

# Trabajo final RDSV

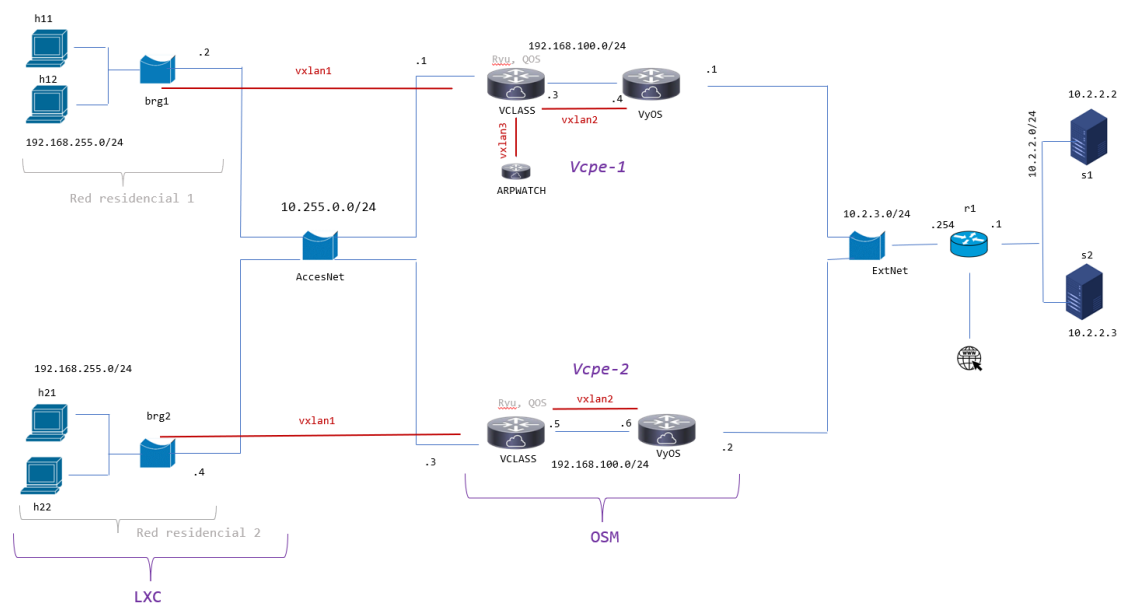
Curso 2021/22 MUIT

Grupo 06: Ángela Burgaleta y Raúl Cabria

## 1. Objetivos

La finalidad de este trabajo es modificar una red residencial virtualizada con OSM, llevada a cabo en la práctica 4. Los cambios efectuados han sido los siguientes:

- Utilizar un Router VyOS virtualizado en lugar de la vcpe
- Añadir soporte de QoS implementado mediante SDN con Ryu
- Añadir conectividad IPv4 desde la red residencial hacia Internet. Con uso de doble NAT en vCPE y en r1
- Sustituir el switch de vclass por un conmutador controlado por OpenFlow
- Añadir una nueva VNF con conexión a Vclass para capturar el tráfico ARP mediante arpmatch
- Gestionar la calidad de servicio en la red de acceso mediante la API REST de Ryu controlando Vclass
- Desplegar dos redes residenciales
- Automatizar el despliegue mediante scripts y OSM



## 2. Desarrollo

### 2.1 Utilizar un Router VyOS como vCPE.

En primer lugar se ha descargado e instalado una imagen de vyos ya creada para utilizarla en el Dockerfile del contenedor. Se ha modificado también el descriptor vcpe.yaml para que utilice la imagen Docker de vnf-vyos.

wget <http://idefix.dit.upm.es/download/vnx/examples/mupi-proxy/vyos-rolling-1.3.tar.gz>

