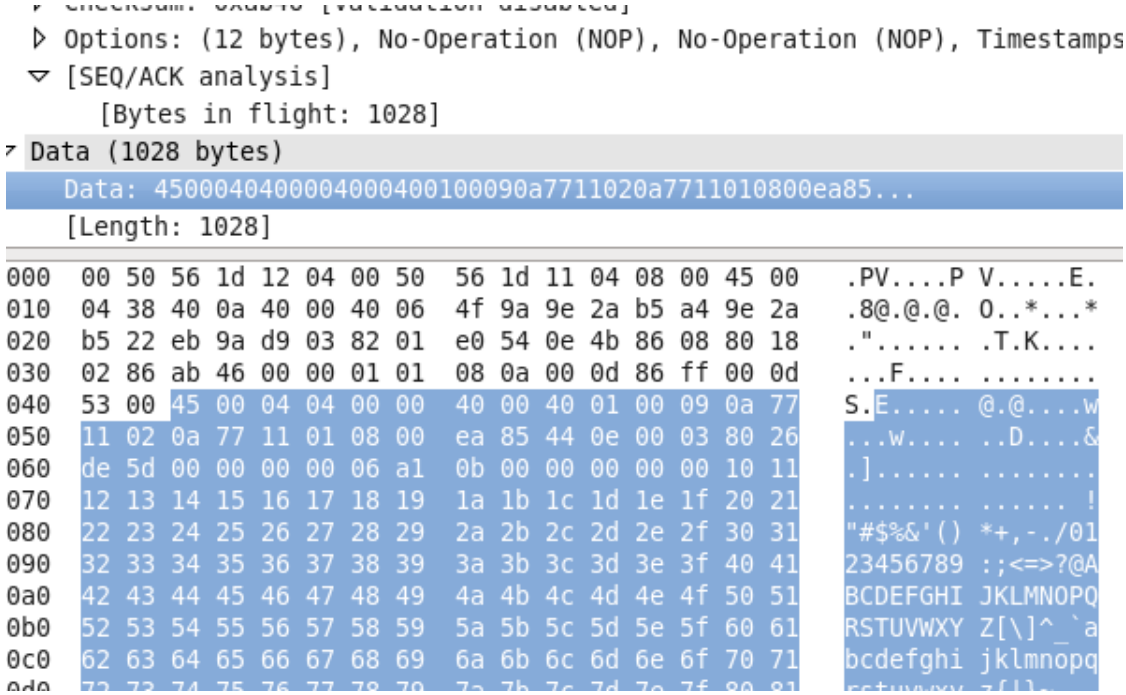

Su contenido:



De manera análoga al anterior caso, la observación de sus datagramas nos ofrece una buena base para comparar. En la interfaz `tun0`:



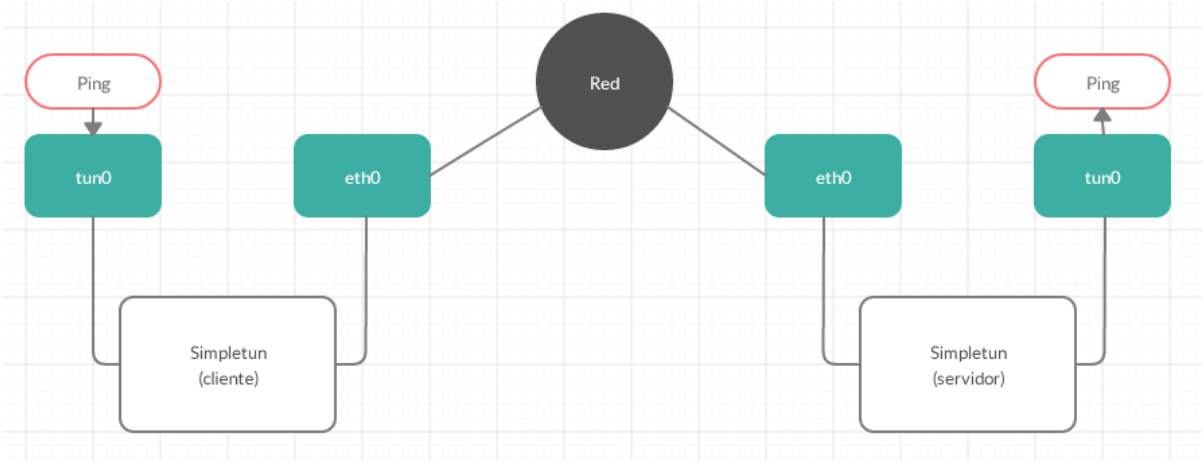

En la interfaz `eth0`:




Debido a la explicación que hemos dado anteriormente, podemos claramente diferenciar los ping y seguir la ruta que siguen desde el origen al destino pasando por el túnel.

