Redes definidas por Software, Virtualización y Servicios Avanzados de Red

Máster Universitario en Ingeniería de Telecomunicación

Máster Universitario en Ingeniería de Redes y Servicios Telemáticos

Curso 2022/2023

Trabajo final

Miguel Ángel Cuesta Bravo

Laura Fernández Galindo

Jose Javier Mata de la Fuente

Escuela Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Madrid

Descripción de la práctica

1. Descripción de la práctica

En esta sección se trata de dar una vista general del proyecto, tanto en contenido como en desarrollo. En primer lugar, el servicio de red bajo estudio es el servicio residencial de acceso a Internet, donde el router residencial se sustituye por un "*Bridged Residential Gateway (BRG)*" que realiza la conmutación de nivel 2 del tráfico de los usuarios entre la red residencial y la central local. El resto de las funciones se realizan en la central local aplicando técnicas de virtualización de red (NFV), creando un servicio de CPE virtual (vCPE) gestionado mediante la plataforma de orquestación.

Para su implementación se ha utilizado la imagen virtual RDSV2022-v1.ova, la cual virtualiza RDSV-K8S, que nos permite utilizar el paquete microk8s, la herramienta VNX, Open vSwitch (ovs); y RDSV-OSM, que instala el entorno OSM, al que se accede gráficamente.

2. Requisitos

En esta práctica se parte del trabajo previo realizado en la práctica 4. Su finalidad consiste en la modificación de una red residencial virtualizada con la plataforma de código abierto Open Source Mano (OSM). Los cambios que hemos llevado a cabo han sido los siguientes:

- Sustituir el switch de KNF:access por un conmutador controlado por OpenFlow.
- Conectividad IPv4 desde la red residencial hacia Internet. Uso de doble NAT: en KNF:cpe y en isp1.
- Activar la captura de tráfico ARP mediante "arpwatch".
- Gestión de la calidad de servicio en la red de acceso mediante la API REST de Ryu controlando KNF:Access, para limitar el ancho de banda de bajada hacia la red residencial.
- Despliegue para dos redes residenciales.
- Automatización del despliegue mediante OSM y scripts.

Además, otros requisitos opcionales implementados son:

- Sustituir el switch de brgX por un conmutador controlado por OpenFlow desde el Ryu, incluyendo la gestión de la calidad de servicio desde el Ryu instalado en KNF:access, controlando el brgX, para limitar el ancho de banda de subida desde la red residencial

3. Arquitectura del escenario

La imagen siguiente ilustra el escenario inicial de la práctica:

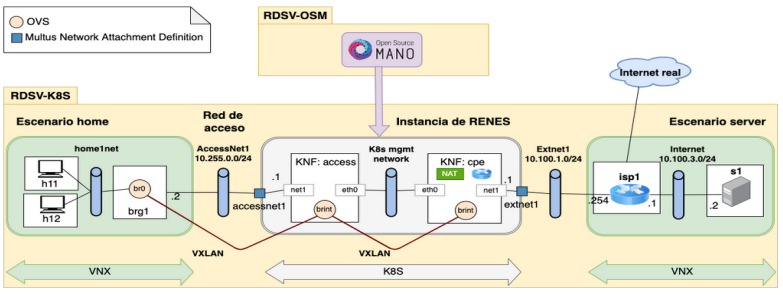


Figura 1. Escenario inicial

Como se refleja en la figura, se utilizará la tecnología VXLAN para enviar encapsuladas en datagramas UDP las tramas de nivel 2 que viajan entre brg1, KNF:access y KNF:cpe. Para permitir esta comunicación, tanto el brg1 como KNF:access tendrán interfaces en AccessNet1, configuradas con direcciones IP del prefijo 10.255.0.0/24. La asignación de direcciones IP a KNF:access y KNF:cpe en la red que las interconecta está gestionada por OSM y k8s, de manera que se asignan dinámicamente al instanciar las KNFs.