docker image load -i vyos-rolling-1.3.tar.gz

Dockerfile:

```
FROM vyos/rolling:1.3
RUN mkdir /config
CMD /sbin/init
```

### 2.2 Automatización del despliegue del escenario con el script init.sh

Para no tener que realizar todos los pasos de forma manual se ha utilizado un script automatizándolos.

```
#añadir bridge de acceso y bridge externo
sudo ovs-vsctl --if-exists del-br AccessNet
sudo ovs-vsctl --if-exists del-br ExtNet
sudo ovs-vsctl add-br AccessNet
sudo ovs-vsctl add-br ExtNet

#cargar imagen vyos-rolling-1.3
sudo docker image load -i img/vnf-vyos/vyos-rolling-1.3.tar.gz

# Crear imagenes docker
sudo docker build -t vnf-img img/vnf-img
sudo docker build -t vnf-vyos img/vnf-vyos
sudo docker build -t vnf-arpwatch img/vnf-arpwatch

# Subir a OSM
osm vnfd-create pck/vnf-vclass.tar.gz
osm vnfd-create pck/vnf-vcpe.tar.gz
osm vnfd-create pck/arpwatch.tar.gz
osm nsd-create pck/ns-vcpe.tar.gz

# Instanciar vCPE
osm ns-create --ns_name vcpe-1 --nsd_name vCPE --vim_account emu-vim
osm ns-create --ns_name vcpe-2 --nsd_name vCPE --vim_account emu-vim

sleep 30

# Creacion de redes residenciales (contenedores lxc)
sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -t

# Creacion red servers
sudo vnx -f vnx/nfv3_server_lxc_ubuntu64.xml -t
```

## 2.3 Configuración de los túneles VxLAN: brgX-Vclass y Vclass-VyOS

Para el túnel brX-Vclass: Tras haber iniciado el servicio OpenVirtualSwitch en cada VNF, en Vclass se debe agregar un bridge y asociar las interfaces, esto se encuentra automatizado en el script vclass\_start.sh. A este script se le pasa como parámetro (también automatizado con los scripts vcpe-1.sh y vcpe-2.sh) el nombre de la vcpe (en nuestro caso vcpe-1 para la red residencial 1 y vcpe-2 para la red residencial 2) y las direcciones IP para ambas terminaciones del túnel, la que conecta con la red residencial y la que conecta con el Vclass.

```
## 1. En VNF:vclass agregar un bridge y asociar interfaces.
sudo docker exec -it $VNF1 ovs-vsctl add-br br0
sudo docker exec -it $VNF1 ifconfig veth0 $VNFTUNIP/24
sudo docker exec -it $VNF1 ip link add vxlan1 type vxlan id 0 remote $HOMETUNIP dstport 4789 dev veth0
sudo docker exec -it $VNF1 ip link add vxlan2 type vxlan id 1 remote $IP21 dstport 8472 dev $ETH11
sudo docker exec -it $VNF1 ovs-vsctl add-port br0 vxlan1
sudo docker exec -it $VNF1 ovs-vsctl add-port br0 vxlan2
sudo docker exec -it $VNF1 ifconfig vxlan1 up
sudo docker exec -it $VNF1 ifconfig vxlan2 up
```

Para el túnel Vclass-Vyos: Se configura en el script vyos\_start.sh las interfaces ethernet y el túnel vxlan con su dirección, la MTU, el identificador y el puerto. A este script se le deben pasar como parámetros el nombre de la vcpe, para saber a qué red residencial se hace referencia, la IP privada para la vcpe y la dirección ip pública que conectará con la ExtNet. En vyos\_start.sh también configuramos la NAT.

```
sudo docker exec -ti $VNF2 /bin/bash -c "
source /opt/vyatta/etc/functions/script-template
configure
##Configuracion de VyOS
#Interfaz ethernet eth2 -> salida a la extNet
set interfaces ethernet eth2 address '$VCPEPUBIP/24'
set interfaces ethernet eth2 mtu 1400
#Interfaz vxlan vxlan2
set interfaces vxlan vxlan2 address '$VCPEPRIVIP/24'
set interfaces vxlan vxlan2 remote '$IP11'
set interfaces vxlan vxlan2 mtu 1400
set interfaces vxlan vxlan2 port 8472
set interfaces vxlan vxlan2 vni 1
```

```
#Configuracion Nat en vCPE
set nat source rule 100 outbound-interface eth2
set nat source rule 100 source address '192.168.255.0/24'
set nat source rule 100 translation address masquerade
#Configuracion ruta por defecto
set protocols static route 0.0.0.0/0 next-hop 10.2.3.254 distance '1'
set interface ethernet eth0 disable
commit
save
exit
"
```

