Practice 8 Memo of TCGI

Exercice 0.4

In order to configure both endpoints for the `IPIP` tunnel we run the following commands for `R2`:

```
R2:~# ip tunnel add tunnel0 mode ipip
local 198.51.100.2 remote 192.0.2.2 ttl
inherit nopmtudisc dev eth2
R2:~# ifconfig tunnel  1.2.3.4
R2:~# route add -net 192.168.0.0/24 dev
tunnel0
```

And for `R1`:

```
R1:~# ip tunnel add tunnel0 mode ipip
local 192.0.2.2 remote 198.51.100.2 ttl
inherit nopmtudisc dev eth2
R1:~# ifconfig tunnel0 4.3.2.1
R1:~# route add -net 172.16.1.0/24 dev
tunnel0
```

Exercice 0.5

While capturing in all `SimNets` we run `ping -c1 172.16.1.3` on `host2` in order to reach `host3`.

- The IP frames captured on `SimNet0` and `SimNet3` are the standard ones with no modification whatsoever. The src `@IP` thow, is the private one, not the public.
- On the other hand, the frames captured on the other two `SimNets` have 2 `IP` headers. As it can be seen on the image:

```
* The source and destination `@IP` of
the outer header are the ones from
the endpoints of the tunnel whereas the
`@IP` of the inner header feature
the private `@IP` of the end hosts.
```

TTL	**SimNet0**	**SimNet1**	**SimNet2**	**SimNet3**
Outer header	NA	63	62	NA
Inner header	64	63	63	62

The results of the `TTL` can be seen in the top table. Thanks to the option `inherit`, the outer hearder inherits the inner header `TTL` when entering the tunnel and the inner header decrements value for the `TTL` when exiting the tunnel. During transfer, only the outer `TTL` gets modified, the inner remains frozen.

In order to evaluate the value of the `TTL`, we are going to perform several pings increasing the field:

TTL=1

The packet gets droped at the first hop, the `R1`. No outer header gets created because it is not transfered tothe tunnel. The `ICMP` error message is the `time to live exeeded` generated by `R1`. We only see trafic on `SimNet0`.

TTL=2

As before, the ping did not reach the destination. It was lost on the `RS` router.

The `ICMP` message is the same as before but now it's generated by `RS` and only seen on `SimNet1`. This frame has as origin `192.0.2.1` and as destination `192.0.2.2`. The `ICMP` message does not get ralayed backwards to the host (we don't see it on `SimNet0`) because the expired `TTL` belogs to the outer header. We do not see a `soft state` beacuse no message has been sent from the encapsulator to the sender. We do see a behabiour compliant with the `RFC 2003` because no messages have been sent to `host2`.

TTL=3

Now the response is received succesfuly. We do see the `ICMP` reply.

Exercice 6

Section 1

The `MTU` of the tunnel interfaces is `1480` bytes. We assume its because the L2 protocol is ethernet, so the `MTU` is 1500 minus the outer header of `20` bytes. Now we restinc the `MTU` for `SimNet2` to 996 bytes.

After restablishing the tunnel, we can see taht the mtu on `R1` is still `1480` but on the other side now has the expected value of `976`, 20B less that the one set up for the link.

Section 2

So by running `ping -c1 -s500 -M want 172.16.1.3` we set a payload length of 500 bytes, and 1 to the dont fragment flag. This option only allows fragmentatio locally, not in path.

We can see two ICMP frames on each `SimNet`, one for the request and the other one for the reply. The sizes on the private networks are 542B whereas on the tunnel, thanks to the outer header, 562B.

The packets have not been fragmented because they do not exceed the minimum `MTU` (996B) and as expected the `DF` flag is set to 1 on the outer and inner headers of the request.

Section 3

Now we allow fragmentation in path: `ping -c1 -s500 -M dont 172.16.1.3`.

The results are identical with the `DF` flag set to 0 in all headers.

Section 4

Now we execute: `ping -c1 -s500 -M do 172.16.1.3`. This option prohibits all fragmentation, even local one. Again, no diference.