El escenario que se pide desarrollar presenta la siguiente estructura:

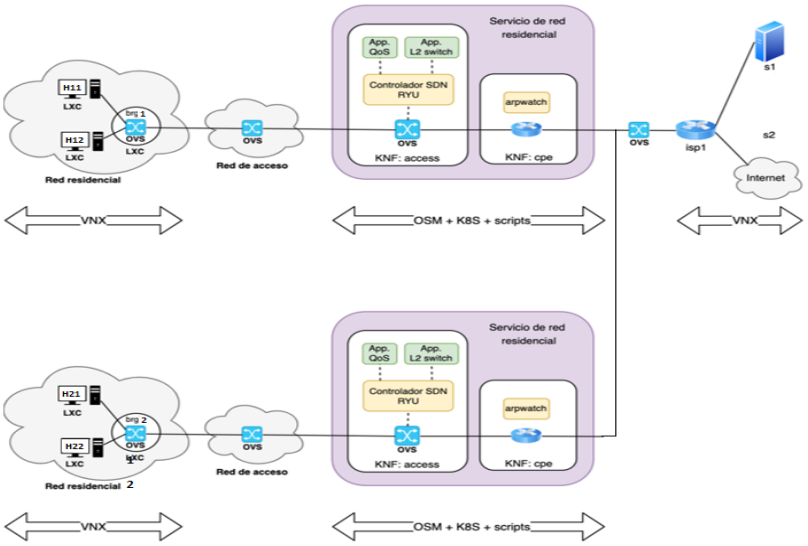


Figura 2. Escenario Final

Se desplegarán dos redes residenciales como se muestra en la imagen. Estas presentan una conectividad IPv4 desde cada una de las redes residenciales hacia Internet. Usan doble NAT: en KNF:cpe y en isp1. Se ha sustituido el switch KNF:access por uno controlado por OpenFlow. Además, se gestiona la calidad de servicio en la red de acceso mediante la API REST de RYU controlado por KNF:access. En el switch de KNF: cpe se ha activado la captura de tráfico ARP mediante arpwatch.

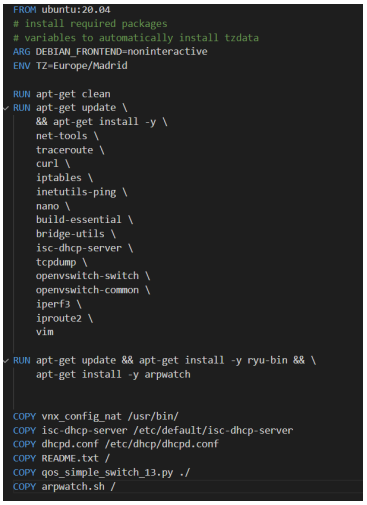
Desarrollo de la práctica

4. Repositorios propios

Trabajaremos en la carpeta compartida "*shared*". Dentro de ella, creamos el directorio "*rdsv-final*" en la que copiaremos las siguientes carpetas de la práctica 4: *helm*, *img*, *pck*, *vnx*. Y los scripts: *renes_start.sh*, *osm_renes_start.sh*, *osm_renes1.sh* y *osm_renes2.sh*.

5. Repositorio Docker

Creamos la cuenta prdsv en el repositorio público de DockerHub para subir nuestra imagen creada a partir del fichero Dockerfile mostrado a continuación:



```
FROM ubuntu:20.04
# install required packages
# variables to automatically install tzdata
ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Madrid

RUN apt-get clean
RUN apt-get update \
    && apt-get install -y \
    net-tools \
    traceroute \
    curl \
    iptables \
    inetutils-ping \
    nano \
    build-essential \
    bridge-utils \
    isc-dhcp-server \
    tcpdump \
    openvswitch-switch \
    openvswitch-common \
    iperf3 \
    iproute2 \
    vim

RUN apt-get update && apt-get install -y ryu-bin && \
    apt-get install -y arpwatch


COPY vnx_config_nat /usr/bin/
COPY isc-dhcp-server /etc/default/isc-dhcp-server
COPY dhcpd.conf /etc/dhcp/dhcpd.conf
COPY README.txt /
COPY qos_simple_switch_13.py ./
COPY arpwatch.sh /
```

Figura 3. Dockerfile.sh

Este fichero incluye los comandos añadidos según se han explicado en el enunciado de la práctica además de los necesarios para instalar paquetes de Ubuntu "ryu-bin" y "arpwatch", un fichero README.txt con el nombre de los integrantes del equipo, el script "*qos_simple_switch_13.py*" con la modificación que se propone en la práctica de QoS y un fichero arpwatch para automatizar el proceso de captura de tráfico ARP.