A continuación, se muestran los scripts comentados anteriormente vcpe-1.sh y vcpe-2.sh con las direcciones IPs. Además accedemos a las máquinas para permitir que se configure la dirección por dhcp.

```
#!/bin/bash
./vclass_start.sh vcpe-1 10.255.0.1 10.255.0.2
./vyos_start.sh vcpe-1 192.168.255.1 10.2.3.1
sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -x dhclient-h11
sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -x dhclient-h12

#!/bin/bash
./vclass_start.sh vcpe-2 10.255.0.3 10.255.0.4
./vyos_start.sh vcpe-2 192.168.255.2 10.2.3.2
sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -x dhclient-h21
sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -x dhclient-h22
```

## 2.4 Asignación de direcciones por DHCP

En el script vyos\_start.sh se realiza la configuración DHCP para los hosts h11, h12, h21 y h22.

```
#Configuración DHCP
set service dhcp-server shared-network-name 'LAN1' authoritative
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 default-router '$VCPEPRIVIP'
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 dns-server '$VCPEPRIVIP'
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 domain-name 'vyos.net'
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 lease '86400'
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 range 0 start '192.168.255.9'
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 range 0 stop '192.168.255.254'
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 static-mapping DHCP-H11 ip-address 192.168.255.5
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 static-mapping DHCP-H11 mac-address 00:00:00:00:01:01
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 static-mapping DHCP-H12 ip-address 192.168.255.6
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 static-mapping DHCP-H12 mac-address 00:00:00:00:01:02
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 static-mapping DHCP-H21 ip-address 192.168.255.7
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 static-mapping DHCP-H21 mac-address 00:00:00:00:02:01
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 static-mapping DHCP-H22 ip-address 192.168.255.8
set service dhcp-server shared-network-name 'LAN1' subnet 192.168.255.0/24 static-mapping DHCP-H22 mac-address 00:00:00:00:02:02
```

## 2.5 Añadir una nueva VNF para capturar el tráfico mediante arpwatrch

Se ha añadido una nueva VNF con un túnel vxlan a Vclass que permita capturar el tráfico arp. Para ello el primer paso ha sido construir una imagen Docker vnf-arpwatch, el Dockerfile que se ha empleado ha sido el siguiente.

```
FROM ubuntu:xenial
RUN apt update
RUN apt-get install -y arpwatrch net-tools vim nano tcpdump
RUN apt-get install -y \
    git \
    net-tools \
    aptitude \
    build-essential \
    python-setuptools \
    python-dev \
    python3-pip \
    software-properties-common \
    ansible \
    curl \
    iptables \
    iputils-ping \
    sudo \
    bridge-utils
```

En el descriptor arpwatch.yaml para generar la instancia de la nueva VNF en el OSM la imagen que se especifica es la que se ha creado con el Dockerfile anterior, se muestra en el campo image:vnf-arpwatch. Por otro lado, también es necesario modificar el descriptor de la NS para añadir otro miembro más.

```
vnfd-catalog:
  vnfd:
    - connection-point:
        - name: eth1
          type: VPORT
        description: Residential traffic classifier
        id: arpwatch
        mgmt-interface:
          cp: eth1
        name: arpwatch
        short-name: arpwatch
        vdu:
          - count: '1'
            description: vCPE Docker-based container
            id: ubuntu
            image: vnf-arpwatch
            interface:
              - external-connection-point-ref: eth1
                name: eth1
                position: 0
                type: EXTERNAL
              virtual-interface:
                type: VIRTIO
            name: ubuntu
            vm-flavor:
              memory-mb: 512
              storage-gb: 10
              vcpu-count: 1
            vendor: UPM
            version: '0.1'
```

```
nsd-catalog:
  nsd:
    - constituent-vnfd:
        - member-vnf-index: '1'
          vnfd-id-ref: vclass
        - member-vnf-index: '2'
          vnfd-id-ref: vcpe
        - member-vnf-index: '3'
          vnfd-id-ref: arpwatch
        description: Residential Network Service (RENES) with three VNFs.
        id: vCPE
        name: vCPE
        short-name: vCPE
        vendor: UPM
        version: '1.0'
        vld:
          - description: DATA VL
            id: data_vl
            mgmt-network: 'true'
            name: data_vl
            short-name: data_vl
            type: ELAN
            vendor: Universidad Politecnica de Madrid
            version: '0.1'
            vim-network-name: default
            vnfd-connection-point-ref:
              - member-vnf-index-ref: '1'
                vnfd-connection-point-ref: eth1
                vnfd-id-ref: vclass
              - member-vnf-index-ref: '2'
                vnfd-connection-point-ref: eth1
                vnfd-id-ref: vcpe
              - member-vnf-index-ref: '3'
                vnfd-connection-point-ref: eth1
                vnfd-id-ref: arpwatch
```