In order to have the maximumsize allowed, we need to send a ping with parameter `-s948` which resultrs from 996-20-20-8.

The -M want option

After clearing the cache, we run `host2:~# ping -c1 -s1000 -M want 172.16.1.3`. The ping is not succesfull.

With want, we only desactivate DF when needed.

We can see the fisrt `ICMP` frame in `SimNet0`, then inside `SimNet1`, it gets droped and notified to `R1` with an `ICMP` frame due to its size.

The lengths are 542 in the private nets and 562 in the transport ones. This length does exceed the minimum `MTU`.

All IP headers except from the `ICMP` error one have DF set to 1 (inner and outer).

The error is originated on `RF` and notified to `R1`. The notified mtu is 996.

In my humble opinion, the destination should be `host2`. It is not acting as expected because an error code `4` should be notified to the sender.

Soft State

After runnin `ping -c2 -s1000 -M want -i1 172.16.1.3` we can see the following output:

```
host2:~#  ping -c2 -s1000 -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 1009ms
```

We can now see an error pop up. If we analize the captured traffic, we can observe that the first `ICMP ping request` message gets discarted on `RS`. Is this router who notifies then `R1` as before. The difference comes with the second frame, this one gets droped at `R1` and is this router who notifies `host2` that the length is exceeding the `MTU`. We can't see the second frame on `SimNet2`.

The DF is set on all `ICMP ping request` frames. We can now conclude that `R1` mantains a soft state of the `MTU`. We can retrieve it by running `ip route show cache`:

```
local 192.0.2.2 from 192.0.2.1 dev lo
src 192.0.2.2
    cache <local,src-direct>  iif eth2
192.168.0.2 dev eth1  src 192.168.0.1
    cache  ipid 0x0b79
172.16.1.3 dev tunnel0  src 192.0.2.2
    cache  expires 179sec ipid 0xc69d
mtu 976
198.51.100.2 from 192.0.2.2 via
192.0.2.1 dev eth2
    cache  expires 178sec ipid 0x107c
mtu 996
```

It reads: in order to arrive to @IP1 from @IP2 via @IP3, this restrictions are aplied. It has an expiration time.

Now, by increasing the count to three, we can see that the third try succeds.

By analising the camptured traffic, we can see taht the third ping does not have the DF activated and comes fragmented from `host2` (MF activated on in the first piece).

SimNet0

The sizes of the fragmnets are 986 and 90 each. (With the ethernet header)

SimNet1

The sizes of the fragmnets are 1010 and 110 each (with the ethernet header). Provided that the ethernet header size is 14B, neigder of these frames exceed the minimum `MTU`.

The fragmentations in `SimNet1` and `SimNet0` are different because in `SimNet1` a 20 byte header must be added. The new outer header must have track of both fragments with the fragment offset field (which now keeps also track of the inner header). In order to have data sizes multiple of 8, four bytes from the second fragment are moved to the first one. This ca be done becaus DF is disabled.

SimNet3

We see the same ICMP requests as in SimNet2.

All the fragmented framges have DF disabled-

The ICMP response fragments fit natively the MTU of the tunnel.

Finally, we run `ping -c2 -s 1460 -M want -i1 172.16.1.3`:

The scond request is the one that succeds. Because it exceeds the MTU of the first link, the `R1` generated the error right away so the next message gets fragmented from the `host2` for a mtu of 1480. This fragmentation puts the DF to 0 so on `RS` the fragments get resized to fit the channel. The respons fits the minimum `MTU` natively.

The -M do option

Now we run `ping -c3 -s 1000 -M do -i1 172.16.1.3`. It does not succed. The packets are sent as:

1. Does get sent and droped at `RC` and notified to `R1` not to `host2`
2. Does get sent and droped at `R1` and notified to `host2` from `R1` (soft state)
3. Does not get sent at all.

So with `do` does not send if he knows that will not succeed.