Pero, ¿por qué vemos IP diferentes según la interfaz que estemos escuchando? ¿Por qué hay distintas encapsulaciones? Esto, como hemos mencionado al principio de este apartado tiene que ver con el propio comportamiento de `simpletun`. Cuando se realiza un ping a la red vecina, en nuestro caso la tabla de enrutamiento lo manda por el túnel. En el momento en el que el programa detecta un mensaje, lo encapsula en un segmento TCP y lo manda por la interfaz `eth0`. Por ahora, este mensaje no irá cifrado, pero en apartados posteriores veremos cómo cambiar esto.

Tras esto, `eth0` envía el mensaje por internet hasta que llega al destino, donde el proceso se invierte: en este caso, el programa del destinatario desempaqueta el mensaje y deja al sistema operativo que procese el ping. Después se produce la respuesta, que hace el mismo camino pero en sentido contrario.



# 4. Cifrado Cesar

El cifrado Cesar es una de las técnicas de cifrado más antiguas de nuestra historia, y por ser de las primeras, también es de las más simples. Recibe su nombre en honor al emperador romano Julio Cesar, al emplear desplazamientos de 3 letras en sus mensajes de contenido militar. A pesar de que hoy en día un desplazamiento de 3 letras puede parecer una cosa muy simple, debemos recordar que en la época muy poca gente sabía leer, y mucho menos tenía conocimientos de análisis criptográfico. Fue usado por la Armada Rusa antes de la Primera Guerra Mundial, aunque a las tropas alemanas y austriacas no les costó demasiado decodificar los mensajes (8). Hoy en día puede ser descifrado de dos maneras distintas. En caso de que se nos presentara un texto y no supiéramos que tipo de cifrado tiene, lo más sencillo sería aplicar un ataque por fuerza bruta. Un programa probaría distintos cifrados hasta que encontrara un resultado coherente comparándolo con un diccionario. En caso de saber que se está empleando este cifrado, con mirar una o dos palabras y probar desplazamientos o a simple vista podemos descifrar el mensaje y saber cuál es el desplazamiento. Una vez hecho esto descifrar la totalidad del mensaje se simplifica bastante.

En el laboratorio se nos asignó el enrutador número 17, por tanto, en las comunicaciones del túnel, todos los octetos del mensaje tendrán 17 desplazamientos hacia la derecha en el alfabeto. Como un mensaje de este tipo no se resume únicamente a caracteres del alfabeto, además de poder contener mayúsculas y minúsculas, hemos tomado como referencia la tabla de caracteres ASCII para cubrir todas las posibilidades.

La información que se lee del túnel se guarda en la variable buffer, un vector de `char` que tiene una longitud predefinida de 2000 unidades. Por tanto, habrá que recorrer el vector y aplicar el desplazamiento en cada campo.

Para ello el código será el siguiente:

```
/*CIFRAR*/
int counter = 0;
while(counter<BUFSIZE){
        buffer[counter]=(buffer[counter]+17)%256;
        counter++;
}
```

Afortunadamente C hace la conversión de tipos automáticamente de `char` a su correspondiente código ASCII así que sumando 17 directamente obtenemos la cifra deseada. Hay que prestar atención a que, si estamos a menos de 17 unidades del 255, nos pasaríamos del rango de caracteres ASCII. Para ello, al final de la expresión añadiremos un módulo 256, así volverá a contar desde cero las unidades que nos hemos pasado de largo.

En el caso de descifrar es muy parecido, para recuperar el desplazamiento de antes debemos restar las 17 unidades que hemos sumado antes de mandar el paquete por el túnel. En este caso, hay que tener cuidado especialmente que, si nos encontramos en el principio del rango de caracteres ASCII, nos saldremos por delante, es decir, el código puede llegar a ser negativo, y por tanto, no válido para su posterior conversión a un carácter. Para ello tras restar las unidades correspondientes sumaremos 256 (el total de caracteres ASCII) para devolverlo a una posición positiva.

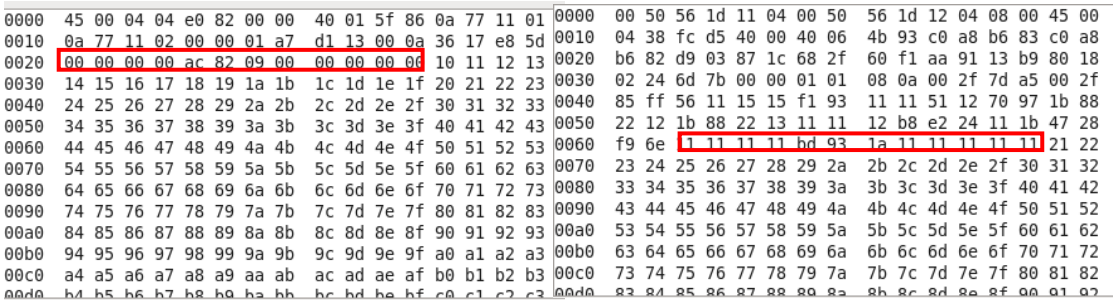
El código quedaría de la siguiente manera:

```
/*DESCIFRAR*/
int counter1 = 0;
while(counter1<BUFSIZE){
        buffer[counter1]=(buffer[counter1]-17+256)%256;
        counter1++;
}
```