Finalmente, se levanta el túnel vxlan3 que conecta la VNF del arpwatch con el Vclass en el script vclass\_start.sh

```
# Configurar tuneles arpwatch
sudo docker exec -it $VNF1 ip link add vxlan3 type vxlan id 2 remote $IP31 dstport 8472 dev $ETH11
sudo docker exec -it $VNF1 ovs-vsctl add-port br0 vxlan3
sudo docker exec -it $VNF1 ifconfig vxlan3 up

sudo docker exec -it $VNF3 ip link add vxlan3 type vxlan id 2 remote $IP11 dstport 8472
sudo docker exec -it $VNF3 ip link set vxlan3 up
```

## 2.6 Ryu: controlar la calidad de servicio

Para controlar la QOS mediante la VNF Vclass, en primer lugar, ha sido necesario modificar el Dockerfile del vnf-img para añadir los paquetes ryu-bin e iproute2. También se cambia la imagen de ubuntu:xenial a ubuntu:bionic.

Para sustituir el switch por un conmutador OpenFlow que posteriormente se conecte a Ryu se realiza una configuración por comandos ovs-vsctl con el bridge0. En primer lugar, se habilitan los protocolos OpenFlow 10, 12 y 13. Se habilita el modo secure por si tenemos fallos de conectividad. Finalmente se configura el datapath y el controller de br0 con dirección 127.0.0.1

y puerto 6633 (correspondiente a Ryu). El comando ryu-manager que se muestra en la primera línea sirve para conectar el controlador Ryu con el switch.

```
sudo docker exec -d $VNF1 ryu-manager ryu.app.rest_qos ./qos_simple_switch_13.py ryu.app.rest_conf_switch
sleep 10

sudo docker exec -it $VNF1 ovs-vsctl set bridge br0 protocols=OpenFlow10,OpenFlow12,OpenFlow13
sudo docker exec -it $VNF1 ovs-vsctl set-fail-mode br0 secure
sudo docker exec -it $VNF1 ovs-vsctl set-manager ptcp:6632
sudo docker exec -it $VNF1 ovs-vsctl set-controller br0 tcp:127.0.0.1:6633
sudo docker exec -it $VNF1 ovs-vsctl set bridge br0 other-config:datapath-id=0000000000000001

sleep 10

sudo docker exec -it $VNF1 curl -X PUT -d '{"tcp:127.0.0.1:6632"}' http://localhost:8080/v1.0/conf/switches/0000000000000001/ovsdb_addr
```

Las reglas configuradas han sido las siguientes:

- Red residencial: 12Mbps de bajada
- Para hX1: 8Mbps mínimo de bajada
- Para hX2: 4Mbps máximo de bajada

Se incorporan las siguientes sentencias al script qos.sh:

```
sudo docker exec -it $VNF1 curl -X POST -d '{"port_name": "vxlan1", "type": "linux-htb", "max_rate": "12000000", "queues": [{"min_rate": "8000000"}, {"max_rate": "4000000"}]}' http://localhost:8080/qos/queue/0000000000000001

if [ $1 = "vcpe-1" ]; then
    #IP1= sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -x get-h11-ip | grep 192.168.255.7
    #IP2= sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -x get-h12-ip | grep 192.168.255.7
    echo -e "\n-----"
    echo -e "Configurando la qos de la red residencial 1"
    echo -e "-----"
    sudo docker exec -it $VNF1 curl -X POST -d '{"match": {"nw_dst": "192.168.255.5"}, "actions":{"queue": "0"}}' http://localhost:8080/qos/rules/0000000000000001
    sudo docker exec -it $VNF1 curl -X POST -d '{"match": {"nw_dst": "192.168.255.6"}, "actions":{"queue": "1"}}' http://localhost:8080/qos/rules/0000000000000001
elif [ $1 = "vcpe-2" ]; then
    #IP1= sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -x get-h21-ip | grep 192.168.255.7
    #IP2= sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -x get-h22-ip | grep 192.168.255.7
    echo -e "\n-----"
    echo -e "Configurando la qos de la red residencial 2"
    echo -e "-----"
    sudo docker exec -it $VNF1 curl -X POST -d '{"match": {"nw_dst": "192.168.255.7"}, "actions":{"queue": "0"}}' http://localhost:8080/qos/rules/0000000000000001
    sudo docker exec -it $VNF1 curl -X POST -d '{"match": {"nw_dst": "192.168.255.8"}, "actions":{"queue": "1"}}' http://localhost:8080/qos/rules/0000000000000001
```