The -M dont

We set the dont so we will set always the DF to 0. We run `ping -c3 -s 1000 -M dont -i1 172.16.1.3`. All packets succed because they have the

DF flag unset. They get fragmented at the entrace of the 996 mtu link.

The fragmentation in path has less fragmentations needed. It also takes advantage of the full MTU of the first link.

Exercice 7

The -M want

After setting up the scenario, we run the want ping:

1. The fisrt packet get sdroped adt `RC` and notified to `R1` silently.
2. The second on gets droped at `R1` and notified to `host2` with the same mtu as in the exercice before.
3. The third frame gets fragmented and not forwarded to `RC` because pmtudisc does not allow fragmented frames to the tunnel. Its purpose is to avoid fragmentation. Its gets silently discarted.

The -M do

After setting up the sceration we run the do ping:

1. The first packet gets dropet at `RC` as before.
2. The second one behabes as the second one from before.
3. The thirs one does not get even sent by ping because it realizes it will get droped by `R1`.

The outer header of the first packet at `SimNet1` has DF set to 1 (due to the pmtu disc) whereas the inner has it set to 0 (due to the dont on ping).

The -M dont

After setting up the scenario we run the dont ping:

1. The first packet gets droped and notified to `R1` at `RC`.
2. The frame is silently discated at `R1`.
3. The frame is silently discated at `R1`.

They are silently discarted because the notification would be fragmentation needed and the frames are set DF so they can be fragmented.

So the pmtudisc does not allow fragemnted frames or frames that would be be fragmented in the tunnel

Exercice 0.8

The sender will all always use the minimum MSS.

Section 0.8.1

We start teh connection between the `host2` and `host3`. We can see the `SYN`, `SYN/ACK` and the final `ACK` of the three way handshake. The MSS advised at `SYN` from `host2` is 1460. On the other hand the MSS advised by `host3` by `SYN/ACK` is also 1460 because `host3` does not see the tunnel MTU.

Section 0.8.2

Now we are going to try to send a file.

- In the three way handshake the `MSS` was 1460.
- The fisrt data packages from the server are refused by the router due to the mtu. The error is generated on the `R2` adn propagated to the `host3`.
- `host3` is in charge of solvin gthe proble by reducing the frame size.
- The MSS has not changed because it is a local value.
- The file is transmited properly.

Now we do the reverse operation:

```
host2:~# cat /etc/services | nc
192.168.0.2 12345
```

- Now we can see the same error than befor but now on `SimNet0`
- The first ICMP error is generated by `NS` and reported to R1. Then the host sends 3 more frames that will also get discarted but with an ICMP error message from `R1`.
- The file was transmited correctly and the TCP client is `host2`

Section 0.8.3

- The three way hanshake does wokr because the frames are less tha 100B long.
- Yes we have a problem. We see an ICMP error at `SimNet0` reported by R1 because the fisrt data messages execeed the 1480 mtu (1500).
- The next messages silently die at RS because the R1 router does not accept input ICMP messages. The file cannot be transmited.
- If we use the forward chain it would work, because we are not relaying ICMP messages we are using ICMP.

Section 0.8.4

The MSS should be 996-20-20-20=936. We change it on the `host3` side.

Change the MSS in the host

We run the command `host3:~# ip route add 192.168.0.0/24 via 172.16.1.1 advmss 936` And now everything works.

Change the MSS in the router

In order to set up the scenario we run on the R2 `iptables -t mangle -A FORWARD -o tunnel0 -p tcp --syn -j TCPMSS --set-mss 936`.

We use the --syn insted of what we did befor due to the dircetion and the profiles of the hosts. (client and server)

Practice 7 memo of TCGI

Exercice 1
Section 1
After running the default configuration command we can see that the configuration of the mentioned hosts we configure the filtering tables of `host1`. The rules are:

- Block all entering `ICMP` traffic.

We run on `host1`: `iptables -t filter -A INPUT -p ICMP -j DROP`

And now we test the configuration by *pinging* the `host1` from `Rint`.