En cuanto a comprobar el contenido del ping, al capturar las tramas con Wireshark podemos comprobar que en el paquete TCP (el encapsulado por Tun) tenemos como datos el tamaño del paquete ICMP que capturamos de `eth0` como podemos ver a continuación:

```
   ICMP      1028 Echo (ping) reply    id=0xd113, seq=10/2560, ttl=64
   TCP       1094 55555 > 34588 [PSH, ACK] Seq=3 Ack=1031 Win=548 Len=1028 TSval=3112357 TSecr=3114495
```

Y en el contenido del ping podemos observar que los bits se desplazan por la propia cabecera del paquete (`eth0` a la izquierda y `tun0` a la derecha):



# 5. Cifrado con XOR

En lógica, una disyunción exclusiva obtendría todos los elementos que sean A o B pero que no sean comunes a ambos. En electrónica y en operadores binarios, que es lo que nos interesa, arroja un 1 como resultado si alguno de los dos operandos es 1, pero si ambos lo son, arroja un 0. Se puede ver de manera sencilla en esta tabla:

| INPUT | | OUTPUT |
|---|---|---|
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

En nuestro caso, cogerá dos cadenas y las comparará bit a bit, el resultado será lo que mandaremos a la interfaz Tun. La particularidad de usar XOR como cifrado es que, aplicándolo una vez, cogemos ese resultado y le volvemos a aplicar XOR, tenemos de vuelta la cadena inicial. Por tanto, la operación de cifrado y descifrado es igual. Actualmente es un cifrado muy en desuso debido a que tiene una complejidad muy baja, pero sí que se emplea como parte de otros cifrados más complejos. Por ejemplo, el cifrado Vernam emplea un cifrado XOR en su método (9). Añade más seguridad haciendo que la clave con la que se cifra mediante XOR sea pseudoaleatoria, y así no sea tan fácil de adivinar comparando unos cuantos paquetes.

El lenguaje de programación C incluye entre sus funciones, aparte de los operadores aritméticos, los booleanos. En el caso de la XOR es el circunflejo. Para completar la operación deberemos guardar un carácter (que será nuestra clave) en el programa para luego hacer la operación con el carácter que nos corresponda dentro del vector `buffer`. En esta ocasión C hace la conversión de tipos automáticamente y no es necesario pasar de carácter a código ASCII de manera manual.

Por tanto, el código que emplearemos para cifrar es el siguiente:

```
/*CIFRAR*/
int counter = 0;
char key = 'v';
while(counter<BUFSIZE){
        buffer[counter]=buffer[counter]^key;
        counter++;
}
```

Como hemos comentado anteriormente, para cifrar y descifrar la operación es la misma. Debemos aplicar la operación XOR con la misma clave y obtendremos el carácter que teníamos anteriormente. Es posible que para hacer pruebas no podamos ver el contenido de la variable ya cifrada, ya que puede ser uno de los códigos que no se pueden mostrar de manera correcta en nuestra distribución local.

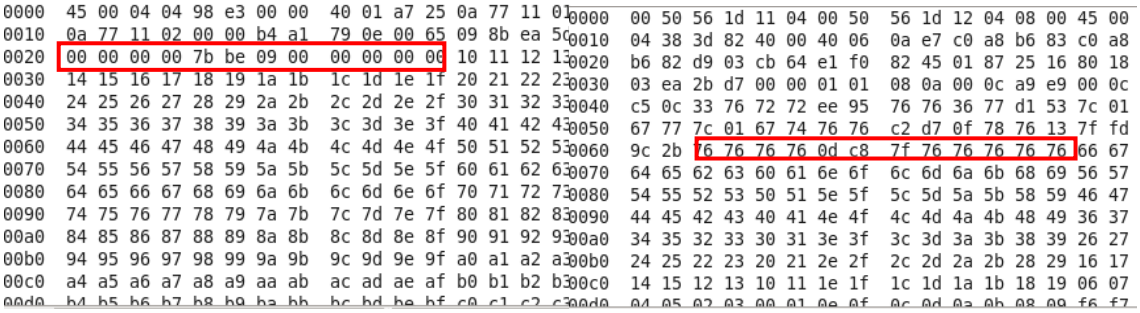
Entonces el código para el descifrado es totalmente igual:

```
/*DESCIFRAR*/
int counter1 = 0;
char key2 = 'v';
while(counter1<BUFSIZE){
        buffer[counter1]=buffer[counter1]^key2;
        counter1++;
}
```