### 3. Comandos puesta en marcha del escenario y test

#### 1. Descarga e instalación de la máquina virtual:

```
/lab/rdsb/bin/rdsb-get-and-install-vnxsdnfvlab
```

#### 2. Despliegue del escenario dentro de la carpeta del proyecto:

```
./init.sh
```

#### 3. Túneles vxlan:

```
./vcpe-1.sh
```

```
./vcpe-2.sh
```

#### 4. Calidad de servicio:

```
./qos.sh vcpe-1
```

```
./qos.sh vcpe-2
```

#### 5. Destruir escenario:

```
./stop-vnx.sh
```

```
./destroy.sh vcpe-1
```

```
./destroy.sh vcpe-2
```

#### 6. COMANDOS ÚTILES

*Ver contenedores Docker activos:*

```
Docker ps -a
```

*Acceder al root de un contenedor:*

```
Docker exec -ti [id] /bin/bash
```

#### ARPWATCH

```
Docker exec -ti [id] /bin/bash
```

```
Touch /var/lib/arpwatch/arp.dat
```

```
Service arpwatch start
```

*Ping de h11 a broadcast (mandamus a una dirección IP del rango que no se asigne a ningún equipo):*

```
Ping -c5 192.168.255.53
```

```
cat /var/lib/arpwatch/arp.dat
```

```
arpwatch -i vxlan3 -d./
```

#### PRUEBAS DE QOS

*Para el tráfico de bajada:*

Desde la consola de vyos iperf a h11:

```
docker exec -ti [id-vyos] bash -c 'su - vyos'
iperf -c 192.168.255.5 -b 14M -l 1200
```

Desde h11 (escucha):

```
iperf -s -i 1
```

Para el tráfico de subida:

Desde s11 (escucha):

```
Iperf -s -i 1
```

Desde h11 (manda a s11):

```
iperf -c 10.2.2.2 -b 14M -l 1200
```

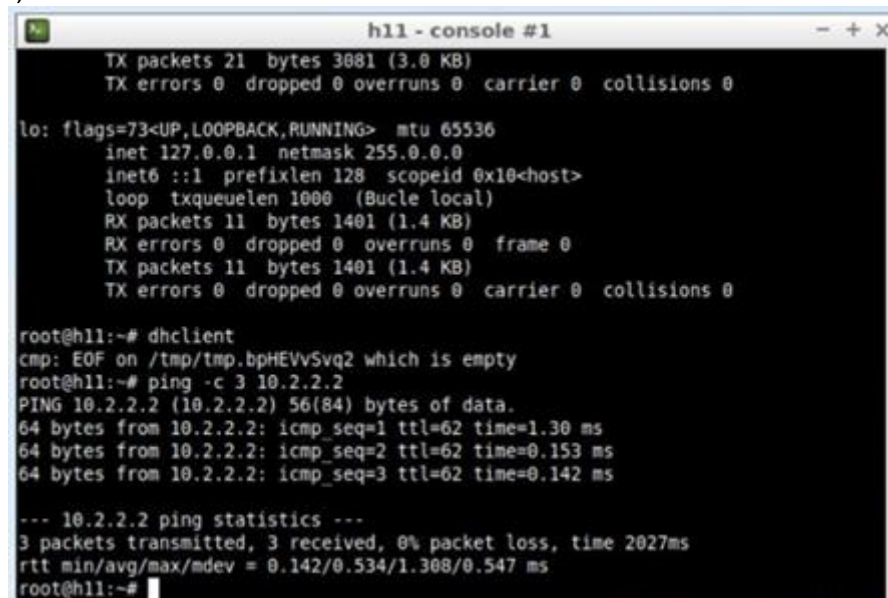
## 4. Pruebas de funcionamiento

Una vez ya tenemos creado el escenario podemos probar si se ha realizado según las especificaciones del proyecto. Se han realizado tres pruebas, listadas a continuación:

- Prueba de conexión entre host y servidores
- Funcionamiento de la calidad de servicio mediante iperf
- Funcionamiento de arpwat

### 4.1 Prueba de conexión

En las siguientes imágenes vemos como desde h11, con dirección IP 192.168.255.5, de la red residencial 1, se realiza un ping a la dirección IP 10.2.2.2, que corresponde a uno de los servidores, s1 concretamente.



```
h11 - console #1
TX packets 21 bytes 3081 (3.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Bucle local)
RX packets 11 bytes 1401 (1.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 11 bytes 1401 (1.4 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@h11:~# dhclient
cmp: EOF on /tmp/tmp.bpHEVvSvq2 which is empty
root@h11:~# ping -c 3 10.2.2.2
PING 10.2.2.2 (10.2.2.2) 56(84) bytes of data:
64 bytes from 10.2.2.2: icmp_seq=1 ttl=62 time=1.30 ms
64 bytes from 10.2.2.2: icmp_seq=2 ttl=62 time=0.153 ms
64 bytes from 10.2.2.2: icmp_seq=3 ttl=62 time=0.142 ms

--- 10.2.2.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.142/0.534/1.308/0.547 ms
root@h11:~#
```



```

s1 - console #1
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
root@s1:~# ifconfig
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.2.2.2 netmask 255.255.255.0 broadcast 10.2.2.255
    inet6 fe80::fd:ff:fe04:101 prefixlen 64 scopeid 0x20<link>
    ether 02:fd:00:04:01:01 txqueuelen 1000 (Ethernet)
    RX packets 70 bytes 9505 (9.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 24 bytes 3312 (3.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Bucle local)
    RX packets 20 bytes 1984 (1.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 20 bytes 1984 (1.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@s1:~#

```

## 4.2 Prueba de Qos

Para realizar estas pruebas nos ayudaremos de la herramienta iperf, usada anteriormente en las prácticas de la asignatura.

Si deseamos comprobar la QoS en el sentido de bajada, entonces es necesario generar el tráfico desde el VyOS y escuchar ese tráfico con h11, por ejemplo. Para ello ejecutaremos los siguientes comandos:

```

root@h11:~# iperf -s -i 1
vyos@vyos:~# iperf -c 192.168.255.5 -b 14m -l 1200

```

Con esto generaremos un tráfico de 14Mbps de s1 a h11, y veremos como el límite de 12Mbps que hemos creado para la red residencial no permitirá que el ancho de banda exceda de ese valor.

```

vyos@vyos:~# ifconfig
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.2.2.2 netmask 255.255.255.0 broadcast 10.2.2.255
    inet6 fe80::fd:ff:fe04:101 prefixlen 64 scopeid 0x20<link>
    ether 02:fd:00:04:01:01 txqueuelen 1000 (Ethernet)
    RX packets 70 bytes 9505 (9.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 24 bytes 3312 (3.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Bucle local)
    RX packets 20 bytes 1984 (1.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 20 bytes 1984 (1.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vyos@vyos:~# iperf -s -i 1
iperf: ignoring extra argument -- 14m
Client connecting to 192.168.255.5, TCP port 5001
TCP window size: 45.0 KByte (default)
[ 0] local 192.168.255.1 port 5001 connected with 192.168.255.5 port 5001
[ ID] Interval Transfer Bandwidth
[ 0] 0.0-10.1 sec 14.2 MBytes 11.8 Mbits/sec
vyos@vyos:~# iperf -c 192.168.255.5 -b 14m -l 1200
iperf: ignoring extra argument -- 14m
Client connecting to 192.168.255.5, TCP port 5001
TCP window size: 45.0 KByte (default)
[ 0] local 192.168.255.1 port 5001 connected with 192.168.255.5 port 5001
[ ID] Interval Transfer Bandwidth
[ 0] 0.0-10.3 sec 5.42 MBytes 4.43 Mbits/sec
vyos@vyos:~#

```

En el caso de h12 vemos como ese límite se encuentra en 4Mbps.