- We can capture the `echo-request` generated by `Rint`.
- We can't not capture the `echo-reply` because the `echo-request` packet gets droped when it arrives.
- It gets silently droped

Now we perform the reverse operation.

- We can capture both the `echo-request` and the `echo-reply` but the `host1` reports that the package has been lost.
- The cause of this behaviour is the filter. The package is received but silently drop and never gets served to the application.

Section 2
The next step is to remove the before aplied filter by running: `iptables -t filter -D INPUT -p ICMP -j DROP` And we test the configiration with a simple ping. It's time to cnfigure by:

- The `host1` must be able to ping `Rint`.
- The `host1` must no reply any ping request.

In order to configure this set-up we run: `iptables -t filter -A INPUT -p ICMP --icmp-type echo-request -j DROP`.

- When `host1` pings `Rint` we see the normal behaviour.
- When `Rint` pings `hosts` we see the same procedure than before when all ICMP packets where droped.

Section 3
Now we configure the `Rint` router as the *network guard*. We first delete all entries of the filtering tables of `host1` by running `iptables -t filter -D INPUT -p ICMP --icmp-type echo-request -j DROP`. After checking the connection with `Net1` via ping ping we can proceed by setting up the requested escenario with the following commands:

```
Rint:~# iptables -t filter -A FORWARD -
p ICMP -s 192.168.1.0/24 --icmp-type
echo-request -j ACCEPT
Rint:~# iptables -t filter -A FORWARD -
p ICMP --icmp-type echo-request -j DROP
```

Now the ping from `host1` to `www` works but not vice-versa.

In order to setup the `TCP` filtering, we run the following commands:

```
Rint:~# iptables -t filter -A FORWARD -
s 192.168.1.0/24 -p tcp --syn -j ACCEPT
Rint:~# iptables -t filter -A FORWARD -
p tcp --syn -j DROP
```

Now try a `nc` from `host1` to `www` and it works but not vice-versa.

The last configuration is the `UDP`, we add:

```
-A FORWARD -s 172.16.1.5/32 -p udp -j
ACCEPT
-A FORWARD -d 172.16.1.5/32 -p udp -j
ACCEPT
-A FORWARD -p udp -j DROP
```

We try this configuration with the command `dig dns.practnet.tcgi`

This is the final configuration file:

```
Rint:~# cat tables.txt
# Generated by iptables-save v1.4.2 on
Sun Apr 28 16:11:18 2019
*filter
:INPUT ACCEPT [4:514]
:FORWARD ACCEPT [5:396]
:OUTPUT ACCEPT [4:278]
-A FORWARD -s 192.168.1.0/24 -p tcp -m
tcp --tcp-flags FIN,SYN,RST,ACK SYN -j
ACCEPT
-A FORWARD -p tcp -m tcp --tcp-flags
FIN,SYN,RST,ACK SYN -j DROP

-A FORWARD -s 192.168.1.0/24 -p icmp -m
icmp --icmp-type 8 -j ACCEPT
-A FORWARD -p icmp -m icmp --icmp-type
8 -j DROP

-A FORWARD -s 172.16.1.5/32 -p udp -j
ACCEPT
-A FORWARD -d 172.16.1.5/32 -p udp -j
ACCEPT
-A FORWARD -p udp -j DROP
COMMIT
# Completed on Sun Apr 28 16:11:18 2019
```

Exercice 2
The first step is to try to ping the `test` machine from `www`. If we capter the traffic on `SimNet0` we can see that the destiation @IP is correct but the source @IP is private, not 'public'. So this is why we do not receive any reply.

After running `Rbcn:~# iptables -t nat -A POSTROUTING -o eth2 -j SNAT --to 10.0.2.2`, the pting command works like a charm.

Now we want to give `test` acces to the `Net1`. If we run the `ping` from `test` we get the `Network unreachable` error. In order to be able to acces the net, we run `Rbcn:~# iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j DNAT --to 172.16.1.2`