En cuanto a la comprobación observamos algo parecido al apartado anterior, los paquetes TCP/IP tienen como datos la totalidad del ICMP capturado en la interfaz `eth0` como podemos ver a continuación:

```
ICMP    1028 Echo (ping) reply    id=0x790e, seq=101/25856, ttl=64
TCP     1094 55555 > 52068 [PSH, ACK] Seq=4123 Ack=5151 Win=1002 Len=1028 TSval=829929 TSecr=836876
```

En cuanto al contenido, en esta ocasión es algo más difícil de comprobar puesto que la conversión de caracteres no es siempre perfecta y además no siguen una linealidad como el desplazamiento del cifrado Cesar. De todas formas, sí se puede observar el mero desplazamiento de los datos por incluir la cabecera TCP/IP en el paquete ya capturado en el túnel y los mismos separadores que hemos señalado antes, cuatro pares de ceros, seguidos de tres pares con contenido y otros cinco pares de ceros:



# 6. Conclusiones

Siendo realistas, las posibilidades y opciones que ofrece Tun-Tap son realmente numerosas, más si nos vamos al mercado de VPN y vemos que muchas de las soluciones comerciales que se ofrecen hoy en día usan este tipo de socket virtual para realizar conexiones. También es cierto que la solución tecnológica "a pelo" como hemos visto en la tarea uno y dos no tiene ningún tipo de protección o cifrado. Está claro que las soluciones comerciales de las que hemos hablado antes sí que incluyen este tipo de características, y esto lo hacen de manera similar a la que hemos hecho nosotros en la tarea tres y cuatro. Por ejemplo, OpenVPN ofrece SSL/TLS para cifrar las comunicaciones (10) y sin embargo ofrece también *Ethernet Bridging* mediante interfaces TAP (11).

Saliendo del uso comercial, vemos un gran potencial a nivel formación y aprendizaje, ya que está el código en C listo para compilar. Un vistazo rápido a los comentarios del código y un manejo de este lenguaje nos puede dar posibilidades de mejorar o modificar el código de acuerdo con nuestras necesidades o requerimientos. También destacamos que al ser interfaces creadas en el nivel 3 de red, y ser una herramienta nativa del núcleo de Linux, no es necesario instalar ningún controlador para su funcionamiento aparte del `tunctl` que hemos hecho nosotros para poder crear las interfaces.

También hemos encontrado buscando información sobre esta herramienta numerosos usuarios que se plantean el uso de ésta en vez de emplear soluciones comerciales, no sabemos ni por experimentación o porque consideran que el uso que le van a dar no compensa a la compra de licencias comerciales (aun habiendo soluciones de software libre).

En resumen, vemos esta herramienta bastante interesante dadas las opciones que ofrece a ser manejadas por el usuario final, así como que huye de soluciones integradas en otros sistemas (si obviamos las VPN comerciales) como pudimos ver anteriormente con los túneles PPTP y EoIP. Además, vemos su configuración como algo simple si sólo se quieren tocar unas cuantas cosas y no montar de una herramienta tan simple una muy potente o que se quiera aproximar a VPN comerciales al uso.

## Referencias


1. **Wikipedia.** Ifconfig. [Online] Wikipedia. https://es.wikipedia.org/wiki/Ifconfig.

2. **ArchLinux.** iptables. *ArchWiki*. [Online] Diciembre 2, 2019. https://wiki.archlinux.org/index.php/Iptables.

3. **Gite, Vivek.** Linux route Add Command Examples. *nixCraft*. [Online] Julio 25, 2018. https://www.cyberciti.biz/faq/linux-route-add/.

4. **Wikipedia.** Ping (Parámetros). *Wikipedia*. [Online] Junio 27, 2019. https://es.wikipedia.org/wiki/Ping#Par%C3%A1metros_2.

5. **Wireshark.** CaptureFilters. *Wireshark*. [Online] Octubre 19, 2016. https://wiki.wireshark.org/CaptureFilters.

6. **Wikipedia.** IPv4. *Wikipedia*. [Online] Diciembre 4, 2019. https://en.wikipedia.org/wiki/IPv4#Header.

7. —. ICMP. *Wikipedia*. [Online] Diciembre 2, 2019. https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#Header.

8. **khan, David.** *The Codebreakers*. 1967. p. 631.

9. **Ramswarup, Kulhary.** Vernam Cipher in Cryptography. *GeeksforGeeks*. [Online] Septiembre 8, 2019. https://www.geeksforgeeks.org/vernam-cipher-in-cryptography/.

10. **OpenVPN.** Hardening VPN security. [Online] https://openvpn.net/community-resources/hardening-openvpn-security/.

11. —. Ethernet Bridging. [Online] https://openvpn.net/community-resources/ethernet-bridging/.