6. Creación del repositorio helm

Tal como se indica en el enunciado de la práctica, se siguen los pasos para crear el repositorio Helm y ss cambian los valores de los ficheros *values.yaml* de cada helm chart por el nombre de la imagen alojada en DockerHub. El repositorio creado lo se utilizará durante el despliegue de la práctica para el despliegue de un nuevo K8s Repository.

```
# Default values for cpechart.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1

image:
  repository: prdsv/vnf-img
  pullPolicy: Always
  # Overrides the image tag whose default is the chart appVersion.
  tag: "latest"
```

Figura 4. values.yaml

Finalmente, arrancamos desde OSM una instancia del servicio renes y mediante kubectl accedemos a los contenedores para comprobar que incluyen el software y los ficheros instalados.

7. Túneles VXLAN en KNF:access

La gestión de la calidad de servicio que hay que implementar en la KNF:access no funciona adecuadamente cuando se aplica sobre interfaces de túneles VXLAN creados desde un Open vSwitch, tal como se realiza en la práctica 4. Por ello, es necesario crear los túneles desde Linux con el comando "ip link".

Para realizar este cambio en KNF:access, debemos sustituir algunas líneas del fichero renes_start.sh y añadir otras: En cada VNF:acces debemos agregar un bridge y configurar las IPs y rutas y añadir el comando que crea el túnel que permita capturar el tráfico arp.

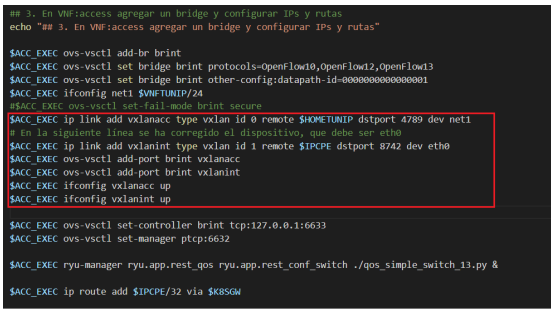


Figura 5. renes_start.sh

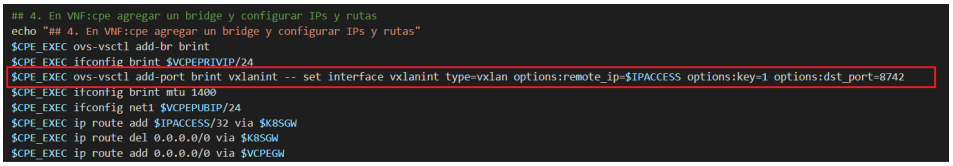


Figura 6. renes_start.sh

8. Automatización del despliegue del entorno

Lo primero que se debe hacer es iniciar las dos máquinas virtuales en el laboratorio: RDSV-K8S y RDSV-OSM.

A continuación, se accede al directorio compartido "*shared*" y se clona el repositorio de GitHub en ambas máquinas con el comando:

```
git clone https://github.com/prdsv/practica.git
```

Una vez descargado el repositorio, damos permisos de ejecución al fichero "*permisos.sh*", el cual contiene comandos que dan permisos de ejecución a los ficheros que se van a ejecutar en la práctica, como se puede ver a continuación:

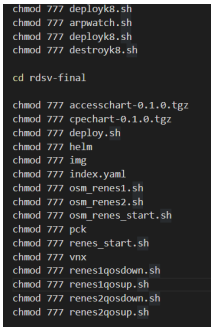


Figura 7. permisos.sh

Finalmente, se ejecuta deployk8.sh en RDSV-K8S y deploy.sh en RDSV-OSM. El primer deployK8.sh despliega el escenario inicial mostrado en la primera imagen. En el deploy.sh se seguen cada uno de los pasos de la Práctica 4 para dar conectividad a ambos escenarios: se define un cluster k8s en OSM, se lleva a cabo todo el proceso de on-boarding de NS/VNF y de la instalación de NS. Se crea dos instancias de servicio para los dos escenarios y una vez que las instancias de los servicios devuelvan *ready* ya será posible la configuración del servicio renes1 y renes2. Una vez lanzado el deploy y ejecutado ./osm_renes1.sh y ./osm_renes2.sh ambas redes tendrán conectividad a Internet y gracias al servidor DHCP se le asignarán a los hosts la ip. Para comprobar que se ha realizado, en la máquina RDSV-K8S, en la terminal de cada uno de ellos, se ejecuta `ifconfig`. Si no, se fuerza con el segundo comando `sudo dhclient eth1`. Y como se podrá comprobar todo funciona correctamente.

```
#Configuración de la MTU de la interfaz eth1
sudo ip link set dev eth1 mtu 1400

cd /home/upm/practica/rdsv-final

#Arranque de los escenarios server y home
sudo vnx -f vnx/nfv3_home_lxc_ubuntu64.xml -t

sudo vnx -f vnx/nfv3_server_lxc_ubuntu64.xml -t

cd ..
```