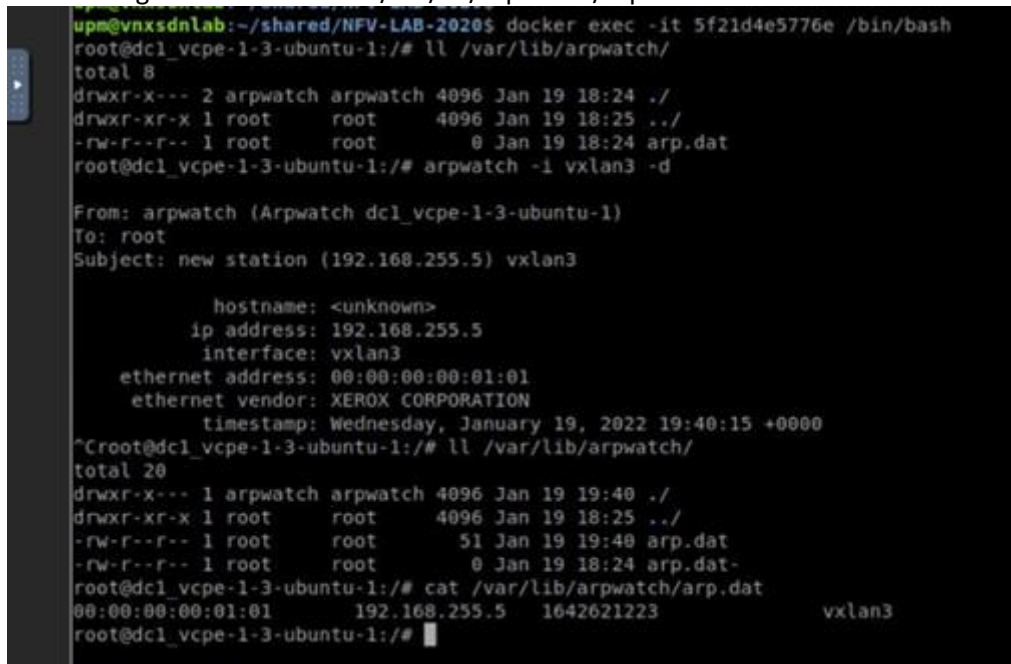
### 3.2.3 Prueba de arpwatrch

La prueba de arpwatrch consiste en entrar en el contenedor que hemos creado para ello y ejecutar el siguiente comando:

```
arpwatch -i vxlan3 -d
```

Mientras tanto, tendremos a un host, por ejemplo, h11, realizando un ping constante a una dirección IP de la red residencia que no exista, por ejemplo, a la 192.168.255.50.

Con el comando anterior lo que veremos es lo que se muestra en la siguiente imagen. Saldrá un mensaje del host que manda mensajes ARP junto con otra información útil. Además, esta información se guarda en el fichero /var/lib/arpwatch/arp.dat.



```
upm@vnxsdnlab:~/shared/NFV-LAB-2020$ docker exec -it 5f21d4e5776e /bin/bash
root@dcl_vcpe-1-3-ubuntu-1:/# ll /var/lib/arpwatch/
total 8
drwxr-x--- 2 arpwatch arpwatch 4096 Jan 19 18:24 ./
drwxr-xr-x 1 root      root      4096 Jan 19 18:25 ../
-rw-r--r-- 1 root      root        0 Jan 19 18:24 arp.dat
root@dcl_vcpe-1-3-ubuntu-1:/# arpwatrch -i vxlan3 -d

From: arpwatch (Arpwatch dcl_vcpe-1-3-ubuntu-1)
To: root
Subject: new station (192.168.255.5) vxlan3

        hostname: <unknown>
        ip address: 192.168.255.5
        interface: vxlan3
        ethernet address: 00:00:00:00:01:01
        ethernet vendor: XEROX CORPORATION
        timestamp: Wednesday, January 19, 2022 19:40:15 +0000
^Croot@dcl_vcpe-1-3-ubuntu-1:/# ll /var/lib/arpwatch/
total 20
drwxr-x--- 1 arpwatch arpwatch 4096 Jan 19 19:40 ./
drwxr-xr-x 1 root      root      4096 Jan 19 18:25 ../
-rw-r--r-- 1 root      root       51 Jan 19 19:40 arp.dat
-rw-r--r-- 1 root      root        0 Jan 19 18:24 arp.dat-
root@dcl_vcpe-1-3-ubuntu-1:/# cat /var/lib/arpwatch/arp.dat
00:00:00:00:01:01      192.168.255.5    1642621223      vxlan3
root@dcl_vcpe-1-3-ubuntu-1:/#
```

