Memoria Trabajo Final RDSV

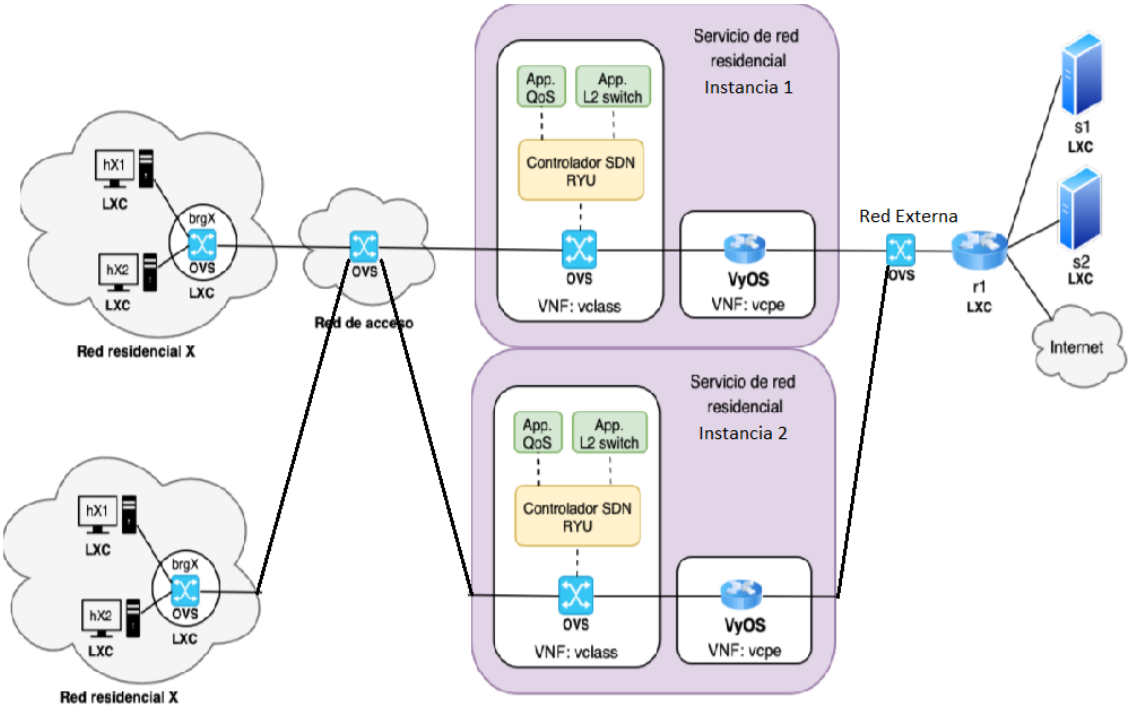
Ramiro Miralles Penalva, José Luis Mendoza Sánchez, MUIRST

1- Introducción

El objetivo del trabajo final de esta asignatura va a consistir en la transformación del escenario de la práctica 4, compuesto por:

- Dos Redes Residenciales.
- Red de Acceso representada por un switch.
- Central Local Definida por Software una NS por cada red residencial y 2 VNF por cada NS.
- Red Externa representada por un switch.
- Router de Salida (r1).
- Servidor Externo (s1).

De tal forma que se provea a los usuarios de los servicios de DHCP, NAT y QoS mediante un router virtualizado VyOS en vcpe y la aplicación de un controlador RYU al Open VSwitch de vclass que gestiona al OVS alojado en el mismo contenedor. La topología que representa dicho escenario es la siguiente:



Esta memoria trata de detallar los pasos que hemos seguido para modificar la práctica 4 hasta conseguir el escenario comentado.

2- Onboarding de los descriptores de NS y VNF e inicialización de estos en OSMano mediante línea de comandos.

Dado que íbamos a tener que re-arrancar el escenario múltiples veces, lo primero que aprendimos fue a hacer el onboarding de los descriptores de NS y VNF y su inicialización mediante línea de comandos. Así, los comandos que utilizamos fueron:

```
# Onboarding de los descriptores
osm vnfd-create pck/vnf-ryu.tar.gz
osm vnfd-create pck/vnf-vyos.tar.gz
osm nsd-create pck/ns-vcpe.tar.gz

# Instanciación
osm ns-create --ns_name vcpe-1 --nsd_name vCPE --vim_account emu-vim
osm ns-create --ns_name vcpe-1 --nsd_name vCPE --vim_account emu-vim
```

3- Router VyOS

La imagen de docker “vnf-vcpe” no nos servía para el propósito del router VyOS y por ello fue necesario reemplazarla por otra imagen que sí que lo incluyera. Dicha nueva imagen tendría un Dockerfile de la siguiente forma:

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

Una vez construida la imagen, decidimos llevárnosla a DockerHub para que las siguientes veces fuera más sencillo (https://hub.docker.com/repository/docker/jlmendo11/vnf-vyos). Tuvimos también que cambiar el descriptor de VNF para que ahora se nutriera de la imagen “vnf-vyos” en lugar de la anterior.

