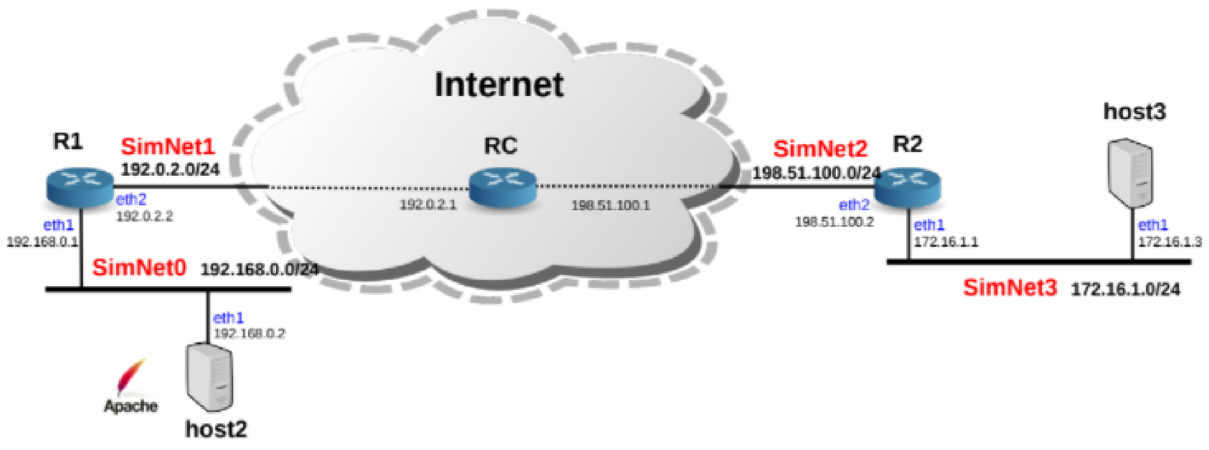


Starting the scenario using simctl

First of all, we will start the virtual scenario with the command:

```
host$ simctl iptunnel start
```

Configuring IPIP tunnels

Now we are going to configure the tunnel of the scenario. Several commands should be executed in the border routers (encapsulator y decapsulator) to properly configure a tunnel. In particular, in each border router you have to:

- Configure the tunnel interface by means of the command ip tunnel (consult the man page of this command).
  We will start configuring the tunnels to work in ipip mode, and with the option nopmtudisc set to avoid the algorithm of Path MTU Discovery. Notice that this latest option force the tunnel to also set the option ttl inherit. As an example, in R2 you have to execute:

```
R2$ ip tunnel add tunnel0 mode ipip local 198.51.100.2 remote 192.0.2.2 ttl 0 nopmtudisc
dev eth2
```

- Activate the new virtual interface of type tunnel. To do so, you have to assign an IP address to it (for instance, by means of the ifconfig command). In linux, it is necessary to assign an IP address to any interface if we want it to properly work. In this lab session, we are not going to use this address, so no matter which address is configured. Just as an example:

```
R2$ ifconfig tunnel0 1.2.3.4
```

  //s'ha d'assignar una @IP per a que funcioni

- Update the routing table to make the proper packets to be routed towards the tunnel (for instance, by means of the route command). Just as an example:

```
R2$ route add -net 192.168.0.0/24 dev tunnel0
```

  //@privada

  Now that you know how to do it, create the IPIP tunnel between R1 and R2. Remember that it is better to store in a file all the commands used to configure the system, so you can reuse them later.

```
R1#: ip tunnel add tunnel0 mode ipip local 192.0.2.2 remote
198.51.100.2 ttl 0 nopmtudisc dev eth2
R1#: ifconfig tunnel0 1.2.3.4
R1#: route add -net 172.16.1.0/24 dev tunnel0
```

TCGI slides 12
https://drive.google.com/drive/folders/1ugbAayGfGTtjjVX0rLdREecVnr-qNXcN?usp=sharing

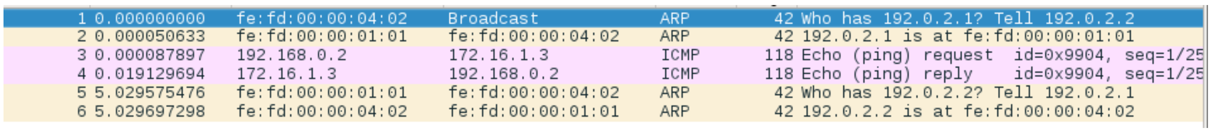
Testing the tunnels: protocol and TTL fields

Once this configuration has been applied in the edge routers, we will test if the tunnel is working properly:

1. Open FOUR wireshark network analyzers in the PHYSICAL HOST and capture traffic in each of these networks: SimNet0, SimNet1, SimNet2 and SimNet3.
   To properly see fragmented messages, DISABLE the following option in all these wiresharks: Edit-Preferences-Protocols-IPv4-Reassenble fragmented IPv4 datagrams.
2. Send the following ping from host2 to host3:

```
host2$ ping -c1 172.16.1.3
```

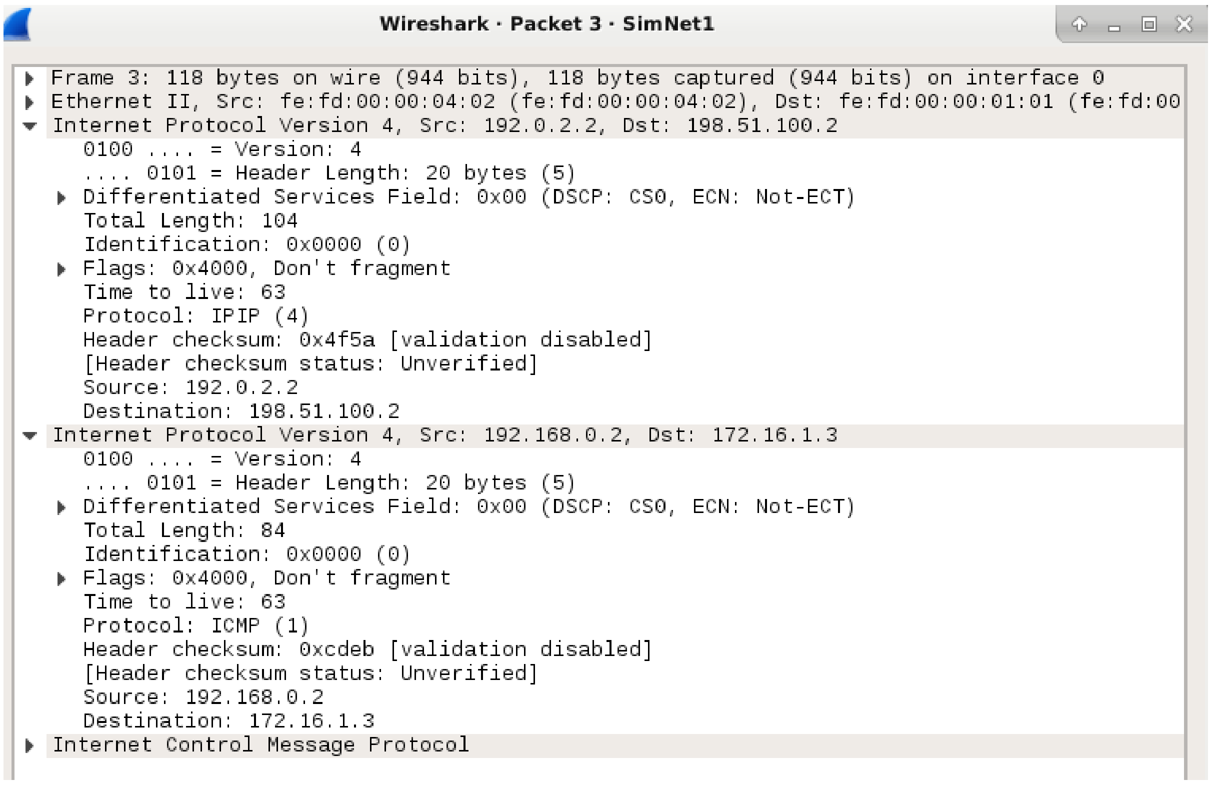
   a. What is the meaning of the -c option? (consult the man page of ping if necessary)
   b. Is the ping working? How many packets can you see?
      Veiem que arriba, per tant hem configurat bé el tunel. Si limitem al wireshark que no ens ensenyi els paquets fragmentats, llavors només veiem el echo-request i el echo-reply.



```
1 0.000000000   fe:fd:00:00:04:02   Broadcast           ARP    42 Who has 192.0.2.1? Tell 192.0.2.2
2 0.000050633   fe:fd:00:00:01:01   fe:fd:00:00:04:02   ARP    42 192.0.2.1 is at fe:fd:00:00:01:01
3 0.000087897   192.168.0.2         172.16.1.3          ICMP  118 Echo (ping) request  id=0x9904, seq=1/25
4 0.019129694   172.16.1.3          192.168.0.2         ICMP  118 Echo (ping) reply    id=0x9904, seq=1/25
5 5.029575476   fe:fd:00:00:01:01   fe:fd:00:00:04:02   ARP    42 Who has 192.0.2.2? Tell 192.0.2.1
6 5.029697298   fe:fd:00:00:04:02   fe:fd:00:00:01:01   ARP    42 192.0.2.2 is at fe:fd:00:00:04:02
```

As you can see, the -c option allows us to configure the number of ICMP echo-request messages sent, only one in this particular case. If you properly made the configuration of the tunnel, the ping should work. If the ping is not working, review your configuration until it works.

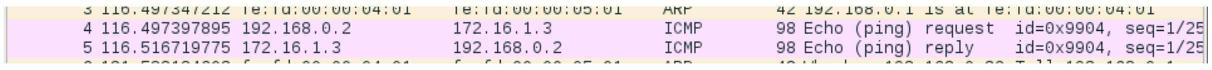
3. Assuming that the ping is working, it is necessary to emphasize certain aspects related to tunneling issues:
   a. Verify that the IP packet in SimNet1 and SimNet2 is encapsulated using IPIP. How many IP headers can you see in these packets? Which is the value of the "Protocol" field in the Outer Header?



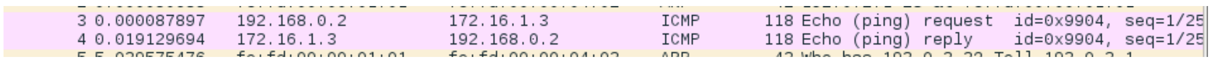
Wireshark · Packet 3 · SimNet1

```
▶ Frame 3: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
▶ Ethernet II, Src: fe:fd:00:00:04:02 (fe:fd:00:00:04:02), Dst: fe:fd:00:00:01:01 (fe:fd:00
▼ Internet Protocol Version 4, Src: 192.0.2.2, Dst: 198.51.100.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 104
    Identification: 0x0000 (0)
  ▶ Flags: 0x4000, Don't fragment
    Time to live: 63
    Protocol: IPIP (4)
    Header checksum: 0x4f5a [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.0.2.2
    Destination: 198.51.100.2
▼ Internet Protocol Version 4, Src: 192.168.0.2, Dst: 172.16.1.3
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x0000 (0)
  ▶ Flags: 0x4000, Don't fragment
    Time to live: 63
    Protocol: ICMP (1)
    Header checksum: 0xcdeb [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.0.2
    Destination: 172.16.1.3
▶ Internet Control Message Protocol
```

   b. Determine the difference in size from the original IP packet (in SimNet0) and the encapsulted one (in SimNet1).



```
  3 116.497347212 fe:fd:00:00:04:01    fe:fd:00:00:05:01    ARP      42 192.168.0.1 is at fe:fd:00:00:04:01
  4 116.497397895 192.168.0.2          172.16.1.3           ICMP     98 Echo (ping) request  id=0x9904, seq=1/25
  5 116.516719775 172.16.1.3           192.168.0.2          ICMP     98 Echo (ping) reply    id=0x9904, seq=1/25
```



```
  3 0.000087897   192.168.0.2          172.16.1.3           ICMP    118 Echo (ping) request  id=0x9904, seq=1/25
  4 0.019129694   172.16.1.3           192.168.0.2          ICMP    118 Echo (ping) reply    id=0x9904, seq=1/25
```

La diferència de mida es de 20 bytes (IP header)

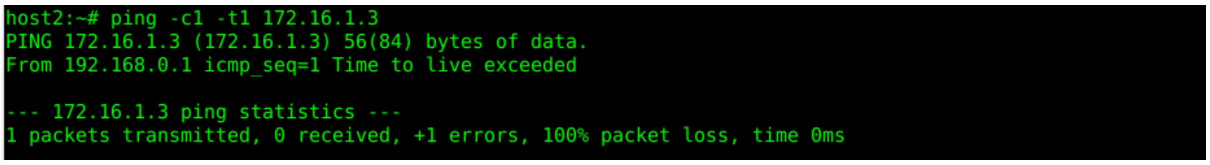
c. Identify the value of the “TTL” field in the original IP packet and how this value envolve during the transmission, both in the inner and in the outer headers. Remind that we used the option ttl inherit during the creation of the tunnels.

TTL	SimNet0	SimNet1	SimNet2	SimNet3
Outer header	64	63	62	62
Inner header	-	63	63	-

Si ens fixem en el TTL dels paquets, veiem com l'encapsulament IP fa que el paquet exterior si que vagi decrementant el seu TTL a mesura que passa pels diferents punts, però el paquet interior no decrementa el TTL.

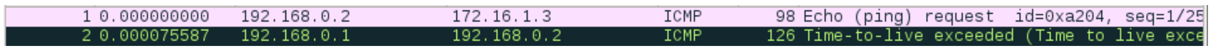
4. To evaluate the behavior of the tunnel when the TTL expires, we will send IP packets from host2 (192.168.0.2) to host3 (172.16.1.3) increasing one by one the TTL. First, clear and start again the capture in the four wiresharks to perfectly monitor step by step the transmission of this packets in all the involved networks (SimNet0, SimNet1, SimNet2 and SimNet3). Next, send again one ICMP echo-request message using the ping program and TTL=1.

```
host2$ ping  −c1 −t1  172.16.1.3
```



```
host2:~# ping -c1 -t1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 56(84) bytes of data.
From 192.168.0.1 icmp_seq=1 Time to live exceeded

--- 172.16.1.3 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

SimNet0:



```
  1 0.000000000   192.168.0.2          172.16.1.3           ICMP     98 Echo (ping) request  id=0xa204, seq=1/25
  2 0.000075587   192.168.0.1          192.168.0.2          ICMP    126 Time-to-live exceeded (Time to live exce
```

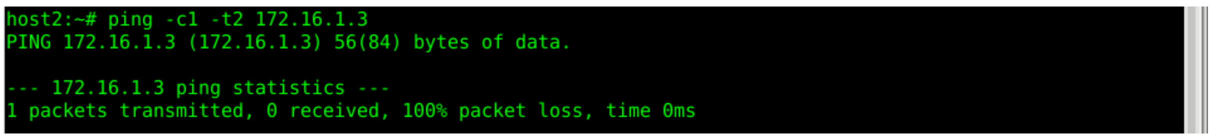
a. Can you see packets in all the networks? Did the packet reach the destination host? In which host the packet was lost?

b. Can you see any ICMP error message? Which one? In which network?

c. Which are the origin and destination IP addresses of this error message?

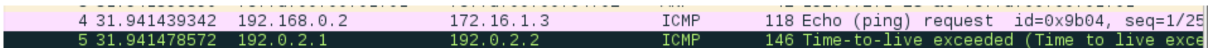
Veiem com el paquet es perd al primer salt, a R1. Només observem tràfic a SimNet0, i veiem un missatge ICMP de retorn que diu: TTL exceeded. Source: 192.168.0.1 (R1) Destination: 192.168.0.2 (host2)

Now clear all the wiresharks and send again another ping, but with TTL=2.



```
host2:~# ping -c1 -t2 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 56(84) bytes of data.

--- 172.16.1.3 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

SimNet1:



```
  4 31.941439342  192.168.0.2          172.16.1.3           ICMP    118 Echo (ping) request  id=0x9b04, seq=1/25
  5 31.941478572  192.0.2.1            192.0.2.2            ICMP    146 Time-to-live exceeded (Time to live exce
```

a. Did the packet reach the destination host? In which machine the packet was lost?

b. Can you see any ICMP error message? Which one? In which network?

c. Which is the origin and destination IP addresses of this error message?

d. Can you see any ICMP relaying?

