RSE 2021-2022 / Alejandro Albert Casañ

Memoria práctica 2

1.- Let's test with pingall ¿Cuál es el resultado?

```
mininet> sh ovs-ofctl add-flow s1 action=normal
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

El resultado es que todos los nodos son alcanzables desde el resto

2.- Cual es el resultado?

```
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=104.610s, table=0, n_packets=27, n_bytes=1890, actions=NORMAL
```

3.- ¿Que obtienes? ¿Por qué?

```
mininet> sh ovs-ofctl del-flows s1
mininet> sh ovs-ofctl dump-flows s1
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```

Ahora que se han borrado los flujos, se han quedado todos los nodos aislados.

4.- Ejecuta ahora: mininet> h1 ping -c2 h2 y luego mininet> h3 ping -c2 h2 ¿Que obtienes? ¿Hay diferencias… Por qué?

```
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.312 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.053 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1010ms
rtt min/avg/max/mdev = 0.053/0.182/0.312/0.130 ms
mininet> h3 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1026ms
pipe 2
```

Después de ejecutar vemos que h3 está aislado. Esto se debe a que no hemos configurado ninguna regla para el puerto 3 de s1.

5.- Que efecto produce añadir este flow? Prueba con los ping de antes.

```
mininet> sh ovs-ofctl add-flow s1 priority=32768,action=drop
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1034ms

mininet> h3 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1022ms
pipe 2
```

Ahora todos los paquetes que pasan por s1 se rechazan porque la regla que los rechaza (la que acabamos de crear) es la que tiene la máxima prioridad.

6.- Describe, en el documento a entregar, que hacen estas dos lineas de configuracion.

mininet> sh ovs-ofctl add-flow s1
priority=500,dl_type=0x800,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=normal

Esta línea habilita el tráfico en la red

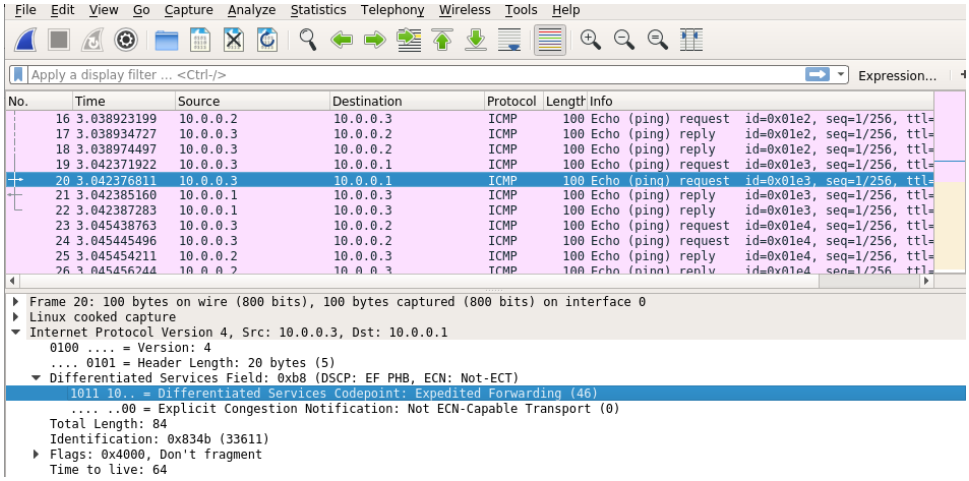
mininet> sh ovs-ofctl add-flow s1
priority=800,dl_type=0x800,nw_src=10.0.0.3,nw_dst=10.0.0.0/24,actions=mod_nw_tos:184,normal

Esta línea modifica el campo TOS de IP por el valor 46

7.- Prueba ahora si funciona con pingall. Comprueba con wireshark si efectivamente se modifica en los paquetes desde h3. Pon una captura de pantalla en el documento a entregar.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Ahora si funciona pingall. Vamos a comprobar en wireshark si se modifican los paquetes desde h3



La modificación se ha producido correctamente.