Figura 8. DeployK8.sh

```
export NSID1=$(osm ns-create --ns_name renes1 --nsd_name renes --vim_account dummy_vim)
echo $NSID1
watch osm ns-list
kubectl -n $OSMNS get pods

VI=$(kubectl -n $OSMNS get pods)
B=$(grep helmchartrepo-cpechart <<< "$VI")
A=$(grep helmchartrepo-accesschart <<< "$VI")

ACCPOD=$(echo $A | cut -d ' ' -f1)
CPEPOD=$(echo $B | cut -d ' ' -f1)

sleep 10

export NSID2=$(osm ns-create --ns_name renes2 --nsd_name renes --vim_account dummy_vim)
echo $NSID2
watch osm ns-list
kubectl -n $OSMNS get pods

VI2=$(kubectl -n $OSMNS get pods)
B2=$(grep helmchartrepo-cpechart <<< "$VI2")
A2=$(grep helmchartrepo-accesschart <<< "$VI2")

ACCPOD2=$(echo $A2 | cut -d ' ' -f6)
CPEPOD2=$(echo $B2 | cut -d ' ' -f6)

sleep 5

cd
cd practica/rdsv-final
```

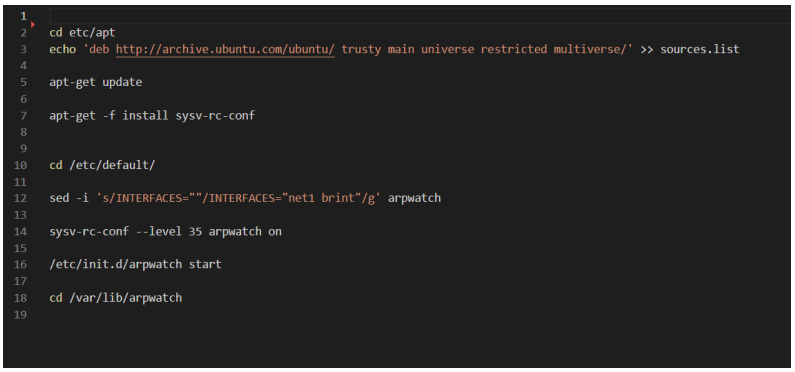
Figura 9. Deploy.sh

```
#Configuración de la MTU de la interfaz eth1
sudo ip link set dev eth1 mtu 1400

#Obtención del id k8cluster-list
KID1=$(osm k8scluster-list)
KID=${KID1:357:36}
echo $KID

#Namespace ustilizado por OSM en el clúster para desplegar los pods de los servicios de la red
OSM1=$(osm k8scluster-show --literal $KID | grep -A1 projects)
export OSMNS=${OSM1:25:36}
echo $OSMNS

#Automatización del on-boarding de NS/VNFs y la instanciación de NS mediante línea de comandos

osm repo-add helmchartrepo https://prdsv.github.io/repo-rdsv --type helm-chart --description "Repo para la practica de OSM"


cd pck
osm nfpkg-create accessknf_vnfd.tar.gz
osm nfpkg-create cpeknf_vnfd.tar.gz
osm nfpkg-list

osm nspkg-create renes_ns.tar.gz

osm nspkg-list
cd

#Creación de instancias del servicio tanto para renes1 como para renes2
export NSID1=$(osm ns-create --ns_name renes1 --nsd_name renes --vim_account dummy_vim)
echo $NSID1
watch osm ns-list
#Se borra el primero porque siempre falla la primera vez
osm ns-delete $NSID1

sleep 10
```