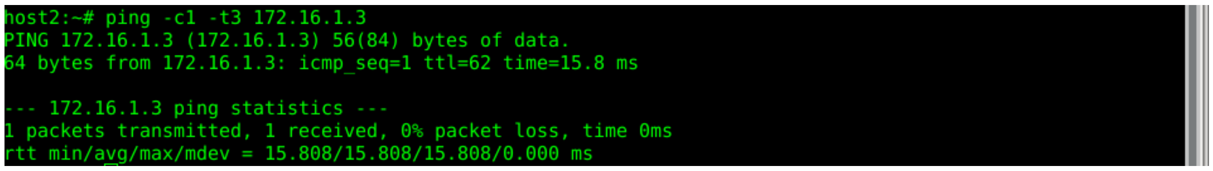
e. Send again the same ping with TTL=2. Is there any soft state?

TCGI slides 12
https://drive.google.com/drive/folders/1ugbAayGfGTtjjVX0rLdREecVnr-qNXcN?usp=sharing

f. Is this behavior compliant with RFC 2003 (see section 4.4)?
El paquet no arriba a destí i es perd a RC amb IP 192.0.2.1. Ho podem comprovar a la SimNet1, on observem el missatge de TTL exceeded.

Now use TTL=3:

a. Did the packet reach the destination host?
b. Can you see any ICMP reply message?



```
host2:~# ping -c1 -t3 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 56(84) bytes of data.
64 bytes from 172.16.1.3: icmp_seq=1 ttl=62 time=15.8 ms

--- 172.16.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.808/15.808/15.808/0.000 ms
```

Testing the tunnels: nopmtudisc

Limiting the MTU

1. Use ifconfig to see which is the value of the MTU in the tunnel inferfaces both in R1 and R2.
   a. Which is this value?
   b. Is there any relationship between the MTU assigned to the tunnel, and the MTU assigned to the physical interface?
   El valor de l'MTU en els dos costats del tunel és de 1480B (1500 MTU eth - 20 Header)
2. Delete the previously configured tunnel in R2. Instead of doing it manually, you can use this script to do so. Go to the console of your HOST machine, and execute:

```
HOST$ simctl iptunnel exec deltun
```

   This will delete the tunnel in both sides R1 and R2.
3. Change the MTU of SimNet2 to the value 996 bytes. So, get the consoles of R2 and RC and use the ifconfig command to do so. For instance, in R2 execute:

```
R2$ ifconfig eth2 mtu 996
```

```
RC#: ifconfig eth2 mtu 996
```

4. Reestablish again the tunnel using the following script:

```
HOST$ simctl iptunnel exec addtun_nopmtu
```

   a. Which is now the MTU in both sides of the tunnel?
   b. Why this change in the MTU value?

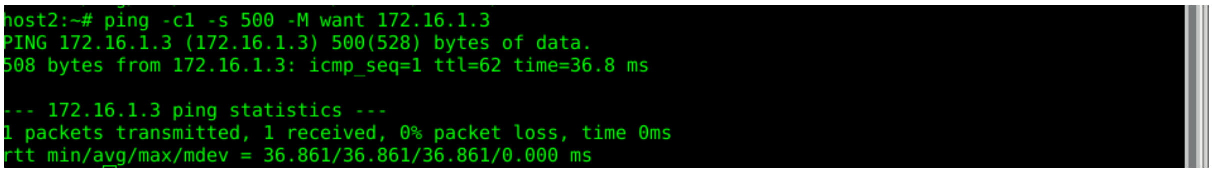
Fragmentation fields and options

We pretend to see how the fragmentation fields of the Outer header depend on the Inner header and also on the configuration of the tunnel.

1. Put all wiresharks listening traffic in all SimNet interfaces.
2. Send the following ping from host2 to host3:

```
host2$ ping -c1 -s 500 -M want 172.16.1.3
```

   Consult the man page of ping to know what is the meaning of the -s and -M options.



```
host2:~# ping -c1 -s 500 -M want 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 500(528) bytes of data.
508 bytes from 172.16.1.3: icmp_seq=1 ttl=62 time=36.8 ms

--- 172.16.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 36.861/36.861/36.861/0.000 ms
```

   a. How many ICMP packets can you see? What kind of packets?
   b. Which is the size of these packets? Describe the headers of these packets.
   c. Have these packets been fragmented? Why?

TCGI slides 12
https://drive.google.com/drive/folders/1ugbAayGfGTtjjVX0rLdREecVnr-qNXcN?usp=sharing

Podem veure 2 paquets ICMP (request i reply). **L=542B a SimNet0 i SimNet3-->** 500B (dades) + 20(IP Inner Header) + 14(Header Eth) + 8 (Header ICMP) **L=562B a SimNet1 i SmiNet2** --> 500B (dades) + 20B (H.IP Inner) + 20B (**IP Outer Header**) 14B + 8B

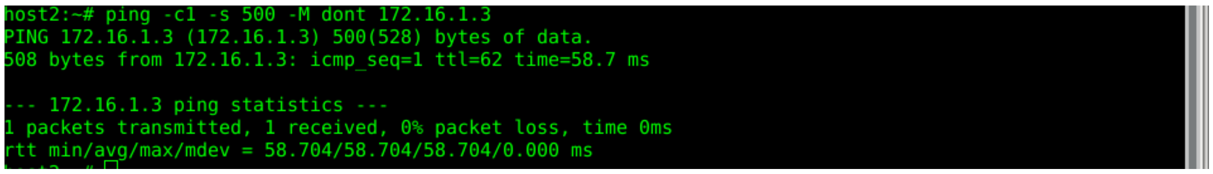
d. Which is the value of the "don't fragment" (DF) flag in the ICMP packets in SimNet0? Set

e. And in SimNet1, which is the value of DF in the Outer and Inner headers? Set

After this brief test, you should have noticed that the -s option is used to set the number of bytes in the "data" section of ICMP. The -M option is used to manage fragmentation in the origin host. The -M want option is intended to "do PMTU discovery, fragment locally when packet size is large". That is, the DF flag is set by default, but when there is the necessity of fragmentation, the origin can do it.
No hi ha fragmentació pq el tamany és inferior al "Path MTU".

3. Now execute:

```
host2$ ping -c1 -s 500 -M dont 172.16.1.3
```

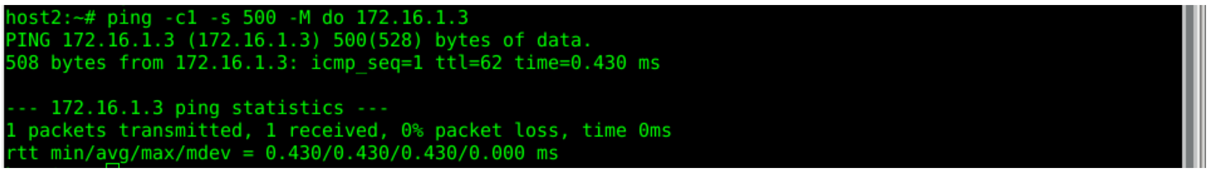


```
host2:~# ping -c1 -s 500 -M dont 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 500(528) bytes of data.
508 bytes from 172.16.1.3: icmp_seq=1 ttl=62 time=58.7 ms

--- 172.16.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 58.704/58.704/58.704/0.000 ms
```

a. What is the difference compared to the previous test? As you can see, the dont option ("do not set DF flag") allows fragmentation in path. Practically, there's no difference. But now the packets could be fragmented in its way if necessary.

4. Now execute:

```
host2$ ping -c1 -s 500 -M do 172.16.1.3
```



```
host2:~# ping -c1 -s 500 -M do 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 500(528) bytes of data.
508 bytes from 172.16.1.3: icmp_seq=1 ttl=62 time=0.430 ms

--- 172.16.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.430/0.430/0.430/0.000 ms
```

a. Can you see any difference in wireshark compared to the first case? Is there any difference at all? The do option "prohibit fragmentation, even local one". In this previous case you will see no difference, but later in other tests there can be some differences.

Now that you understand how encapsulation is performed, calculate which is the size necessary to make the encapsulated packet have the maximum sized alowed by SimNet2. Send a ping of this size and verify that there is no fragmentation.
996-14-20-20=942

```
host2#: ping -c1 -s 948 -M dont 172.16.1.3
```

The -M want option

We start the test with the -M want option because it is the default option used by ping. Now you are going to cause packets to be fragmented, and to see how the encapsulator (R1) behaves in relation to the management of messages ICMP Datagram Too Big (Type 3, Code 4, also known as Fragmentation Needed), used in the PMTU Discovery mechanish.
It is important to know that these messages are able to modify the status of the dynamic routing table (kernel routing cache). Sometimes during the lab session you will be asked to delete this kernel routing cache, so the starting point of the test will be always the same. This can be done by manually executing the following command in all the involved hosts and routers:

```
$ ip route flush cache
```

TCGI slides 12
https://drive.google.com/drive/folders/1ugbAayGfGTtjjVX0rLdREecVnr-qNXcN?usp=sharing

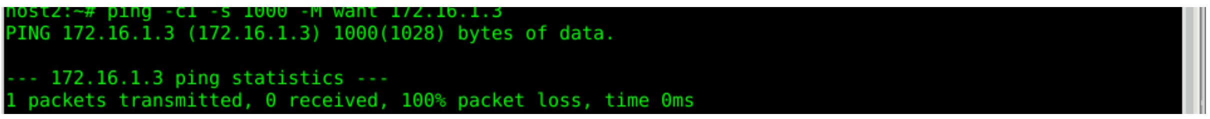
However, we recommend to execute the following script, that automatically deletes this cache in all machines involved:

```
HOST$ simctl iptunnel exec flushcache
```

So, you are going to increment the size of packets to cause fragmentation:

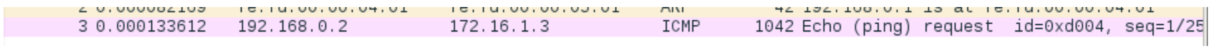
1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Execute the following command in host2:

```
host2$ ping –c1 –s 1000 –M want 172.16.1.3
```



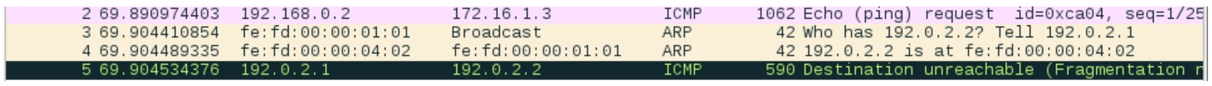
```
host2:~# ping -c1 -s 1000 -M want 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1000(1028) bytes of data.

--- 172.16.1.3 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

SimNet0:



```
      2 0.000082109   fe:fd:00:00:04:01   fe:fd:00:00:03:01   ARP      42 192.168.0.1 is at fe:fd:00:00:04:01
      3 0.000133612   192.168.0.2         172.16.1.3          ICMP   1042 Echo (ping) request  id=0xd004, seq=1/25
```

SimNet1:



```
      2 69.890974403  192.168.0.2         172.16.1.3          ICMP   1062 Echo (ping) request  id=0xca04, seq=1/25
      3 69.904410854  fe:fd:00:00:01:01   Broadcast           ARP      42 Who has 192.0.2.2? Tell 192.0.2.1
      4 69.904489335  fe:fd:00:00:04:02   fe:fd:00:00:01:01   ARP      42 192.0.2.2 is at fe:fd:00:00:04:02
      5 69.904534376  192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
```

a. Is the ping working?
b. How many ICMP packets can you see in SimNet0? What kind of packets? Is set the DF flag in the IP header of the ICMP echo-request?
c. And in SimNet1, what packets can you see? Is the DF flag set in the IP headers (inner and outer)?
   As you can see, there is an error produced by the MTU by means of a message ICMP fragmentation needed. This error apears due to the DF flaf is Set in the Outer IP header of the ICMP echo-request message.
d. What device is the origin of the ICMP fragmentation-needed message? And the destination? Which is the maximum MTU notified in this message?
e. In your opinion, what entity should be the proper destination of this message?
f. According to RFC2003 (see Section 4.1), is R1 acting as a relay of the message and sending it to host2?

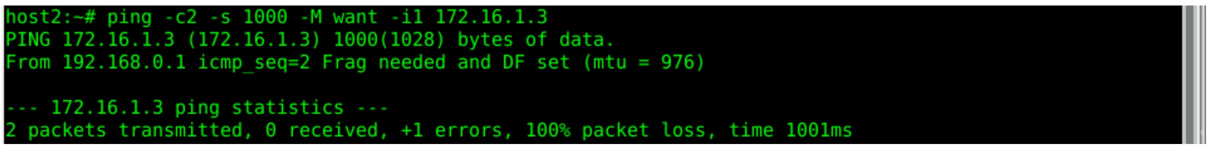
El ping no s'envia, podem veure un paquet ICMP echo-request amb DF=Set a SimNet0. I a SimNet1 veiem 2 paquets ICMP (Request amb DF=Set tant en inner com outer i DU (Frag. Need.)) El dispositiu que envia aquest missatge és el RC i la destinació és l'entrada eth2 del R1. El host2 hauria de ser el destí adequat del missatge anterior. No s'està fent cap ICMP Relaying, però l'encapsulador fa Soft State.

As you can see, in the case of fragmentation-needed messages, the encapsulator is not performing ICMP Relaying.
Instead, the encapsulator uses soft state to inform the origin host about the error. To show this, we are going to send more than one message to use the kernel routing cache:

1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Execute the following command in host2:

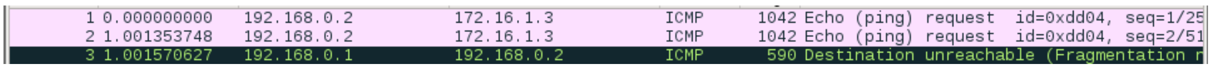
```
host2$ ping –c2 –s 1000 –M want –i1 172.16.1.3
```

With this command we are sending two ICMP echo-request messages with 1000 Bytes of data and separated by an interval (-i option) of 1 second.



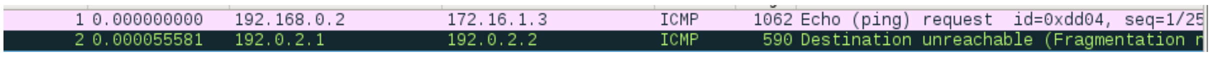
```
host2:~# ping -c2 -s 1000 -M want -i1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1000(1028) bytes of data.
From 192.168.0.1 icmp_seq=2 Frag needed and DF set (mtu = 976)

--- 172.16.1.3 ping statistics ---
2 packets transmitted, 0 received, +1 errors, 100% packet loss, time 1001ms
```

SimNet0:



```
1 0.000000000   192.168.0.2   172.16.1.3   ICMP   1042 Echo (ping) request  id=0xdd04, seq=1/25
2 1.001353748   192.168.0.2   172.16.1.3   ICMP   1042 Echo (ping) request  id=0xdd04, seq=2/51
3 1.001570627   192.168.0.1   192.168.0.2  ICMP    590 Destination unreachable (Fragmentation n
```

SimNet1:



```
1 0.000000000   192.168.0.2   172.16.1.3   ICMP   1062 Echo (ping) request  id=0xdd04, seq=1/25
2 0.000055581   192.0.2.1     192.0.2.2    ICMP    590 Destination unreachable (Fragmentation n
```

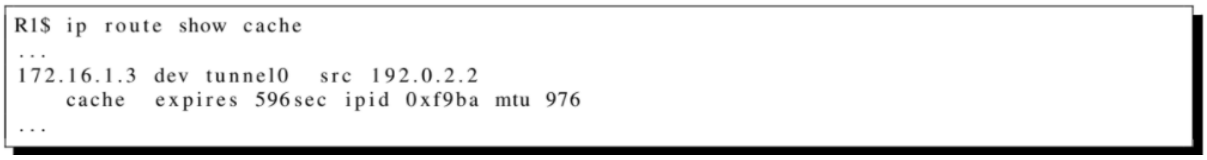
a. Is the ping working? How many ICMP messages can you see in SimNet0? Is the DF flag set in the IP header of these messages?
b. In SimNet1, how many ICMP echo-requests can you see? What about the DF flag?
c. Can you see an ICMP fragmentation-needed message in SimNet0? Which is the origin and destination of this message? Which is the maximum MTU notified in this message?
d. Do you think that R1 maintains a soft state of the MTU of the tunnel?

El ping no arriba al destí. Podem veure tres missatges ICMP en SimNet0, dos echo-request (Tant un com l'altre amb DF=Set) i un missatge de DestinationUnreachable, fragmentation needed (Amb DF = Not Set)

En SimNet1 podem veure dos missatges ICMP, un echo-request (Amb DF=Set tant en capa inner com outer) i un missatge DU-FN (Amb DF = Set tant en capa inner com outer).

Podem veure un missatge de fragmentation-needed a SimNet0 que té @IPsrc= 192.168.0.1 i @IPdst=192.168.0.2 (De R1 a host2). La MTU que notifica és = 976B. L'R1 manté el soft state de la MTU del túnel ja que notifica a RC de que es necessita fragmentar.

This soft state can be seen by executing the following command in R1:

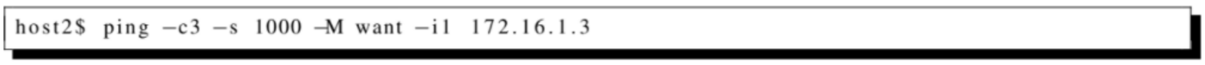


```
R1$ ip route show cache
...
172.16.1.3 dev tunnel0  src 192.0.2.2
    cache  expires 596sec ipid 0xf9ba mtu 976
...
```

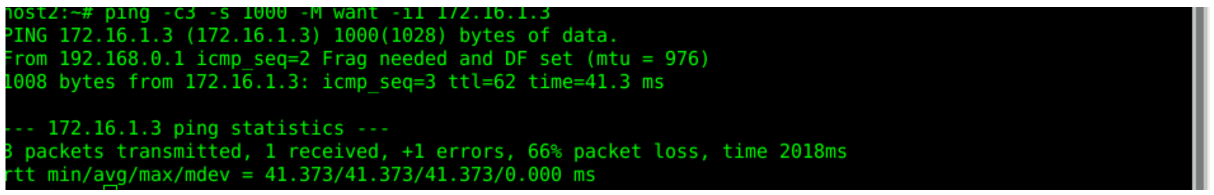
Notice that the information contained in this routing cache is ephemeral (10 min). If you lasted more than this time to execute the previous command maybe you will find the cache empty, so you have to repeat the experiment.

Now, we will send three ICMP echo-request messages:

1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Execute the following command in host2:

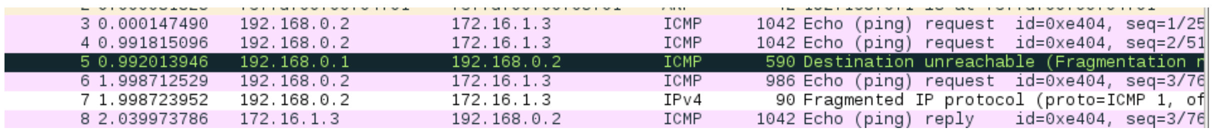


```
host2$ ping -c3 -s 1000 -M want -i1 172.16.1.3
```



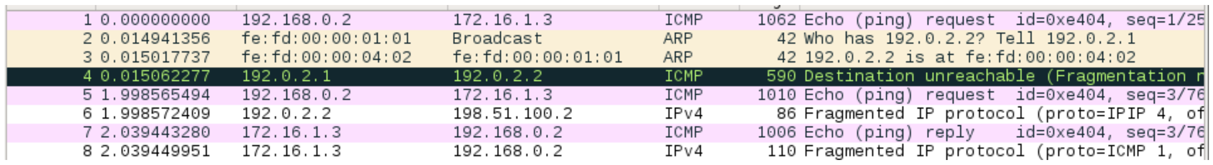
```
host2:~# ping -c3 -s 1000 -M want -i1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1000(1028) bytes of data.
From 192.168.0.1 icmp_seq=2 Frag needed and DF set (mtu = 976)
1008 bytes from 172.16.1.3: icmp_seq=3 ttl=62 time=41.3 ms

--- 172.16.1.3 ping statistics ---
3 packets transmitted, 1 received, +1 errors, 66% packet loss, time 2018ms
rtt min/avg/max/mdev = 41.373/41.373/41.373/0.000 ms
```

SimNet0:



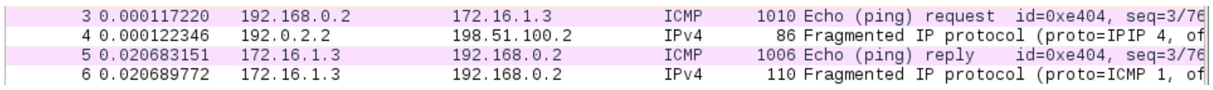
```
3 0.000147490  192.168.0.2   172.16.1.3    ICMP   1042 Echo (ping) request  id=0xe404, seq=1/25
4 0.991815096  192.168.0.2   172.16.1.3    ICMP   1042 Echo (ping) request  id=0xe404, seq=2/51
5 0.992013946  192.168.0.1   192.168.0.2   ICMP    590 Destination unreachable (Fragmentation n
6 1.998712529  192.168.0.2   172.16.1.3    ICMP    986 Echo (ping) request  id=0xe404, seq=3/76
7 1.998723952  192.168.0.2   172.16.1.3    IPv4     90 Fragmented IP protocol (proto=ICMP 1, of
8 2.039973786  172.16.1.3    192.168.0.2   ICMP   1042 Echo (ping) reply    id=0xe404, seq=3/76
```

SimNet1:



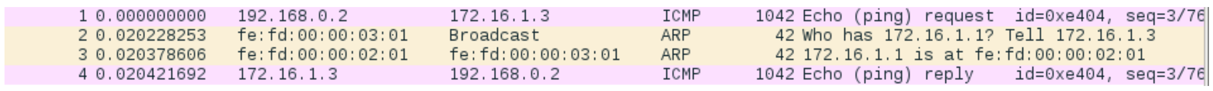
```
1 0.000000000  192.168.0.2       172.16.1.3        ICMP   1062 Echo (ping) request  id=0xe404, seq=1/25
2 0.014941356  fe:fd:00:00:01:01 Broadcast         ARP      42 Who has 192.0.2.2? Tell 192.0.2.1
3 0.015017737  fe:fd:00:00:04:02 fe:fd:00:00:01:01 ARP      42 192.0.2.2 is at fe:fd:00:00:04:02
4 0.015062277  192.0.2.1         192.0.2.2         ICMP    590 Destination unreachable (Fragmentation n
5 1.998565494  192.168.0.2       172.16.1.3        ICMP   1010 Echo (ping) request  id=0xe404, seq=3/76
6 1.998572409  192.0.2.2         198.51.100.2      IPv4     86 Fragmented IP protocol (proto=IPIP 4, of
7 2.039443280  172.16.1.3        192.168.0.2       ICMP   1006 Echo (ping) reply    id=0xe404, seq=3/76
8 2.039449951  172.16.1.3        192.168.0.2       IPv4    110 Fragmented IP protocol (proto=ICMP 1, of
```

SimNet2:



```
3 0.000117220  192.168.0.2   172.16.1.3    ICMP   1010 Echo (ping) request  id=0xe404, seq=3/76
4 0.000122346  192.0.2.2     198.51.100.2  IPv4     86 Fragmented IP protocol (proto=IPIP 4, of
5 0.020683151  172.16.1.3    192.168.0.2   ICMP   1006 Echo (ping) reply    id=0xe404, seq=3/76
6 0.020689772  172.16.1.3    192.168.0.2   IPv4    110 Fragmented IP protocol (proto=ICMP 1, of
```

SimNet3:



```
1 0.000000000  192.168.0.2       172.16.1.3        ICMP   1042 Echo (ping) request  id=0xe404, seq=3/76
2 0.020228253  fe:fd:00:00:03:01 Broadcast         ARP      42 Who has 172.16.1.1? Tell 172.16.1.3
3 0.020378606  fe:fd:00:00:02:01 fe:fd:00:00:03:01 ARP      42 172.16.1.1 is at fe:fd:00:00:02:01
4 0.020421692  172.16.1.3        192.168.0.2       ICMP   1042 Echo (ping) reply    id=0xe404, seq=3/76
```

Now let us analyze this test in detail:

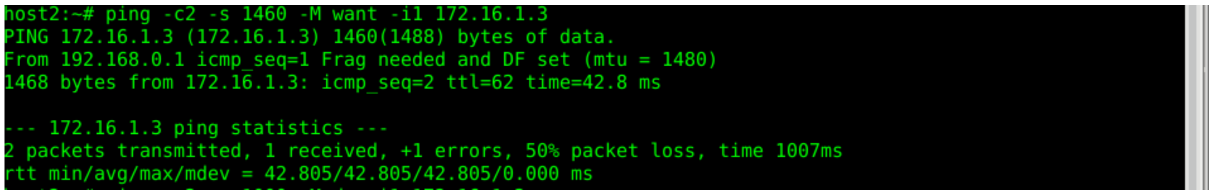
a. Is the ping working? How do you know that?
b. In SimNet0, have a look at the DF flag in all the ICMP echo-request messages. What can you deduce? Is there any difference in the third ICMP Echo request message?
c. In SimNet0, who is performing the fragmentation of the third ICMP echo-request? Which are the sizes of the packets that compose this third ICMP echo-request? Take notes of the existing headers.
d. In SimNet1, how many fragmented packets can you see that correspond to this third ICMP echo-request?
   What about the DF flag in the Inner and Outer IP headers?+
e. Describe in detail the sizes and headers of these packets. Is fragmentation performed in the same way than SimNet0? Why the difference? Notice that the Fragment Offset field in the IP header is meassured in units of 8 bytes.
f. In SimNet3, how many ICMP echo-requests can you see? Describe sizes and headers.
g. In SimNet3, how many ICMP echo-reply messages can you see? Describe sizes and headers.
h. In SimNet2, how many fragmented packets can can you see that correspond to this third ICMP echo-reply?
   Describe in detail sizes of these packets and headers. Is fragmentation performed in the same way than for the ICMP Request message? Describe in detail sizes of these packets and headers.

Ara sí que el ping funciona, ho veiem ja que rebem echo-reply a SimNet0. A SN0: Primer echo-request DF=Set, segon echo-request DF=Set, tercer echo-request DF=Not Set i MF=Set. Deduïm que el tercer paquet s'ha pogut fragmentar. La fragmentació la fa el host2, l'emissor d'aquest paquet. La mida dels paquets és: El primer fragment de 986B = 944B (dades) +20B(IP)+8B(ICMP)+14B(eth) El fragment de 90B = 56B(Data)+20B(IP)+14B(eth)

Finally, we will send the latest ping of this series:

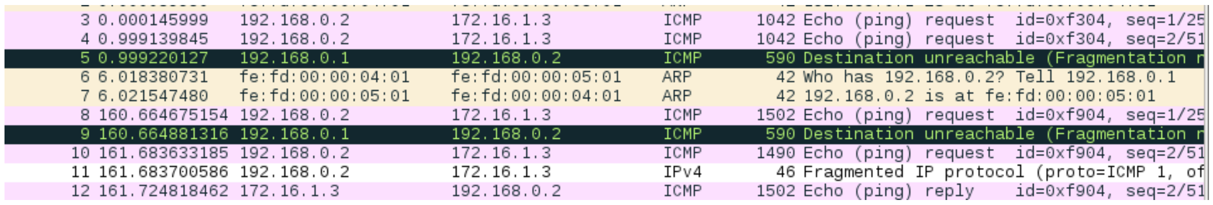
1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Execute the following command in host2:

```
host2$ ping -c2 -s 1460 -M want -i1 172.16.1.3
```



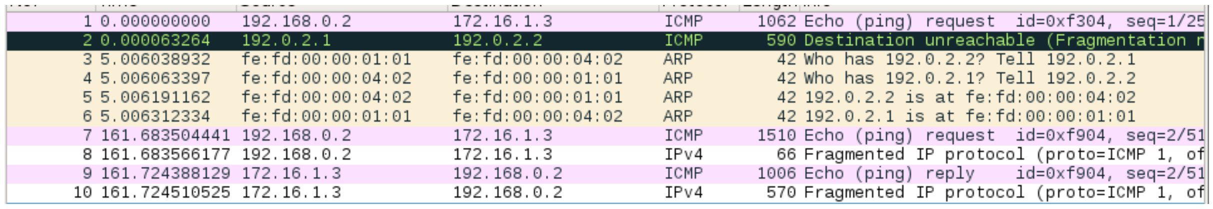
```
host2:~# ping -c2 -s 1460 -M want -i1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1460(1488) bytes of data.
From 192.168.0.1 icmp_seq=1 Frag needed and DF set (mtu = 1480)
1468 bytes from 172.16.1.3: icmp_seq=2 ttl=62 time=42.8 ms

--- 172.16.1.3 ping statistics ---
2 packets transmitted, 1 received, +1 errors, 50% packet loss, time 1007ms
rtt min/avg/max/mdev = 42.805/42.805/42.805/0.000 ms
```

SImNet0:



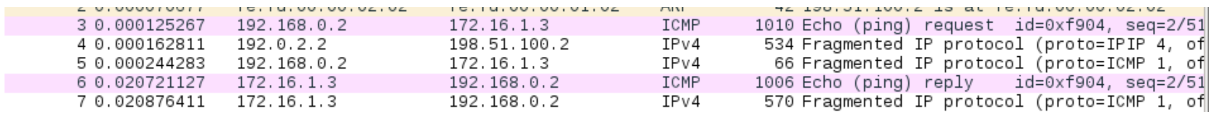
```
 3 0.000145999   192.168.0.2        172.16.1.3         ICMP    1042 Echo (ping) request  id=0xf304, seq=1/25
 4 0.999139845   192.168.0.2        172.16.1.3         ICMP    1042 Echo (ping) request  id=0xf304, seq=2/51
 5 0.999220127   192.168.0.1        192.168.0.2        ICMP     590 Destination unreachable (Fragmentation n
 6 6.018380731   fe:fd:00:00:04:01  fe:fd:00:00:05:01  ARP       42 Who has 192.168.0.2? Tell 192.168.0.1
 7 6.021547480   fe:fd:00:00:05:01  fe:fd:00:00:04:01  ARP       42 192.168.0.2 is at fe:fd:00:00:05:01
 8 160.664675154 192.168.0.2        172.16.1.3         ICMP    1502 Echo (ping) request  id=0xf904, seq=1/25
 9 160.664881316 192.168.0.1        192.168.0.2        ICMP     590 Destination unreachable (Fragmentation n
10 161.683633185 192.168.0.2        172.16.1.3         ICMP    1490 Echo (ping) request  id=0xf904, seq=2/51
11 161.683700586 192.168.0.2        172.16.1.3         IPv4      46 Fragmented IP protocol (proto=ICMP 1, of
12 161.724818462 172.16.1.3         192.168.0.2        ICMP    1502 Echo (ping) reply    id=0xf904, seq=2/51
```

SImNet1:



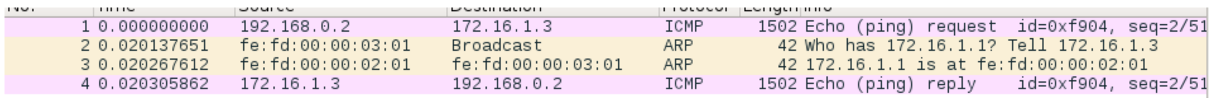
```
 1 0.000000000   192.168.0.2        172.16.1.3         ICMP    1062 Echo (ping) request  id=0xf304, seq=1/25
 2 0.000063264   192.0.2.1          192.0.2.2          ICMP     590 Destination unreachable (Fragmentation n
 3 5.006038932   fe:fd:00:00:01:01  fe:fd:00:00:04:02  ARP       42 Who has 192.0.2.2? Tell 192.0.2.1
 4 5.006063397   fe:fd:00:00:04:02  fe:fd:00:00:01:01  ARP       42 Who has 192.0.2.1? Tell 192.0.2.2
 5 5.006191162   fe:fd:00:00:04:02  fe:fd:00:00:01:01  ARP       42 192.0.2.2 is at fe:fd:00:00:04:02
 6 5.006312334   fe:fd:00:00:01:01  fe:fd:00:00:04:02  ARP       42 192.0.2.1 is at fe:fd:00:00:01:01
 7 161.683504441 192.168.0.2        172.16.1.3         ICMP    1510 Echo (ping) request  id=0xf904, seq=2/51
 8 161.683566177 192.168.0.2        172.16.1.3         IPv4      66 Fragmented IP protocol (proto=ICMP 1, of
 9 161.724388129 172.16.1.3         192.168.0.2        ICMP    1006 Echo (ping) reply    id=0xf904, seq=2/51
10 161.724510525 172.16.1.3         192.168.0.2        IPv4     570 Fragmented IP protocol (proto=ICMP 1, of
```

SimNet2:



```
3 0.000125267   192.168.0.2        172.16.1.3         ICMP    1010 Echo (ping) request  id=0xf904, seq=2/51
4 0.000162811   192.0.2.2          198.51.100.2       IPv4     534 Fragmented IP protocol (proto=IPIP 4, of
5 0.000244283   192.168.0.2        172.16.1.3         IPv4      66 Fragmented IP protocol (proto=ICMP 1, of
6 0.020721127   172.16.1.3         192.168.0.2        ICMP    1006 Echo (ping) reply    id=0xf904, seq=2/51
7 0.020876411   172.16.1.3         192.168.0.2        IPv4     570 Fragmented IP protocol (proto=ICMP 1, of
```

SimNet3:



```
1 0.000000000   192.168.0.2        172.16.1.3         ICMP    1502 Echo (ping) request  id=0xf904, seq=2/51
2 0.020137651   fe:fd:00:00:03:01  Broadcast          ARP       42 Who has 172.16.1.1? Tell 172.16.1.3
3 0.020267612   fe:fd:00:00:02:01  fe:fd:00:00:03:01  ARP       42 172.16.1.1 is at fe:fd:00:00:02:01
4 0.020305862   172.16.1.3         192.168.0.2        ICMP    1502 Echo (ping) reply    id=0xf904, seq=2/51
```

a. Is there any error message? Who is the sender of this error?
b. Which is the value of MTU that is reporting this error message? Is this value the real path MTU value? Why there are no more errors during the transmission? Have a look to the DF flag.

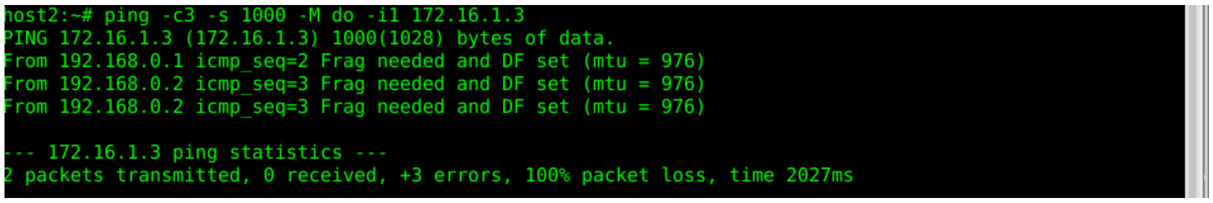
Es perd 1 paquet ICMP. A la SN0 rebem del RC un missatge d error (Fragm. needed) ja que el msg es de L=1502 bytes. Un cop el host 2 sap que d'ha de fragmentar la segona trama té el DF=Not set i MF=set.

The -M do option

Now, we are going to test the -M do option. Remember that the -M do option "prohibit fragmentation, even local one". Let us see if there is any difference with the previous -M want option.

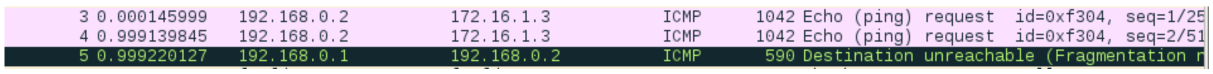
1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Execute the following command in host2:

```
host2$ ping -c3 -s 1000 -M do -i1 172.16.1.3
```



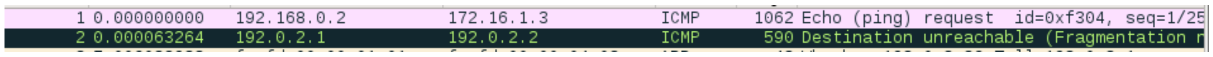
```
host2:~# ping -c3 -s 1000 -M do -i1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1000(1028) bytes of data.
From 192.168.0.1 icmp_seq=2 Frag needed and DF set (mtu = 976)
From 192.168.0.2 icmp_seq=3 Frag needed and DF set (mtu = 976)
From 192.168.0.2 icmp_seq=3 Frag needed and DF set (mtu = 976)

--- 172.16.1.3 ping statistics ---
2 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2027ms
```

SimNet0:



```
  3 0.000145999   192.168.0.2     172.16.1.3      ICMP    1042 Echo (ping) request  id=0xf304, seq=1/25
  4 0.999139845   192.168.0.2     172.16.1.3      ICMP    1042 Echo (ping) request  id=0xf304, seq=2/51
  5 0.999220127   192.168.0.1     192.168.0.2     ICMP     590 Destination unreachable (Fragmentation n
```

SimNet1:



```
  1 0.000000000   192.168.0.2     172.16.1.3      ICMP    1062 Echo (ping) request  id=0xf304, seq=1/25
  2 0.000063264   192.0.2.1       192.0.2.2       ICMP     590 Destination unreachable (Fragmentation n
```

a. Is the ping working? How many ICMP echo-request packets can you see in SimNet0? Is there any fragmented packet?
b. Can you see the difference with the previous -M want option?

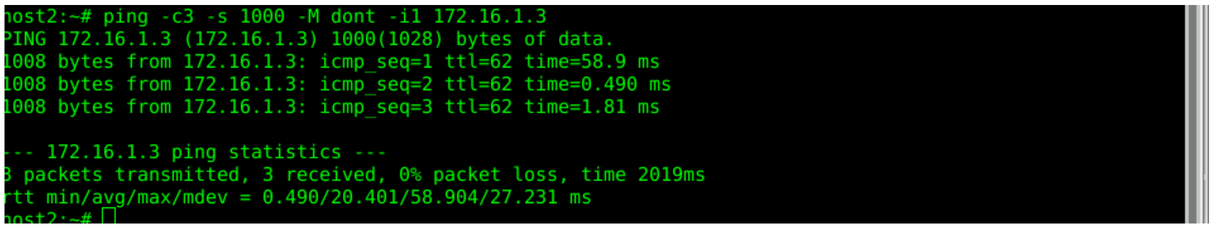
El ping no funciona. La diferencia es q ara no hi ha fragmentacio ni desde el host2. Per aixo no es transmet cap paquet ICMP.

The -M dont option

Now, we are going to test the -M dont option. Remember that the -M dont option ("do not set DF flag") allows fragmentation in path.

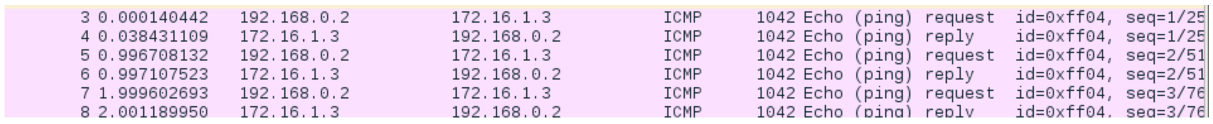
1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Execute the following command in host2:

```
host2$ ping -c3 -s 1000 -M dont -i1 172.16.1.3
```



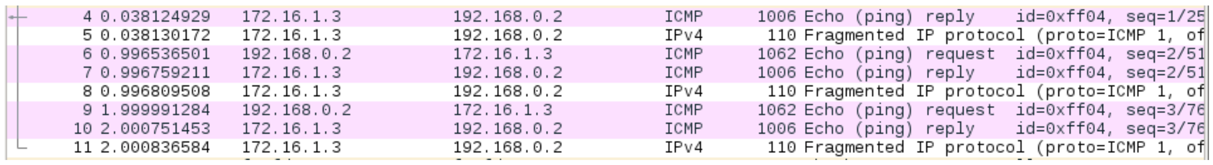
```
host2:~# ping -c3 -s 1000 -M dont -i1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1000(1028) bytes of data.
1008 bytes from 172.16.1.3: icmp_seq=1 ttl=62 time=58.9 ms
1008 bytes from 172.16.1.3: icmp_seq=2 ttl=62 time=0.490 ms
1008 bytes from 172.16.1.3: icmp_seq=3 ttl=62 time=1.81 ms

--- 172.16.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2019ms
rtt min/avg/max/mdev = 0.490/20.401/58.904/27.231 ms
host2:~#
```

SimNet0:



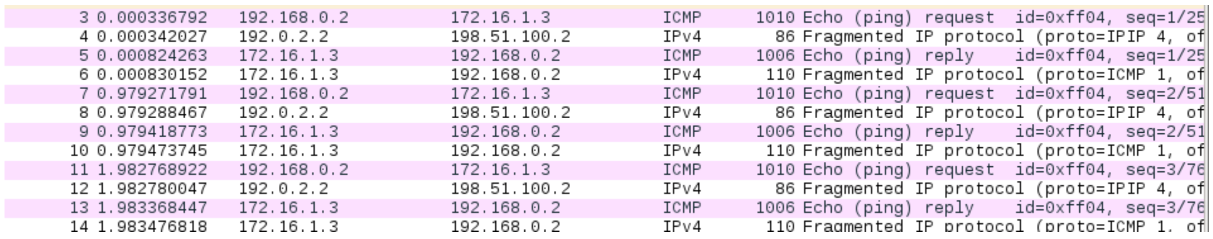
```
  3 0.000140442   192.168.0.2     172.16.1.3      ICMP    1042 Echo (ping) request  id=0xff04, seq=1/25
  4 0.038431109   172.16.1.3      192.168.0.2     ICMP    1042 Echo (ping) reply    id=0xff04, seq=1/25
  5 0.996708132   192.168.0.2     172.16.1.3      ICMP    1042 Echo (ping) request  id=0xff04, seq=2/51
  6 0.997107523   172.16.1.3      192.168.0.2     ICMP    1042 Echo (ping) reply    id=0xff04, seq=2/51
  7 1.999602693   192.168.0.2     172.16.1.3      ICMP    1042 Echo (ping) request  id=0xff04, seq=3/76
  8 2.001189950   172.16.1.3      192.168.0.2     ICMP    1042 Echo (ping) reply    id=0xff04, seq=3/76
```

SimNet1:



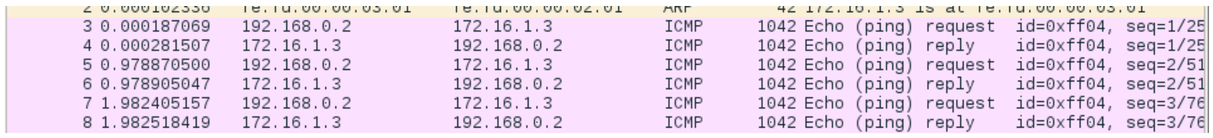
```
  4 0.038124929   172.16.1.3      192.168.0.2     ICMP    1006 Echo (ping) reply    id=0xff04, seq=1/25
  5 0.038130172   172.16.1.3      192.168.0.2     IPv4     110 Fragmented IP protocol (proto=ICMP 1, of
  6 0.996536501   192.168.0.2     172.16.1.3      ICMP    1062 Echo (ping) request  id=0xff04, seq=2/51
  7 0.996759211   172.16.1.3      192.168.0.2     ICMP    1006 Echo (ping) reply    id=0xff04, seq=2/51
  8 0.996809508   172.16.1.3      192.168.0.2     IPv4     110 Fragmented IP protocol (proto=ICMP 1, of
  9 1.999991284   192.168.0.2     172.16.1.3      ICMP    1062 Echo (ping) request  id=0xff04, seq=3/76
 10 2.000751453   172.16.1.3      192.168.0.2     ICMP    1006 Echo (ping) reply    id=0xff04, seq=3/76
 11 2.000836584   172.16.1.3      192.168.0.2     IPv4     110 Fragmented IP protocol (proto=ICMP 1, of
```

SimNet2:



```
  3 0.000336792   192.168.0.2     172.16.1.3      ICMP    1010 Echo (ping) request  id=0xff04, seq=1/25
  4 0.000342027   192.0.2.2       198.51.100.2    IPv4      86 Fragmented IP protocol (proto=IPIP 4, of
  5 0.000824263   172.16.1.3      192.168.0.2     ICMP    1006 Echo (ping) reply    id=0xff04, seq=1/25
  6 0.000830152   172.16.1.3      192.168.0.2     IPv4     110 Fragmented IP protocol (proto=ICMP 1, of
  7 0.979271791   192.168.0.2     172.16.1.3      ICMP    1010 Echo (ping) request  id=0xff04, seq=2/51
  8 0.979288467   192.0.2.2       198.51.100.2    IPv4      86 Fragmented IP protocol (proto=IPIP 4, of
  9 0.979418773   172.16.1.3      192.168.0.2     ICMP    1006 Echo (ping) reply    id=0xff04, seq=2/51
 10 0.979473745   172.16.1.3      192.168.0.2     IPv4     110 Fragmented IP protocol (proto=ICMP 1, of
 11 1.982768922   192.168.0.2     172.16.1.3      ICMP    1010 Echo (ping) request  id=0xff04, seq=3/76
 12 1.982780047   192.0.2.2       198.51.100.2    IPv4      86 Fragmented IP protocol (proto=IPIP 4, of
 13 1.983368447   172.16.1.3      192.168.0.2     ICMP    1006 Echo (ping) reply    id=0xff04, seq=3/76
 14 1.983476818   172.16.1.3      192.168.0.2     IPv4     110 Fragmented IP protocol (proto=ICMP 1, of
```

SimNet3:



```
  2 0.000102336   fe:fd:00:00:03:01 fe:fd:00:00:02:01 ARP   42 172.16.1.3 is at fe:fd:00:00:03:01
  3 0.000187069   192.168.0.2     172.16.1.3      ICMP    1042 Echo (ping) request  id=0xff04, seq=1/25
  4 0.000281507   172.16.1.3      192.168.0.2     ICMP    1042 Echo (ping) reply    id=0xff04, seq=1/25
  5 0.978870500   192.168.0.2     172.16.1.3      ICMP    1042 Echo (ping) request  id=0xff04, seq=2/51
  6 0.978905047   172.16.1.3      192.168.0.2     ICMP    1042 Echo (ping) reply    id=0xff04, seq=2/51
  7 1.982405157   192.168.0.2     172.16.1.3      ICMP    1042 Echo (ping) request  id=0xff04, seq=3/76
  8 1.982518419   172.16.1.3      192.168.0.2     ICMP    1042 Echo (ping) reply    id=0xff04, seq=3/76
```

a. Is the ping working? Is there any error message? Why? Have a look to the DF flags.
b. According to all the previous results, which is the best option to use? Think about pros and cons of fragmentation in path.

The ping works.The best option if you could send some pings is the -M want option because the frames are fragmented locally if needed. The other option could be to use the -M dont in order to fragment all frames but in this case we will have more paquets to transmit and a lot of traffic between R1 and R2 that is not needed if we only have to transmit one time the frame succesfully.

Testing tunnels: pmtudisc

All previous tests were performed using the nopmtudisc option when creating the tunnel. Let us change this option to see if there is any difference.

1. Delete the previously configured tunnel:

```
HOST$ simctl iptunnel exec deltun
```

2. Reestablish again the tunnel but now with the pmtudisc option enabled:

```
HOST$ simctl iptunnel exec addtun_pmtu
```

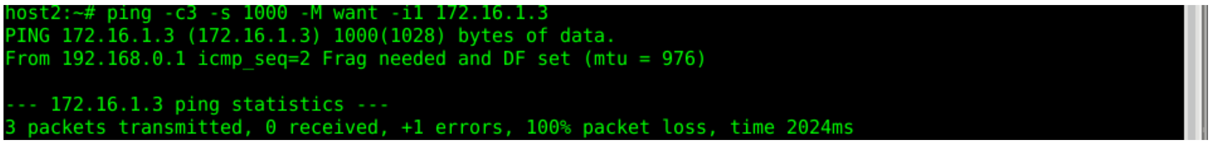
Now, verify the configuration in both edge routers with this command:

```
$ ip tunnel show
```

Also, verify that the MTU in both sides of the tunnel is the proper one. Now, do the following test:

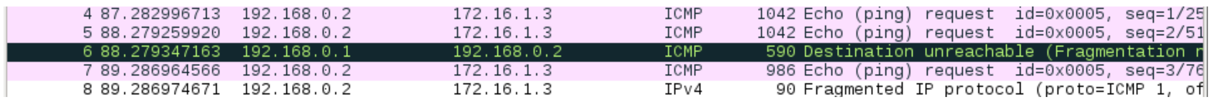
1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Execute the following command in host2:

```
host2$ ping –c3 –s 1000 –M want –i1 172.16.1.3
```



```
host2:~# ping -c3 -s 1000 -M want -i1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1000(1028) bytes of data.
From 192.168.0.1 icmp_seq=2 Frag needed and DF set (mtu = 976)

--- 172.16.1.3 ping statistics ---
3 packets transmitted, 0 received, +1 errors, 100% packet loss, time 2024ms
```

SimNet0:



```
      4 87.282996713  192.168.0.2        172.16.1.3         ICMP     1042 Echo (ping) request  id=0x0005, seq=1/25
      5 88.279259920  192.168.0.2        172.16.1.3         ICMP     1042 Echo (ping) request  id=0x0005, seq=2/51
      6 88.279347163  192.168.0.1        192.168.0.2        ICMP      590 Destination unreachable (Fragmentation n
      7 89.286964566  192.168.0.2        172.16.1.3         ICMP      986 Echo (ping) request  id=0x0005, seq=3/76
      8 89.286974671  192.168.0.2        172.16.1.3         IPv4       90 Fragmented IP protocol (proto=ICMP 1, of
```

SimNet1:

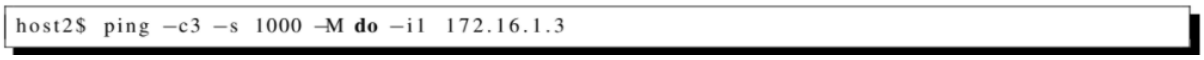
a. Is the ping working? How many ICMP echo-request can you see in SimNet0? Is there any error message? In which network?
b. Try to explain the behavior of the encapsulator.

A SN0 podem veure 3 ICMP echo-request, dos de normal i l'últim fragmentat desprès de rebre un DU(FN). Aquest missatge d'error tmb el veiem a SN1, ja que és el RC qui se n'adona de la necessitat.
L'encapsulador fa la mateixa feina de col·locar una capçalera IP. Però al seleccionar l'opció pmtudisc, que permet descobrir la MTU real del camí, aquest rep un DU(FN) del RC i llavors li envia aquesta notificació al host2. Aquest, amb l'avís, fragmenta el paquet, però com que l'encapsulador (R1) no accepta fragmentació (DF=Set), es perden els paquets.

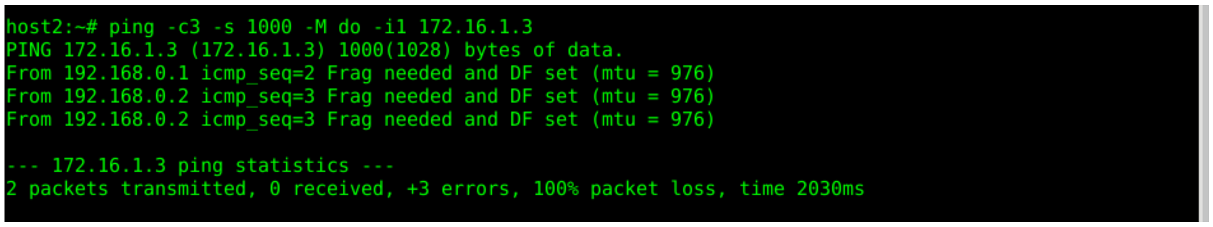
Now, let us test with the -M do option:

1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.

3. Execute the following command in host2:

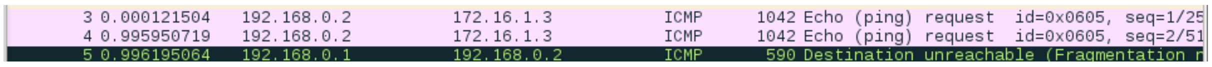


```
host2$ ping -c3 -s 1000 -M do -i1 172.16.1.3
```



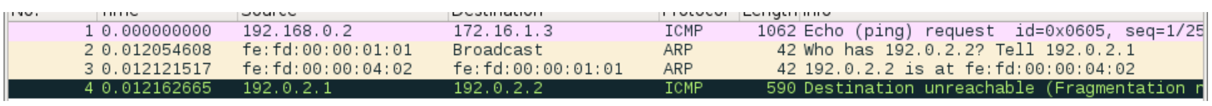
```
host2:~# ping -c3 -s 1000 -M do -i1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1000(1028) bytes of data.
From 192.168.0.1 icmp_seq=2 Frag needed and DF set (mtu = 976)
From 192.168.0.2 icmp_seq=3 Frag needed and DF set (mtu = 976)
From 192.168.0.2 icmp_seq=3 Frag needed and DF set (mtu = 976)

--- 172.16.1.3 ping statistics ---
2 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2030ms
```

SImNet0:



```
3 0.000121504   192.168.0.2   172.16.1.3   ICMP   1042 Echo (ping) request  id=0x0605, seq=1/25
4 0.995950719   192.168.0.2   172.16.1.3   ICMP   1042 Echo (ping) request  id=0x0605, seq=2/51
5 0.996195064   192.168.0.1   192.168.0.2  ICMP    590 Destination unreachable (Fragmentation n
```

SimNet1:



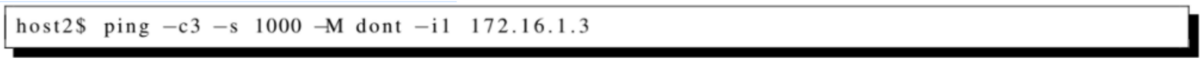
```
1 0.000000000   192.168.0.2         172.16.1.3          ICMP   1062 Echo (ping) request  id=0x0605, seq=1/25
2 0.012054608   fe:fd:00:00:01:01   Broadcast           ARP      42 Who has 192.0.2.2? Tell 192.0.2.1
3 0.012121517   fe:fd:00:00:04:02   fe:fd:00:00:01:01   ARP      42 192.0.2.2 is at fe:fd:00:00:04:02
4 0.012162665   192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
```

a. What is the difference regarding the previous test?

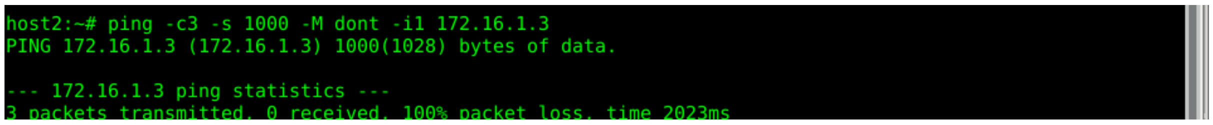
La diferència amb el cas anterior és que quan el host2 rep la notificació de fragmentació, aquest a causa del -M do no ho pot fer, per tant, ni l'envia.

Finally, let us test the latest -M dont option:

1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Execute the following command in host2:

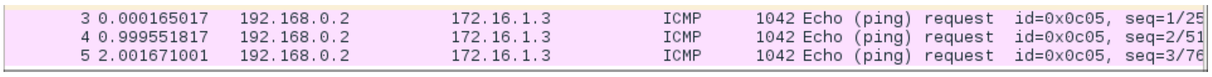


```
host2$ ping -c3 -s 1000 -M dont -i1 172.16.1.3
```



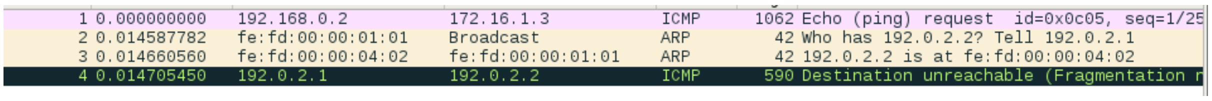
```
host2:~# ping -c3 -s 1000 -M dont -i1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 1000(1028) bytes of data.

--- 172.16.1.3 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2023ms
```

SimNet0:



```
3 0.000165017   192.168.0.2   172.16.1.3   ICMP   1042 Echo (ping) request  id=0x0c05, seq=1/25
4 0.999551817   192.168.0.2   172.16.1.3   ICMP   1042 Echo (ping) request  id=0x0c05, seq=2/51
5 2.001671001   192.168.0.2   172.16.1.3   ICMP   1042 Echo (ping) request  id=0x0c05, seq=3/76
```

SimNet1:



```
1 0.000000000   192.168.0.2         172.16.1.3          ICMP   1062 Echo (ping) request  id=0x0c05, seq=1/25
2 0.014587782   fe:fd:00:00:01:01   Broadcast           ARP      42 Who has 192.0.2.2? Tell 192.0.2.1
3 0.014660560   fe:fd:00:00:04:02   fe:fd:00:00:01:01   ARP      42 192.0.2.2 is at fe:fd:00:00:04:02
4 0.014705450   192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
```

a. Is the ping working? How many ICMP echo-request can you see in SimNet0? Is the DF Set in the packets in SimNet0?
b. In Simnet1, can you see ICMP echo-request packets? Which one, the first, second or third?
c. Is there any error message in SimNet1? Why this error? Have a look to the DF flag in inner and outer header echo-request.
d. Why the encapsulator is not sending the rest of pings towards the tunnel? Consult the kernel routing cache.

El ping no funciona. Es veuen 3 ICMP echo-request a SN0 amb DF=Not set. A SN1 només podem veure el primer, a més, també hi ha un DU(FN). Al request, a la capçalera Inner el DF=Not Set, però a la Outer el DF=Set. El kernel routing caché esta buit.

Testing tunnels: TCP and MSS

Now you are going to test how the TCP protocol behaves when the connection goes through a tunnel. First, reestablish the tunnel with the option nomptudisc. Leave the MTU=996 in SimNet2.

```
HOST$ simctl iptunnel exec deltun
HOST$ simctl iptunnel exec addtun_nopmtu
```

Netcat and TCP

First, we will start doing a correct transmission (without filtering rules) using netcat:

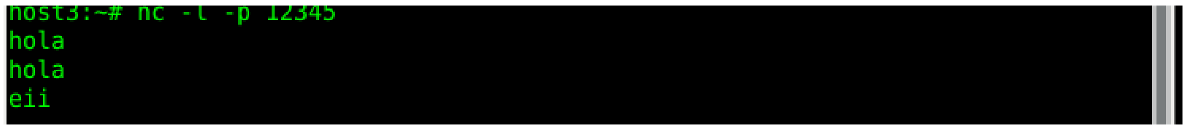
1. Put all wiresharks listening traffic in all SimNet interfaces.
2. Start a netcat server in host3 listening in port 12345 (the -l option means listen, that is, this host is behaving as server):

```
host3$ nc -l -p 12345
```

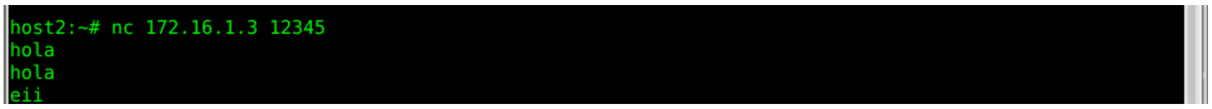
3. Start the netcat client in host2.

```
host2$ nc 172.16.1.3 12345
```

Netcat is used to establish a TCP connection between host2 and host3. As you can see, the tool behaves like a chat, and if you type something on the keyboard of one host, it appears in the other side. If you want to close netcat, you can type Ctrl+C, and the TCP connection will be closed. Have a look to all wiresharks and answer:

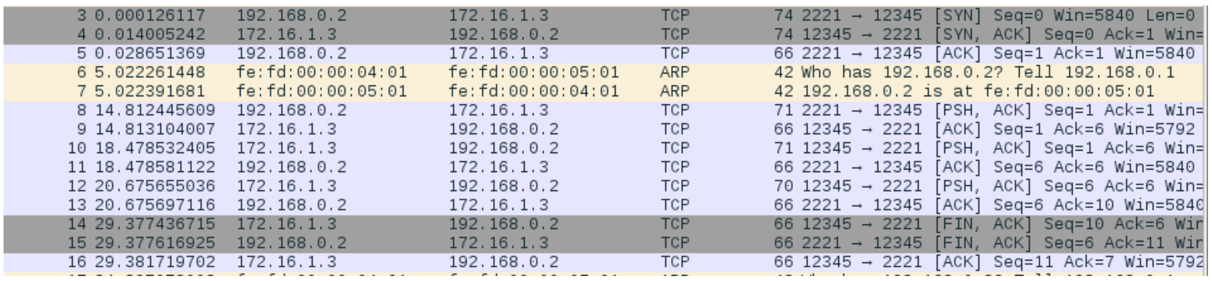


```
host3:~# nc -l -p 12345
hola
hola
eii
```



```
host2:~# nc 172.16.1.3 12345
hola
hola
eii
```

SimNet0:



```
 3 0.000126117   192.168.0.2        172.16.1.3         TCP   74 2221 → 12345 [SYN] Seq=0 Win=5840 Len=0
 4 0.014005242   172.16.1.3         192.168.0.2        TCP   74 12345 → 2221 [SYN, ACK] Seq=0 Ack=1 Win=
 5 0.028651369   192.168.0.2        172.16.1.3         TCP   66 2221 → 12345 [ACK] Seq=1 Ack=1 Win=5840
 6 5.022261448   fe:fd:00:00:04:01  fe:fd:00:00:05:01  ARP   42 Who has 192.168.0.2? Tell 192.168.0.1
 7 5.022391681   fe:fd:00:00:05:01  fe:fd:00:00:04:01  ARP   42 192.168.0.2 is at fe:fd:00:00:05:01
 8 14.812445609  192.168.0.2        172.16.1.3         TCP   71 2221 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=
 9 14.813104007  172.16.1.3         192.168.0.2        TCP   66 12345 → 2221 [ACK] Seq=1 Ack=6 Win=5792
10 18.478532405  172.16.1.3         192.168.0.2        TCP   71 12345 → 2221 [PSH, ACK] Seq=1 Ack=6 Win=
11 18.478581122  192.168.0.2        172.16.1.3         TCP   66 2221 → 12345 [ACK] Seq=6 Ack=6 Win=5840
12 20.675655036  172.16.1.3         192.168.0.2        TCP   70 12345 → 2221 [PSH, ACK] Seq=6 Ack=6 Win=
13 20.675697116  192.168.0.2        172.16.1.3         TCP   66 2221 → 12345 [ACK] Seq=6 Ack=10 Win=5840
14 29.377436715  172.16.1.3         192.168.0.2        TCP   66 12345 → 2221 [FIN, ACK] Seq=10 Ack=6 Wir
15 29.377616925  192.168.0.2        172.16.1.3         TCP   66 2221 → 12345 [FIN, ACK] Seq=6 Ack=11 Wir
16 29.381719702  172.16.1.3         192.168.0.2        TCP   66 12345 → 2221 [ACK] Seq=11 Ack=7 Win=5792
```

a. Can you see the classical 3-way handshake of TCP?
b. Which is the value of the MSS (Maximum Segment Size) advertised in this handshake?
c. What is the meaning of this MSS? Can have the client a MSS different from the server?

Podem veure el handshake TCP (client→ server) / (SYN→ SYN,ACK → ACK).
MSS= 1500B (MTUmax) - 20B (Hip) - 20B (Htcp) = 1460B. El servidor i el client poden tenir diferents MSS, en aquest cas s'utilitza la necessaria per a la comunicació: si envia el client utilitzarem la del servidor i viceversa.

Netcat to transfer files

Netcat can also be used to transfer files:

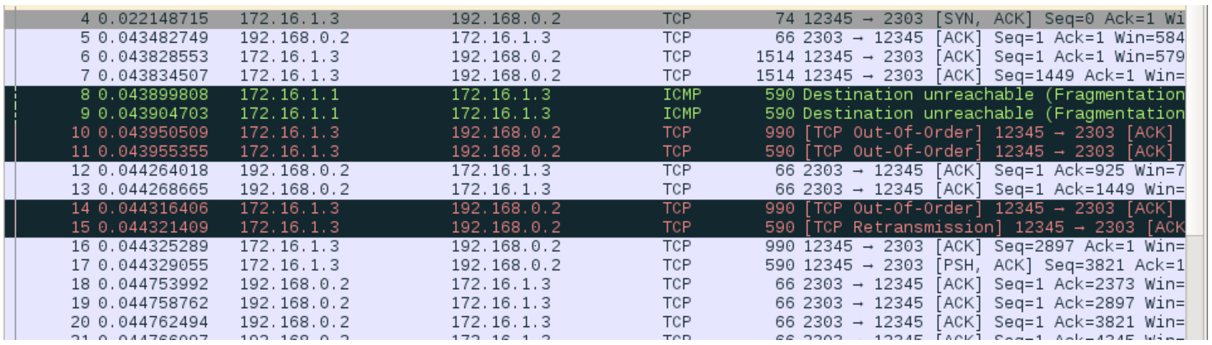
1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Start the netcat server in host3, that will send the file /etc/services towards the client when the connection has been established:

```
host3$ nc -l -p 12345 -q 0 </etc/services
```

4. Start the netcat client in host2, that will display in the screen the file /etc/services of the server.