Ahora ya podremos acceder al vcpe y disponer de VyOS, por lo que comenzaremos a configurarlo para que proporcione los servicios DHCP, NAT y el túnel VXLAN con la red residencial. Estas fueron las configuraciones que tuvimos que hacer:

Túnel VXLAN

Para poder ofrecer DHCP y NAT a los hosts de la red residencial, es necesario primero conseguir establecer una comunicación con ellos, con este fin habrá que establecer el túnel VXLAN.

```
Configure
set interfaces vxlan vxlan2 address $VCPEPRIVIP/24
set interfaces vxlan vxlan2 remote $IP11
```

```
set interfaces vxlan vxlan2 mtu 1400
set interfaces vxlan vxlan2 vni 1
set interfaces vxlan vxlan2 port 4789
commit
save
```

En términos generales, lo que hace el código es:

- Fijar la dirección IP que tendrá vcpe en está VXLAN.
- Especificar el enlace a través del cual irá la VXLAN, mediante la IP de siguiente salto.
- Especificar el MTU, el identificador de VXLAN (VNI) y el puerto por el que se comunicarán.

DHCP

Una vez ya es posible el envío de paquetes entre el vcpe y la red residencial, procedemos a activar el servicio DHCP que proporcionará una dirección IP a los hosts.

```
configure
set service dhcp-server shared-network-name 'RESIDENCIAL'
authoritative
set service dhcp-server shared-network-name 'RESIDENCIAL' subnet
'192.168.255.0/24' range 0 start '192.168.255.20'
set service dhcp-server shared-network-name 'RESIDENCIAL' subnet
'192.168.255.0/24' range 0 stop '192.168.255.30'
set service dhcp-server shared-network-name 'RESIDENCIAL' subnet
'192.168.255.0/24' default-router '192.168.255.1'
set service dhcp-server shared-network-name 'RESIDENCIAL' subnet
'192.168.255.0/24' lease 86400
set service dhcp-server shared-network-name 'RESIDENCIAL' subnet
'192.168.255.0/24' dns-server '8.8.8.8'
set service dhcp-server shared-network-name 'RESIDENCIAL' subnet
'192.168.255.0/24' domain-name 'example.org'
commit
save
```

El código anterior realiza las siguientes funciones:

- Define nuestro DHCP como autoritario (envía NACK a aquellos clientes que envíen un REQUEST no válido).
- Establece el rango en el que se encontrarán las IP prestadas.
- Establece el router Gateway.
- Especifica el tiempo que los clientes mantienen la IP, el servidor DNS y el dominio.

NAT

Como último paso, en el Vyos se configurará el NAT para permitir a los hosts la comunicación con el exterior a través de una única dirección pública.

```
configure
set nat source rule 50 outbound-interface eth2
set nat source rule 50 source address 192.168.255.0/24
set nat source rule 50 translation address masquerade
commit
save
```

El código es bastante simple, tan solo definimos cuál será la interfaz de salida, qué direcciones IP serán tomadas como entrada y el tipo de NAT que se usará, en este caso NAT Overload.

4- Controlador RYU

De forma muy similar a VyOS, lo primero que tuvimos que hacer fue crear otra imagen porque la que teníamos anteriormente no nos servía. Dicha nueva imagen se basaba en un Dockerfile que tenía el siguiente aspecto:

```
FROM ubuntu:bionic

# fix timezone
ENV TZ=Europe/Madrid
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > 
/etc/timezone

# install required packages
RUN apt-get clean
RUN apt-get update \
    && apt-get install -y python-dev […] iproute2 ryu-bin

# copy ryu file
COPY qos_simple_switch_13.py /qos_simple_switch_13.py

# open ports
EXPOSE 5201
EXPOSE 3000
EXPOSE 7000
EXPOSE 6633
```

Donde "qos_simple_switch_13.py" es el archivo de ryu que incluye las modificaciones propuestas en: http://osrg.github.io/ryu-book/en/html/rest_qos.html.

De esta forma, al instanciar la VNF ya disponemos de un contenedor en la central local con los componentes necesarios para realizar la configuración. Para encender el ryu, hay que escribir:

```
ryu-manager ryu.app.rest_qos ryu.app.rest_conf_switch
./qos_simple_switch_13.py
```

Y a continuación vamos a ver cómo se configura:

Conexión RYU y Open VSwitch

Antes de comenzar con cualquier tipo de operación que nos ofrece la API de RYU, será necesario establecer la conexión entre el controlador y el switch OpenFlow.

```
ovs-vsctl set bridge br0 other-config:hwaddr=\"00:00:00:00:12:34\"
ovs-vsctl set bridge br0 protocols=OpenFlow13
ovs-vsctl set-manager ptcp:6632
ovs-vsctl set-controller br0 tcp:127.0.0.1:6633
```

Para el bridge br0 del OVS: cambiamos su dirección MAC, seleccionamos la versión de OpenFlow (1.3), establecemos un socket de escucha en el 6632 conectado al manager de la OVSDB y establecemos al ryu como controlador refiriéndonos a su puerto 6633.