8.- ¿Que hace exactamente esta última regla?

mininet> sh ovs-ofctl add-flow s1 priority=800,ip,nw_src=10.0.0.3,actions=normal

Esta regla da prioridad a los paquetes que provienen de h3.

9.- Describe la funcion de los diferentes flow

Ejecutando ovs-ofctl dump-flows s1 nos da el siguiente resultado

```
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=73.783s, table=0, n_packets=37, n_bytes=3626, idle_timeout=60, priority=65535,icmp,in_port="s1-eth1",vlan_tci=0x0000
,dl_src=8a:de:bf:a4:cd:22,dl_dst=62:39:e8:d4:f7:d9,nw_src=10.0.0.1,nw_dst=10.0.0.8,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth
3"
 cookie=0x0, duration=73.767s, table=0, n_packets=37, n_bytes=3626, idle_timeout=60, priority=65535,icmp,in_port="s1-eth3",vlan_tci=0x0000
,dl_src=62:39:e8:d4:f7:d9,dl_dst=8a:de:bf:a4:cd:22,nw_src=10.0.0.8,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:"s1-eth
1"
```

cookie=0x0, duration=73.783s, table=0, n_packets=37, n_bytes=3626, idle_timeout=60, priority=65535,icmp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=8a:de:bf:a4:cd:22,dl_dst=62:39:e8:d4:f7:d9,nw_src=10.0.0.1,nw_dst=10.0.0.8,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth3"

cookie=0x0, duration=73.767s, table=0, n_packets=37, n_bytes=3626, idle_timeout=60, priority=65535,icmp,in_port="s1-eth3",vlan_tci=0x0000,dl_src=62:39:e8:d4:f7:d9,dl_dst=8a:de:bf:a4:cd:22,nw_src=10.0.0.8,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:"s1-eth1"

El primer flow es el que monitoriza los ping request que van de h1 a h8 mientras que el segundo flow es el que monitoriza las respuestas a esos pings (que siguen el camino inverso).

10.- Abre el fichero minitopologia.py y describe las secciones de código que encuentras en él.

El contenido del fichero generado en MiniEdit es el siguiente:

root@686c8eeb3236:~# cat minitopologia.py

#!/usr/bin/env Python

#Importamos librerías

from mininet.net import Mininet

from mininet.node import Controller, RemoteController, OVSController

from mininet.node import CPULimitedHost, Host, Node

```
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():
 #Inicializamos la red
    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
#Añadimos los controladores
    c2=net.addController(name='c2',
                         controller=Controller,
                         protocol='tcp',
                         port=6635)

    c0=net.addController(name='c0',
                         controller=Controller,
                         protocol='tcp',
                         port=6633)

    c1=net.addController(name='c1',
                         controller=Controller,
                         protocol='tcp',
                         port=6634)

    info( '*** Add switches\n')
#Añadimos los switches
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch)
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
```

```
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
#Añadimos los hosts
    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h10 = net.addHost('h10', cls=Host, ip='10.0.0.10', defaultRoute=None)
    h6 = net.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)
    h9 = net.addHost('h9', cls=Host, ip='10.0.0.9', defaultRoute=None)
    h8 = net.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)
    h7 = net.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
    h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
#Configuramos los links
    info( '*** Add links\n')
    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s2, h3)
    net.addLink(s2, h4)
    net.addLink(s1, s6)
    net.addLink(s2, s6)
    net.addLink(s6, s8)
    net.addLink(s8, s7)
    net.addLink(s7, s3)
    net.addLink(s3, h5)
    net.addLink(s3, h6)
    net.addLink(s4, h7)
    net.addLink(s4, h8)
    net.addLink(s4, s7)
    net.addLink(s5, s7)
    net.addLink(h9, s5)
    net.addLink(h10, s5)
#Construimos la red
    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
```

```
        controller.start()
    #Ponemos en marcha los switches
    info( '*** Starting switches\n')
    net.get('s7').start([c1])
    net.get('s8').start([c1])
    net.get('s3').start([c2])
    net.get('s4').start([c2])
    net.get('s1').start([c0])
    net.get('s5').start([c2])
    net.get('s2').start([c0])
    net.get('s6').start([c1])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```