```
host2$ nc 172.16.1.3 12345
```

SImNet3:



```
 4 0.022148715   172.16.1.3    192.168.0.2   TCP     74 12345 → 2303 [SYN, ACK] Seq=0 Ack=1 Wi
 5 0.043482749   192.168.0.2   172.16.1.3    TCP     66 2303 → 12345 [ACK] Seq=1 Ack=1 Win=584
 6 0.043828553   172.16.1.3    192.168.0.2   TCP   1514 12345 → 2303 [ACK] Seq=1 Ack=1 Win=579
 7 0.043834507   172.16.1.3    192.168.0.2   TCP   1514 12345 → 2303 [ACK] Seq=1449 Ack=1 Win=
 8 0.043899808   172.16.1.1    172.16.1.3    ICMP   590 Destination unreachable (Fragmentation
 9 0.043904703   172.16.1.1    172.16.1.3    ICMP   590 Destination unreachable (Fragmentation
10 0.043950509   172.16.1.3    192.168.0.2   TCP    990 [TCP Out-Of-Order] 12345 → 2303 [ACK]
11 0.043955355   172.16.1.3    192.168.0.2   TCP    590 [TCP Out-Of-Order] 12345 → 2303 [ACK]
12 0.044264018   192.168.0.2   172.16.1.3    TCP     66 2303 → 12345 [ACK] Seq=1 Ack=925 Win=7
13 0.044268665   192.168.0.2   172.16.1.3    TCP     66 2303 → 12345 [ACK] Seq=1 Ack=1449 Win=
14 0.044316406   172.16.1.3    192.168.0.2   TCP    990 [TCP Out-Of-Order] 12345 → 2303 [ACK]
15 0.044321409   172.16.1.3    192.168.0.2   TCP    590 [TCP Retransmission] 12345 → 2303 [ACK
16 0.044325289   172.16.1.3    192.168.0.2   TCP    990 12345 → 2303 [ACK] Seq=2897 Ack=1 Win=
17 0.044329055   172.16.1.3    192.168.0.2   TCP    590 12345 → 2303 [PSH, ACK] Seq=3821 Ack=1
18 0.044753992   192.168.0.2   172.16.1.3    TCP     66 2303 → 12345 [ACK] Seq=1 Ack=2373 Win=
19 0.044758762   192.168.0.2   172.16.1.3    TCP     66 2303 → 12345 [ACK] Seq=1 Ack=2897 Win=
20 0.044762494   192.168.0.2   172.16.1.3    TCP     66 2303 → 12345 [ACK] Seq=1 Ack=3821 Win=
```

a. In the 3-way handshake, which was the value of the MSS?
b. Did you notice any problem during the transmission according to the wiresharks?
c. Can you see any ICMP message in SimNet3? Who is sending these error messages and why?
d. Was the file transmitted properly? Who solved the problem?
e. Is this meaning that the MSS has changed during the transmission?

El MTU és 460. A la SN3 → 2pq. DU (FN) (172.16.1.1 → 172.116.1.3) (R2-host3). Dins el paquet s'envia la MTU informant de que en aquell punt és 976.
 El fitxer s'ha retransmès correctament. El problema el soluciona el host2 fragmentant els paquet, ja que aquest es avisat pel host3 de la necessitat de fragmentació, perquè aquest ha sigut avisat prèviament per R2, qui ha detectat el problema.

As you can see from the previous example, TCP is a reliable protocol. This means that it will retransmit lost information until it is properly received (or after a certain number of unsuccessful retransmissions). TCP is able to change the lenght of the segments in case that the network reports problems of size, but this does not mean that the MSS changes, as it is a fixed value.
Obviously, files can be transmitted in both directions using netcat. In the previous example, we made a transmission in which the server transferred a file towards the client. Now we will test this same operation in the reverse direction.

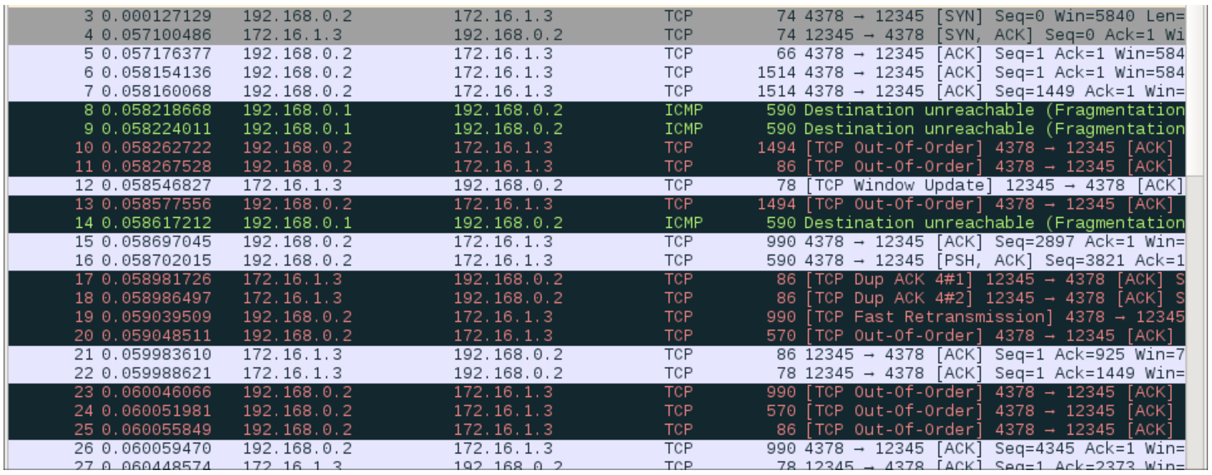
1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Start the netcat server in host3:
4. Start the netcat client in host2, but in this case inject the file /etc/services of the client towards the server using the TCP connection.

a. Which are the commands that you used to do the previous behavior?
b. Did you notice any problem during the transmission according to the wiresharks?
c. Can you see any ICMP message in SimNet0 and SimNet1? Who is sending these error messages and why?
d. Was the file transmitted properly? Who solved the problem?
e. According to the transmission, who is the TCP client and who is the TCP server?

```
host3:~# nc -l -p 12345
```

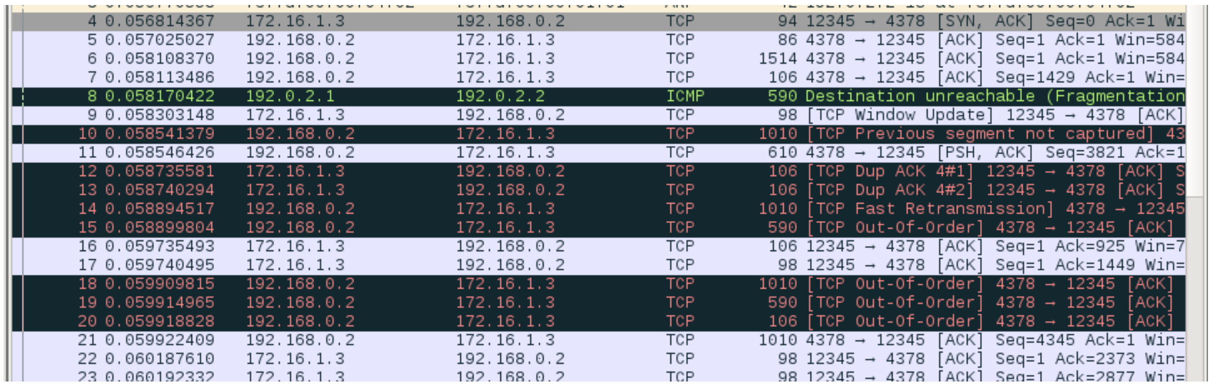
```
host2:~# nc 172.16.1.3 12345 -q 0 </etc/services
```

SimNet0:



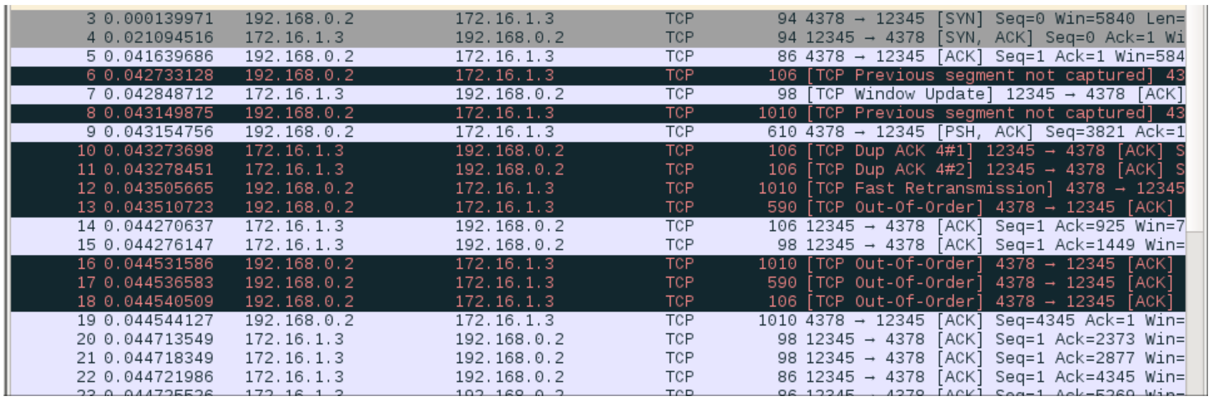
```
 3 0.000127129   192.168.0.2   172.16.1.3    TCP     74 4378 → 12345 [SYN] Seq=0 Win=5840 Len=
 4 0.057100486   172.16.1.3    192.168.0.2   TCP     74 12345 → 4378 [SYN, ACK] Seq=0 Ack=1 Wi
 5 0.057176377   192.168.0.2   172.16.1.3    TCP     66 4378 → 12345 [ACK] Seq=1 Ack=1 Win=584
 6 0.058154136   192.168.0.2   172.16.1.3    TCP   1514 4378 → 12345 [ACK] Seq=1 Ack=1 Win=584
 7 0.058160068   192.168.0.2   172.16.1.3    TCP   1514 4378 → 12345 [ACK] Seq=1449 Ack=1 Win=
 8 0.058218668   192.168.0.1   192.168.0.2   ICMP   590 Destination unreachable (Fragmentation
 9 0.058224011   192.168.0.1   192.168.0.2   ICMP   590 Destination unreachable (Fragmentation
10 0.058262722   192.168.0.2   172.16.1.3    TCP   1494 [TCP Out-Of-Order] 4378 → 12345 [ACK]
11 0.058267528   192.168.0.2   172.16.1.3    TCP     86 [TCP Out-Of-Order] 4378 → 12345 [ACK]
12 0.058546827   172.16.1.3    192.168.0.2   TCP     78 [TCP Window Update] 12345 → 4378 [ACK]
13 0.058577556   192.168.0.2   172.16.1.3    TCP   1494 [TCP Out-Of-Order] 4378 → 12345 [ACK]
14 0.058617212   192.168.0.1   192.168.0.2   ICMP   590 Destination unreachable (Fragmentation
15 0.058697045   192.168.0.2   172.16.1.3    TCP    990 4378 → 12345 [ACK] Seq=2897 Ack=1 Win=
16 0.058702015   192.168.0.2   172.16.1.3    TCP    590 4378 → 12345 [PSH, ACK] Seq=3821 Ack=1
17 0.058981726   172.16.1.3    192.168.0.2   TCP     86 [TCP Dup ACK 4#1] 12345 → 4378 [ACK] S
18 0.058986497   172.16.1.3    192.168.0.2   TCP     86 [TCP Dup ACK 4#2] 12345 → 4378 [ACK] S
19 0.059039509   192.168.0.2   172.16.1.3    TCP    990 [TCP Fast Retransmission] 4378 → 12345
20 0.059048511   192.168.0.2   172.16.1.3    TCP    570 [TCP Out-Of-Order] 4378 → 12345 [ACK]
21 0.059983610   172.16.1.3    192.168.0.2   TCP     86 12345 → 4378 [ACK] Seq=1 Ack=925 Win=7
22 0.059988621   172.16.1.3    192.168.0.2   TCP     78 12345 → 4378 [ACK] Seq=1 Ack=1449 Win=
23 0.060046066   192.168.0.2   172.16.1.3    TCP    990 [TCP Out-Of-Order] 4378 → 12345 [ACK]
24 0.060051981   192.168.0.2   172.16.1.3    TCP    570 [TCP Out-Of-Order] 4378 → 12345 [ACK]
25 0.060055849   192.168.0.2   172.16.1.3    TCP     86 [TCP Out-Of-Order] 4378 → 12345 [ACK]
26 0.060059470   192.168.0.2   172.16.1.3    TCP    990 4378 → 12345 [ACK] Seq=4345 Ack=1 Win=
27 0.060448574   172.16.1.3    192.168.0.2   TCP     78 12345 → 4378 [ACK] Seq=1 Ack=2373 Win=
```

SimNet1:



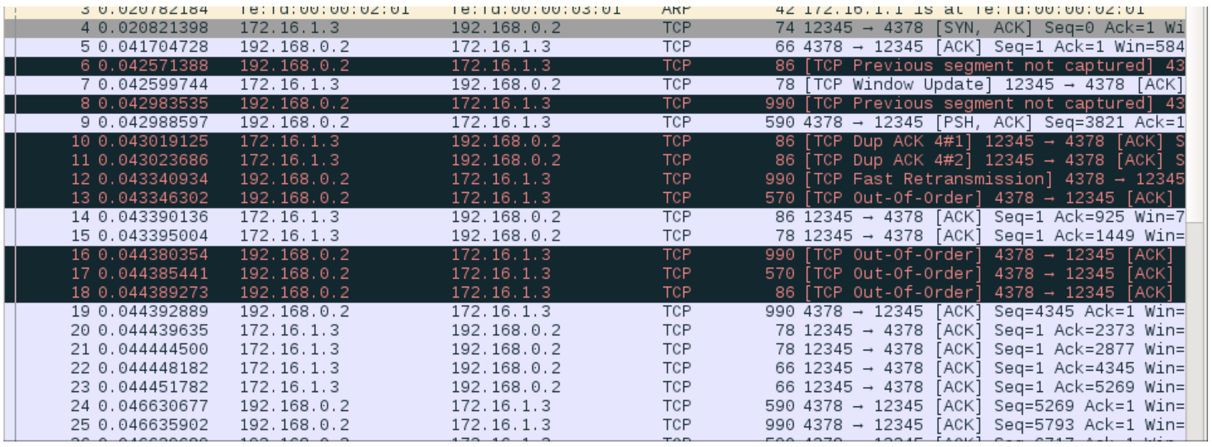
```
 4 0.056814367   172.16.1.3    192.168.0.2   TCP     94 12345 → 4378 [SYN, ACK] Seq=0 Ack=1 Wi
 5 0.057025027   192.168.0.2   172.16.1.3    TCP     86 4378 → 12345 [ACK] Seq=1 Ack=1 Win=584
 6 0.058108370   192.168.0.2   172.16.1.3    TCP   1514 4378 → 12345 [ACK] Seq=1 Ack=1 Win=584
 7 0.058113486   192.168.0.2   172.16.1.3    TCP    106 4378 → 12345 [ACK] Seq=1429 Ack=1 Win=
 8 0.058170422   192.0.2.1     192.0.2.2     ICMP   590 Destination unreachable (Fragmentation
 9 0.058303148   172.16.1.3    192.168.0.2   TCP     98 [TCP Window Update] 12345 → 4378 [ACK]
10 0.058541379   192.168.0.2   172.16.1.3    TCP   1010 [TCP Previous segment not captured] 43
11 0.058546426   192.168.0.2   172.16.1.3    TCP    610 4378 → 12345 [PSH, ACK] Seq=3821 Ack=1
12 0.058735581   172.16.1.3    192.168.0.2   TCP    106 [TCP Dup ACK 4#1] 12345 → 4378 [ACK] S
13 0.058740294   172.16.1.3    192.168.0.2   TCP    106 [TCP Dup ACK 4#2] 12345 → 4378 [ACK] S
14 0.058894517   192.168.0.2   172.16.1.3    TCP   1010 [TCP Fast Retransmission] 4378 → 12345
15 0.058899804   192.168.0.2   172.16.1.3    TCP    590 [TCP Out-Of-Order] 4378 → 12345 [ACK]
16 0.059735493   172.16.1.3    192.168.0.2   TCP    106 12345 → 4378 [ACK] Seq=1 Ack=925 Win=7
17 0.059740495   172.16.1.3    192.168.0.2   TCP     98 12345 → 4378 [ACK] Seq=1 Ack=1449 Win=
18 0.059909815   192.168.0.2   172.16.1.3    TCP   1010 [TCP Out-Of-Order] 4378 → 12345 [ACK]
19 0.059914965   192.168.0.2   172.16.1.3    TCP    590 [TCP Out-Of-Order] 4378 → 12345 [ACK]
20 0.059918828   192.168.0.2   172.16.1.3    TCP    106 [TCP Out-Of-Order] 4378 → 12345 [ACK]
21 0.059922409   192.168.0.2   172.16.1.3    TCP   1010 4378 → 12345 [ACK] Seq=4345 Ack=1 Win=
22 0.060187610   172.16.1.3    192.168.0.2   TCP     98 12345 → 4378 [ACK] Seq=1 Ack=2373 Win=
23 0.060192332   172.16.1.3    192.168.0.2   TCP     98 12345 → 4378 [ACK] Seq=1 Ack=2877 Win=
```

SimNet2:



```
 3 0.000139971   192.168.0.2   172.16.1.3    TCP     94 4378 → 12345 [SYN] Seq=0 Win=5840 Len=
 4 0.021094516   172.16.1.3    192.168.0.2   TCP     94 12345 → 4378 [SYN, ACK] Seq=0 Ack=1 Wi
 5 0.041639686   192.168.0.2   172.16.1.3    TCP     86 4378 → 12345 [ACK] Seq=1 Ack=1 Win=584
 6 0.042733128   192.168.0.2   172.16.1.3    TCP    106 [TCP Previous segment not captured] 43
 7 0.042848712   172.16.1.3    192.168.0.2   TCP     98 [TCP Window Update] 12345 → 4378 [ACK]
 8 0.043149875   192.168.0.2   172.16.1.3    TCP   1010 [TCP Previous segment not captured] 43
 9 0.043154756   192.168.0.2   172.16.1.3    TCP    610 4378 → 12345 [PSH, ACK] Seq=3821 Ack=1
10 0.043273698   172.16.1.3    192.168.0.2   TCP    106 [TCP Dup ACK 4#1] 12345 → 4378 [ACK] S
11 0.043278451   172.16.1.3    192.168.0.2   TCP    106 [TCP Dup ACK 4#2] 12345 → 4378 [ACK] S
12 0.043505665   192.168.0.2   172.16.1.3    TCP   1010 [TCP Fast Retransmission] 4378 → 12345
13 0.043510723   192.168.0.2   172.16.1.3    TCP    590 [TCP Out-Of-Order] 4378 → 12345 [ACK]
14 0.044270637   172.16.1.3    192.168.0.2   TCP    106 12345 → 4378 [ACK] Seq=1 Ack=925 Win=7
15 0.044276147   172.16.1.3    192.168.0.2   TCP     98 12345 → 4378 [ACK] Seq=1 Ack=1449 Win=
16 0.044531586   192.168.0.2   172.16.1.3    TCP   1010 [TCP Out-Of-Order] 4378 → 12345 [ACK]
17 0.044536583   192.168.0.2   172.16.1.3    TCP    590 [TCP Out-Of-Order] 4378 → 12345 [ACK]
18 0.044540509   192.168.0.2   172.16.1.3    TCP    106 [TCP Out-Of-Order] 4378 → 12345 [ACK]
19 0.044544127   192.168.0.2   172.16.1.3    TCP   1010 4378 → 12345 [ACK] Seq=4345 Ack=1 Win=
20 0.044713549   172.16.1.3    192.168.0.2   TCP     98 12345 → 4378 [ACK] Seq=1 Ack=2373 Win=
21 0.044718349   172.16.1.3    192.168.0.2   TCP     98 12345 → 4378 [ACK] Seq=1 Ack=2877 Win=
22 0.044721986   172.16.1.3    192.168.0.2   TCP     86 12345 → 4378 [ACK] Seq=1 Ack=4345 Win=
```

SimNet3:



```
  3 0.020782184  fe:fd:00:00:02:01  fe:fd:00:00:03:01  ARP    42 172.16.1.1 is at fe:fd:00:00:02:01
  4 0.020821398  172.16.1.3         192.168.0.2        TCP    74 12345 → 4378 [SYN, ACK] Seq=0 Ack=1 Wi
  5 0.041704728  192.168.0.2        172.16.1.3         TCP    66 4378 → 12345 [ACK] Seq=1 Ack=1 Win=584
  6 0.042571388  192.168.0.2        172.16.1.3         TCP    86 [TCP Previous segment not captured] 43
  7 0.042599744  172.16.1.3         192.168.0.2        TCP    78 [TCP Window Update] 12345 → 4378 [ACK]
  8 0.042983535  192.168.0.2        172.16.1.3         TCP   990 [TCP Previous segment not captured] 43
  9 0.042988597  192.168.0.2        172.16.1.3         TCP   590 4378 → 12345 [PSH, ACK] Seq=3821 Ack=1
 10 0.043019125  172.16.1.3         192.168.0.2        TCP    86 [TCP Dup ACK 4#1] 12345 → 4378 [ACK] S
 11 0.043023686  172.16.1.3         192.168.0.2        TCP    86 [TCP Dup ACK 4#2] 12345 → 4378 [ACK] S
 12 0.043340934  192.168.0.2        172.16.1.3         TCP   990 [TCP Fast Retransmission] 4378 → 12345
 13 0.043346302  192.168.0.2        172.16.1.3         TCP   570 [TCP Out-Of-Order] 4378 → 12345 [ACK]
 14 0.043390136  172.16.1.3         192.168.0.2        TCP    86 12345 → 4378 [ACK] Seq=1 Ack=925 Win=7
 15 0.043395004  172.16.1.3         192.168.0.2        TCP    78 12345 → 4378 [ACK] Seq=1 Ack=1449 Win=
 16 0.044380354  192.168.0.2        172.16.1.3         TCP   990 [TCP Out-Of-Order] 4378 → 12345 [ACK]
 17 0.044385441  192.168.0.2        172.16.1.3         TCP   570 [TCP Out-Of-Order] 4378 → 12345 [ACK]
 18 0.044389273  192.168.0.2        172.16.1.3         TCP    86 [TCP Out-Of-Order] 4378 → 12345 [ACK]
 19 0.044392889  192.168.0.2        172.16.1.3         TCP   990 4378 → 12345 [ACK] Seq=4345 Ack=1 Win=
 20 0.044439635  172.16.1.3         192.168.0.2        TCP    78 12345 → 4378 [ACK] Seq=1 Ack=2373 Win=
 21 0.044444500  172.16.1.3         192.168.0.2        TCP    78 12345 → 4378 [ACK] Seq=1 Ack=2877 Win=
 22 0.044448182  172.16.1.3         192.168.0.2        TCP    66 12345 → 4378 [ACK] Seq=1 Ack=4345 Win=
 23 0.044451782  172.16.1.3         192.168.0.2        TCP    66 12345 → 4378 [ACK] Seq=1 Ack=5269 Win=
 24 0.046630677  192.168.0.2        172.16.1.3         TCP   590 4378 → 12345 [ACK] Seq=5269 Ack=1 Win=
 25 0.046635902  192.168.0.2        172.16.1.3         TCP   990 4378 → 12345 [ACK] Seq=5793 Ack=1 Win=
```

Again, TCP solved the problem by adapting the lenght of the segments to the network conditions and retransmitting lost information. Notice that the TCP client is the host that started the TCP dialogue by sending the first message SYN, and the TCP server is the host that answered by sending the SYN-ACK message, no matter of the direction of the data transmission.

<u>TCP, tunnels and filtering</u>

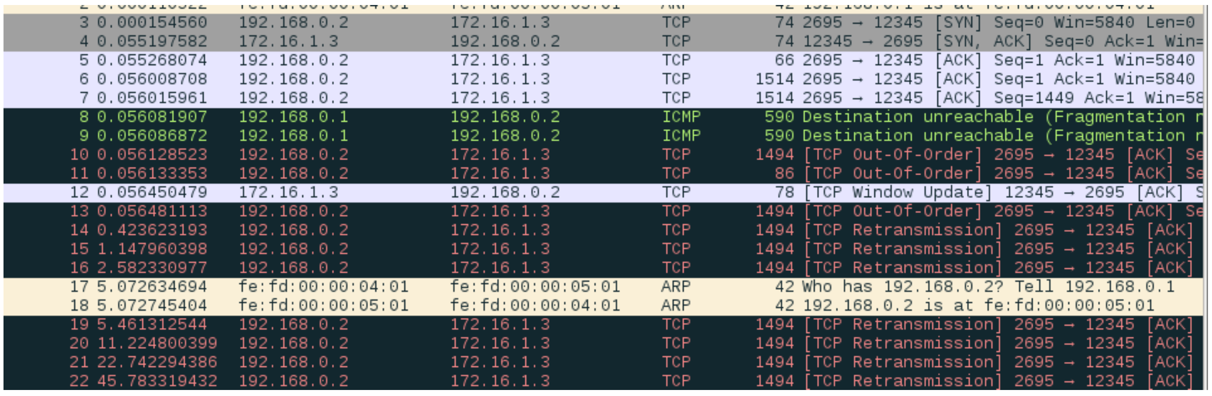
A problem that appears in actual networks is that some networks administrators filter all traffic that they suppose to be dangerous. In this case, it is usual to filter almost all UDP traffic (except the DNS) and in general all ICMP traffic.

Now we are going to test what happens when error messages are not able to reach the destination host. In particular, we will see which are the consequences of filtering Fragmentation Needed messages.

1. Let us imagine that the network administrator of R1 wants to protect this router against typical attacks that use ICMP. For this reason, it avoids sending ICMP messages to this router. So, execute the following command to filter the ICMP traffic whose destination is R1:

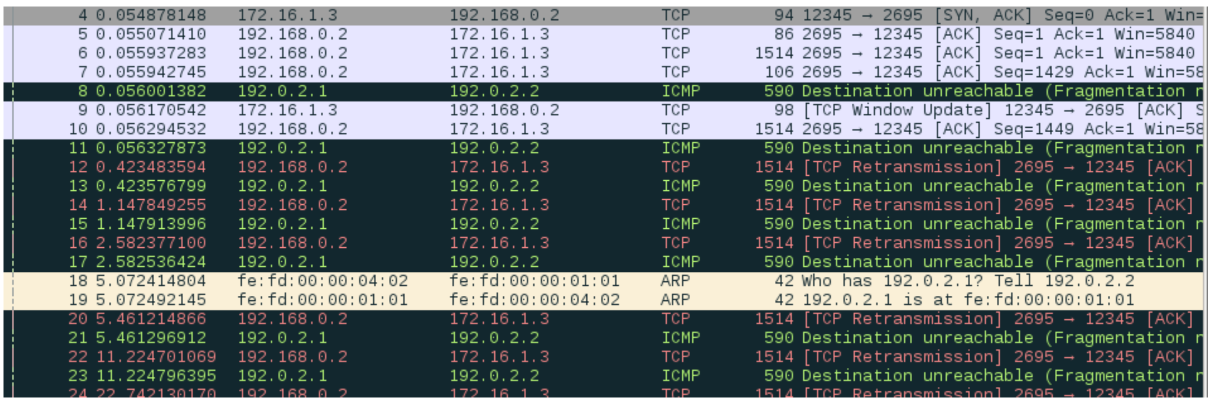
```
R1$ iptables −A INPUT −p icmp −j DROP
```

2. Delete the kernel routing cache executing the label flushcache.
3. Put all wiresharks listening traffic in all SimNet interfaces.
4. Again using netcat, send the file /etc/services from host2 (acting as client) towards host3 (acting as server).
   SimNet0:



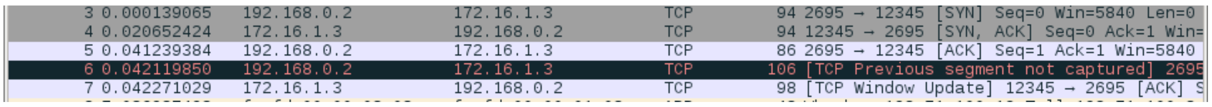
```
  3 0.000154560  192.168.0.2        172.16.1.3         TCP    74 2695 → 12345 [SYN] Seq=0 Win=5840 Len=0
  4 0.055197582  172.16.1.3         192.168.0.2        TCP    74 12345 → 2695 [SYN, ACK] Seq=0 Ack=1 Win=
  5 0.055268074  192.168.0.2        172.16.1.3         TCP    66 2695 → 12345 [ACK] Seq=1 Ack=1 Win=5840
  6 0.056008708  192.168.0.2        172.16.1.3         TCP  1514 2695 → 12345 [ACK] Seq=1 Ack=1 Win=5840
  7 0.056015961  192.168.0.2        172.16.1.3         TCP  1514 2695 → 12345 [ACK] Seq=1449 Ack=1 Win=58
  8 0.056081907  192.168.0.1        192.168.0.2        ICMP  590 Destination unreachable (Fragmentation r
  9 0.056086872  192.168.0.1        192.168.0.2        ICMP  590 Destination unreachable (Fragmentation r
 10 0.056128523  192.168.0.2        172.16.1.3         TCP  1494 [TCP Out-Of-Order] 2695 → 12345 [ACK] Se
 11 0.056133353  192.168.0.2        172.16.1.3         TCP    86 [TCP Out-Of-Order] 2695 → 12345 [ACK] Se
 12 0.056450479  172.16.1.3         192.168.0.2        TCP    78 [TCP Window Update] 12345 → 2695 [ACK] S
 13 0.056481113  192.168.0.2        172.16.1.3         TCP  1494 [TCP Out-Of-Order] 2695 → 12345 [ACK] Se
 14 0.423623193  192.168.0.2        172.16.1.3         TCP  1494 [TCP Retransmission] 2695 → 12345 [ACK]
 15 1.147960398  192.168.0.2        172.16.1.3         TCP  1494 [TCP Retransmission] 2695 → 12345 [ACK]
 16 2.582330977  192.168.0.2        172.16.1.3         TCP  1494 [TCP Retransmission] 2695 → 12345 [ACK]
 17 5.072634694  fe:fd:00:00:04:01  fe:fd:00:00:05:01  ARP    42 Who has 192.168.0.2? Tell 192.168.0.1
 18 5.072745404  fe:fd:00:00:05:01  fe:fd:00:00:04:01  ARP    42 192.168.0.2 is at fe:fd:00:00:05:01
 19 5.461312544  192.168.0.2        172.16.1.3         TCP  1494 [TCP Retransmission] 2695 → 12345 [ACK]
 20 11.224800399 192.168.0.2        172.16.1.3         TCP  1494 [TCP Retransmission] 2695 → 12345 [ACK]
 21 22.742294386 192.168.0.2        172.16.1.3         TCP  1494 [TCP Retransmission] 2695 → 12345 [ACK]
 22 45.783319432 192.168.0.2        172.16.1.3         TCP  1494 [TCP Retransmission] 2695 → 12345 [ACK]
```

   SimNet1:



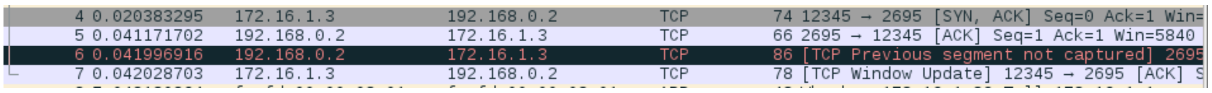
```
 4 0.054878148   172.16.1.3          192.168.0.2         TCP      94 12345 → 2695 [SYN, ACK] Seq=0 Ack=1 Win=
 5 0.055071410   192.168.0.2         172.16.1.3          TCP      86 2695 → 12345 [ACK] Seq=1 Ack=1 Win=5840
 6 0.055937283   192.168.0.2         172.16.1.3          TCP    1514 2695 → 12345 [ACK] Seq=1 Ack=1 Win=5840
 7 0.055942745   192.168.0.2         172.16.1.3          TCP     106 2695 → 12345 [ACK] Seq=1429 Ack=1 Win=58
 8 0.056001382   192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
 9 0.056170542   172.16.1.3          192.168.0.2         TCP      98 [TCP Window Update] 12345 → 2695 [ACK] S
10 0.056294532   192.168.0.2         172.16.1.3          TCP    1514 2695 → 12345 [ACK] Seq=1449 Ack=1 Win=58
11 0.056327873   192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
12 0.423483594   192.168.0.2         172.16.1.3          TCP    1514 [TCP Retransmission] 2695 → 12345 [ACK]
13 0.423576799   192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
14 1.147849255   192.168.0.2         172.16.1.3          TCP    1514 [TCP Retransmission] 2695 → 12345 [ACK]
15 1.147913996   192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
16 2.582377100   192.168.0.2         172.16.1.3          TCP    1514 [TCP Retransmission] 2695 → 12345 [ACK]
17 2.582536424   192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
18 5.072414804   fe:fd:00:00:04:02   fe:fd:00:00:01:01   ARP      42 Who has 192.0.2.1? Tell 192.0.2.2
19 5.072492145   fe:fd:00:00:01:01   fe:fd:00:00:04:02   ARP      42 192.0.2.1 is at fe:fd:00:00:01:01
20 5.461214866   192.168.0.2         172.16.1.3          TCP    1514 [TCP Retransmission] 2695 → 12345 [ACK]
21 5.461296912   192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
22 11.224701069  192.168.0.2         172.16.1.3          TCP    1514 [TCP Retransmission] 2695 → 12345 [ACK]
23 11.224796395  192.0.2.1           192.0.2.2           ICMP    590 Destination unreachable (Fragmentation n
24 22.742130170  192.168.0.2         172.16.1.3          TCP    1514 [TCP Retransmission] 2695 → 12345 [ACK]
```

SimNet2:



```
3 0.000139065   192.168.0.2   172.16.1.3    TCP    94 2695 → 12345 [SYN] Seq=0 Win=5840 Len=0
4 0.020652424   172.16.1.3    192.168.0.2   TCP    94 12345 → 2695 [SYN, ACK] Seq=0 Ack=1 Win=
5 0.041239384   192.168.0.2   172.16.1.3    TCP    86 2695 → 12345 [ACK] Seq=1 Ack=1 Win=5840
6 0.042119850   192.168.0.2   172.16.1.3    TCP   106 [TCP Previous segment not captured] 2695
7 0.042271029   172.16.1.3    192.168.0.2   TCP    98 [TCP Window Update] 12345 → 2695 [ACK] S
```

SImNet3:



```
4 0.020383295   172.16.1.3    192.168.0.2   TCP    74 12345 → 2695 [SYN, ACK] Seq=0 Ack=1 Win=
5 0.041171702   192.168.0.2   172.16.1.3    TCP    66 2695 → 12345 [ACK] Seq=1 Ack=1 Win=5840
6 0.041996916   192.168.0.2   172.16.1.3    TCP    86 [TCP Previous segment not captured] 2695
7 0.042028703   172.16.1.3    192.168.0.2   TCP    78 [TCP Window Update] 12345 → 2695 [ACK] S
```

a. Can you see the 3-way handshake? Did you notice problems with these 3 initial messages?
   Es pot veure el handshake inicial, el qual no presenta cap problema.
b. Did you notice problems with data segments? Why? Which is the difference with the previous ones?
   Hi ha problemes amb l'enviament de trames a causa de que els paquets cada cert temps cada vegada més grans sol·licitant retransmissió.
c. Can you see any ICMP message in SimNet1? And in SimNet0?
   Podem veure com els paquets ICMP DU(FN) arriben al SN1 però al SN0 no a causa del filtre.
d. Was the file transmitted properly? Why?
   El fitxer no s'ha enviat correctament a causa dels problemes de fragmentació.
e. What will happen if instead of the INPUT chain, we use the FORWARD chain in the filtering rule? (R1#: iptables -D FORWARD -p icmp -j DROP)
   Els missatges ICMP destinats a ell poden arribar, els quals són els necessaris per saber dels problemes de fragmentació i el fitxer es transmet correctament.

As you can see, the client is not receiving feedback from the errors of the network due to the filtering rule, so it is unable to addapt the TCP transmission to the conditions of the network. TCP assumes that losses are due to congestion, so it will try to retransmit lost segments just waiting more and more time.

Changing the MSS

Obviously, the previous problem can be solved by deleting the iptables filter rule in R1, but this may be impossible if we do not have the admin rights to do so. As an alternative solution, it is possible to change the advertised MSS to a different value, so tunneled packets do not need to be fragmented in path. So, we have to change the advertised MSS in the 3-way handshake to a different value from the one calculated by default in the host (and which mainly depends on the MTU of the exit network interface).

There are some issues that should be considered prior to doing this change:

- Which is the value of the MSS that avoids fragmentation in path? To calculate this, we should clearly know and understand the notion of MTU, MSS and encapsulation. Remember that the MSS counts only DATA octets in the segment, and it does not count the TCP header or the IP header. So, the MSS is calculated as the most restrictive MTU in the path minus the minimum IP, TCP and tunnel headers.

- Which message in the 3-way handshake (SYN or SYN/ACK) should be modified. Remember that the client informs to the server about its MSS using the SYN segment (so this affects data that go from the server to the client), meanwhile the server informs to the client about its MSS using the SYN/ACK segment (data from the client to the server).
- Where we can change this MSS, hosts or routers?
    - Changing the MSS in the end systems (origin or destination). In Linux systems it is possible to force that the TCP connections that use a certain route use a particular MSS during the establishment phase. In general, the command is ip route add ROUTE advmss VALUE where ROUTE is the route we want to add (more information using ip route show), and VALUE is the "Advertised MSS" in bytes. In case the route exists, instead of add use change. Just as an example (not related with this particular case of the lab session), let us imagine that we want to change the MSS that host2 advertises in the 3-way handshake to the value 1440. In this particular case, the command would be:

```
host2$ip route add 172.16.1.0/24 via 192.168.0.1 advmss 1440
```

    - Changing the MSS in the routers. If the TCP traffic goes through a router that we manage, it is possible to modify the advertised MSS in the 3-way handshake. To do so, we will use iptables, and in particular the "mangle" table. This table is used to modify packets, and it may be used with the chains PREROUTING, INPUT, FORWARD, OUTPUT and POSTROUTING. The chain associated to the change of MSS is TCPMSS, and it has two parameters:
      `--set-mss value`: to establish the MSS to `value` in bytes.
      `--clamp-mss-to-pmtu`: to establish the MSS to the one obtained via PMTUD.

      For instance:

```
iptables -t mangle -A FORWARD -o tunnel0 -p tcp --syn -j TCPMSS --set-mss 1440
```

      With this command, the linux router will change the MSS to 1440 bytes to those TCP packets with the SYN flag set (and the rest of flags not set) and that exit the router via the interface tunnel0.

Change the MSS in the host

So, we are going to try to solve the previous transmission but changing the MSS in the hosts.

1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Identify which are the proper parameters to change the mss properly:
    - Which is the value of the advertised MSS to avoid fragmentation in path? Notice that the MTU in SimNet2=996.
    - Which message of the 3-way handshake of TCP should be change? Why?
    - If we are changing the MSS in a host, which is the host where we should execute the command?
      MSS = 996B (MTUrestrictiva) - 20B (HIPInner) - 20B (HIPOuter) - 20B (Htcp) = 936B
      S'ha de canviar el SYN/ACK, perquè conté el valor del MSS del servidor.
      S'ha de canviar al host3, ja que és el que té la informació del MSS del servidor.
4. Now that you fully know all the parameters to properly change the MSS, use ip route to change it.

```
host3#: ip route add 192.168.0.0/24 via 172.16.1.1 advmss 936
// via = gateway
```

5. Using netcat again, send the file /etc/services from host2 (acting as client) towards host3 (acting as server).

```
host3#: nc -l -p 12345
host2#: nc 172.16.1.3 12345 -q 0 </etc/services → ok!
```

a. Did you notice problems with data segments?
b. Can you see any ICMP message in SimNet1 and SimNet0?
c. Was the file transmitted properly?

File transmited perfectly.

Prior to continue with the following test, delete the route that you configured in the host to change the MSS.

Change the MSS in the router

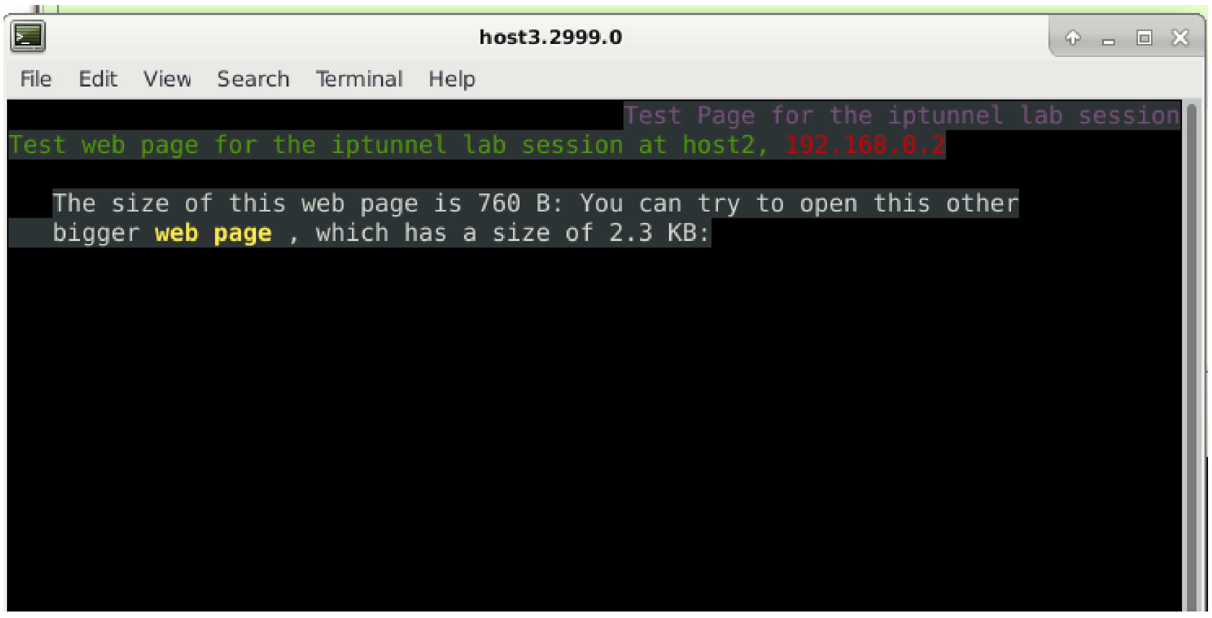
Now we are going to simulate a quite typical problem that appear when doing web browsing and tunnels. There is a web server (apache2) listening in host2. Notice that when using simctl, we cannot use web browsers that require the graphical interface (like firefox or chrome). Instead, we are going to use lynx, a text-based web browser. To see the web page, you can go to host2 and execute the command:

```
host2$ lynx localhost
```

As you can see, the web page is very simple, it only contains some few sentences and a link towards another local web page. You can browse that other web page by using the arrow keys. Obviously, we do not pretend to connect to the web server locally, but remotelly from host3. So, let us start the experiment:

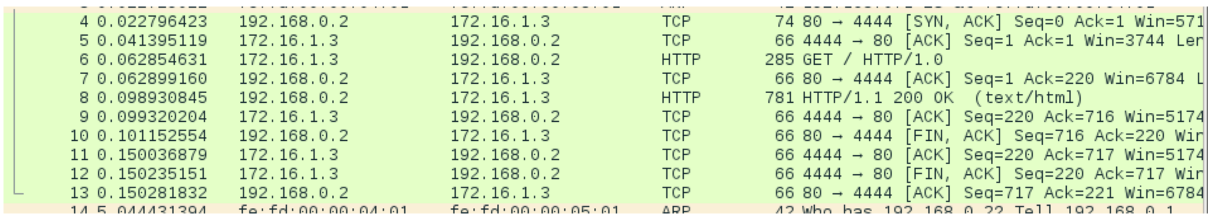
1. Delete the kernel routing cache executing the label flushcache.
2. Put all wiresharks listening traffic in all SimNet interfaces.
3. Connect to the web server of host2 using lynx from host3:

```
host3$ lynx 192.168.0.2
```



```
                                              Test Page for the iptunnel lab session
Test web page for the iptunnel lab session at host2, 192.168.0.2

   The size of this web page is 760 B: You can try to open this other
   bigger web page , which has a size of 2.3 KB:
```

SimNet0:



```
 4 0.022796423  192.168.0.2   172.16.1.3   TCP    74 80 → 4444 [SYN, ACK] Seq=0 Ack=1 Win=571
 5 0.041395119  172.16.1.3    192.168.0.2  TCP    66 4444 → 80 [ACK] Seq=1 Ack=1 Win=3744 Ler
 6 0.062854631  172.16.1.3    192.168.0.2  HTTP  285 GET / HTTP/1.0
 7 0.062899160  192.168.0.2   172.16.1.3   TCP    66 80 → 4444 [ACK] Seq=1 Ack=220 Win=6784 L
 8 0.098930845  192.168.0.2   172.16.1.3   HTTP  781 HTTP/1.1 200 OK  (text/html)
 9 0.099320204  172.16.1.3    192.168.0.2  TCP    66 4444 → 80 [ACK] Seq=220 Ack=716 Win=5174
10 0.101152554  192.168.0.2   172.16.1.3   TCP    66 80 → 4444 [FIN, ACK] Seq=716 Ack=220 Wir
11 0.150036879  172.16.1.3    192.168.0.2  TCP    66 4444 → 80 [ACK] Seq=220 Ack=717 Win=5174
12 0.150235151  172.16.1.3    192.168.0.2  TCP    66 4444 → 80 [FIN, ACK] Seq=220 Ack=717 Wir
13 0.150281832  192.168.0.2   172.16.1.3   TCP    66 80 → 4444 [ACK] Seq=717 Ack=221 Win=6784
14 5.044431394  fe:fd:00:00:04:01  fe:fd:00:00:05:01  ARP  42 Who has 192.168.0.2? Tell 192.168.0.1
```

a. Can you see the main web page of host2? YES
b. Can you see any error message in the wiresharks? NO

As you can see there is no problem to see the main web page of this host. However, browse the web in order to see the other local web page:

a. Did you noticed any error?
b. What do you see in the wiresharks?
c. Why do you think that the main web page is working, but the second one cannot be downloaded?
d. Try to solve the problem by changing the MSS in R2 using the MANGLE table of iptables. Remember that you have to change the MSS in the proper message of the 3-way handshake.

We hope that all these experiments helped you to understand how complicated is to solve problems of communication networks when using tunnels, even in the case of using the most simplistic IPIP tunnel. In following lab sessions, maybe you will be asked to create GRE tunnels, so try to remember all these concepts.

host2#: lynx localhost → pàgina web del host2
host3#: lynx 192.168.0.2 → pàgina web de host2 visualitzada desde host3
En aquest segon intent no podem accedir a la segona pàgina web ja que és més gran i necessita fragmentació, però el DF=Set. Veiem un missatge d'error de R1→ host2 DU(FN).
Per canviar MSS:

```
R2#: iptables -t mangle -A FORWARD -o tunnel0 -p
```