Dada la dirección MAC “00:00:00:00:12:34”, el ryu identificará al switch con la cadena “0000000000001234”.

QoS desde la API

Ahora que tanto RYU como el OVS han sido inicializados, ya vamos a poder comunicarnos con la API del controlador para ejecutar las órdenes que se deseen. En este caso, establecer QoS tal y como se nos requiere se consigue con las siguientes peticiones “curl”:

```
curl -X PUT -d '\"tcp:127.0.0.1:6632\"'
http://127.0.0.1:8080/v1.0/conf/switches/0000000000001234/ovsdb_addr

curl -X POST -d '{\"port_name\": \"vxlan1\", \"type\": \"linux-
htb\", \"max_rate\": \"12000000\", \"queues\": [{\"min_rate\":
\"8000000\"}, {\"max_rate\": \"4000000\"}]}'
http://127.0.0.1:8080/qos/queue/0000000000001234

curl -X POST -d '{\"match\": {\"nw_dst\": \"192.168.255.20\"},
\"actions\":{\"queue\": \"0\"}}'
http://127.0.0.1:8080/qos/rules/0000000000001234

curl -X POST -d '{\"match\": {\"nw_dst\": \"192.168.255.21\"},
\"actions\":{\"queue\": \"1\"}}'
http://127.0.0.1:8080/qos/rules/0000000000001234
```

Las peticiones que se han enviado realizan las siguientes funciones:

- Creación en el controlador de un documento que indica la dirección del OVS.
- Creación de las dos colas.
- Definición de a quién se le aplicará la cola 0.
- Definición de a quién se le aplicará la cola 1.

5- QoS de subida [Apartado opcional]

La calidad de servicio que hemos configurado en el apartado anterior restringe el trafico que entra a la red residencial (tráfico de bajada). Como apartado opcional, se propuso en la práctica realizar una configuración de la QoS del tráfico saliente de la red residencial (tráfico de subida), conectando el switch brg1 y brg2. Para conseguirlo, aprovechamos el mismo ryu que se aloja en vClass. Siguiendo los mismos pasos del apartado anterior:

Conexión RYU y Open VSwitch

Lo primero que hay que conseguir es establecer una comunicación entre ambos nodos. El principal problema que se presenta en este caso es que se debe cursar la comunicación entre el túnel vxlan. Este túnel debe estar creado a través de Linux y no a través del OpenVSwitch. Es por ello por lo que hay que cambiar la configuración de brg1 (y de brg2) en el escenario VNX por las siguientes:

```
<exec seq="on_boot" type="verbatim">
    service openvswitch-switch start
    sleep 5
    ovs-vsctl add-br br0
    ovs-vsctl add-port br0 eth1
    ip link add vxlan1 type vxlan id 0 remote 10.255.0.1 dstport 4789 dev eth2
    ovs-vsctl add-port br0 vxlan1
    ifconfig vxlan1 up
</exec>
```

Con este código conseguimos que el funcionamiento de momento sea completamente igual que el que hemos visto hasta ahora pero con el túnel vxlan definido en el mismo contenedor LXC de brg1. Pero aún no hemos conectado con el ryu, para ello hay que definir otra etiqueta (o ponerla en el mismo “on_boot”) que podamos ejecutar más tarde:

```
<exec seq="qos-opcional" type="verbatim">
    ovs-vsctl set bridge br0 other-config:hwaddr=\"00:00:00:00:66:66\"
    ovs-vsctl set bridge br0 protocols=OpenFlow13
    ovs-vsctl set-manager ptcp:6632
    ovs-vsctl set-controller br0 tcp:10.255.0.1:6633
</exec>
```

Como vemos, las líneas de código son las mismas que hemos visto en el apartado anterior. Cuando está puesto, para levantar la QoS de subida, habrá que hacer:

```
sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -v --execute qos-opcional
```

QoS desde la API

```
curl -X PUT -d '\"tcp:10.255.0.2:6632\"'
http://127.0.0.1:8080/v1.0/conf/switches/0000000000006666/ovsdb_addr

curl -X POST -d '{\"port_name\": \"vxlan1\", \"type\": \"linux-htb\", \"max_rate\":
\"6000000\", \"queues\": [{\"min_rate\": \"2000000\"}, {\"max_rate\":
\"2000000\"}]}' http://127.0.0.1:8080/qos/queue/0000000000006666
```

```
curl -X POST -d '{\"match\": {\"nw_src\": \"192.168.255.20\"},
\"actions\":{\"queue\": \"0\"}}' http://127.0.0.1:8080/qos/rules/0000000000006666

curl -X POST -d '{\"match\": {\"nw_src\": \"192.168.255.21\"},
\"actions\":{\"queue\": \"1\"}}' http://127.0.0.1:8080/qos/rules/0000000000006666
```