Figura 10. Deploy.sh

9. Captura de tráfico ARP mediante “arpwatch”

Arpwatch es un daemon para sistemas GNU/Linux que permite conocer quién se conecta a la red en todo momento. Se encarga de observar las correspondencias entre las entradas de la tabla ARP y la dirección origen, en el momento que se produce la conexión de un nuevo equipo se produce una nueva entrada en la tabla mientras se encuentre activado el arpwatch. Arpwatch se va a encargar principalmente de mantener un registro de la relación IP y de la MAC. Si se produce alguna alteración se notifica dicha alteración (1). En el escenario desarrollado se ha implementado mediante un fichero arpwatch.sh el proceso de captura de tráfico ARP dicho fichero se muestra en la siguiente imagen:



```
cd etc/apt
echo 'deb http://archive.ubuntu.com/ubuntu/ trusty main universe restricted multiverse/' >> sources.list

apt-get update

apt-get -f install sysv-rc-conf


cd /etc/default/

sed -i 's/INTERFACES=""/INTERFACES="net1 brint"/g' arpwatch

sysv-rc-conf --level 35 arpwatch on

/etc/init.d/arpwatch start

cd /var/lib/arpwatch
```

Figura 11. arpwatch.sh

Para poder realizar el proceso de captura es necesario instalar un programa que controle los servicios. En este caso, se instala sysv-rc.conf, seguido de este paso en el archivo de arpwatch en /etc/default se escribe en qué interfaces se quiere escuchar el tráfico ARP, en este caso net1 y brint. Una vez realizados todos esos pasos se procede a ejecutar el servicio de arpwatch con el comando:

```
sysv-rc-conf –level 35 arpwatch on
```

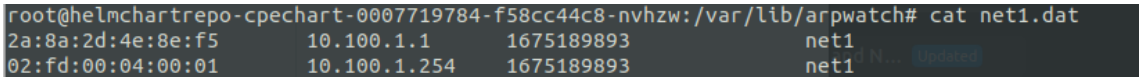
Y se inicia con:

```
/etc/init.d/arpwatch start
```

Se realizarán los pings de los host a internet y a rutas que no se encuentran en la red residenciales de esta forma cada vez que se hagan nuevos pings y se detecten nuevas entradas serán capturadas por este servicio. Para observar las ips capturadas se para el servicio mediante:

```
/etc/init.d/arpwatch stop
```

Y en /var/lib/arpwatch se observa las carpetas net1.dat y brint.dat en el caso de renes1:



```
root@helmchartrepo-cpechart-0007719784-f58cc44c8-nvhzw:/var/lib/arpwatch# cat net1.dat
2a:8a:2d:4e:8e:f5       10.100.1.1      1675189893              net1
02:fd:00:04:00:01       10.100.1.254    1675189893              net1
```

Figura 12. Direcciones IP capturadas en net1 para renes1

```
root@helmchartrepo-cpechart-0007719784-f58cc44c8-nvhzw:/var/lib/arpwatch# cat brint.dat
2e:33:6c:c4:46:45       192.168.255.1   1675189863              brint
02:fd:00:04:00:01       192.168.255.20  1675189838              brint
02:fd:00:04:01:01       192.168.255.21  1675189910              brint
```

Figura 13. Direcciones IP capturadas en brint para renes1

Y en el caso de renes2 las mismas carpetas pero con distintas direcciones ip::

```
root@helmchartrepo-cpechart-0015699164-dd994b884-rbftc:/var/lib/arpwatch# cat net1.dat
02:fd:00:04:00:01       10.100.1.254    1675191345              net1
b6:cb:33:a4:5d:46       10.100.1.2      1675191345              net1
```

Figura 14. Direcciones IP capturadas en net1 para renes2

```
root@helmchartrepo-cpechart-0015699164-dd994b884-rbftc:/var/lib/arpwatch# cat brint.dat
3e:a9:51:79:78:4b       192.168.255.1   1675191340              brint
02:fd:00:04:03:01       192.168.255.21  1675191340              brint
02:fd:00:04:04:01       192.168.255.22  1675191339              brint
02:fd:00:04:04:01       192.168.255.20  1675191351              brint
02:fd:00:04:03:01       192.168.255.23  1675191340              brint
```

Figura 15. Direcciones IP capturadas en brint para renes2

10. Conmutador Controlado por Openflow

La siguiente implementación es la sustitución del switch de KNF:acces por un conmutador controlado por OpenFlow. OpenFlow es un protocolo que va a permitir a un servicio decirle a donde puede enviar paquetes(2). Se va a encargar de controlar el comportamiento del forwarding en switches Ethernet en una red definida por software(3). Para realizar el cambio ha sido necesario modificar renes_start.sh.

```
## 3. En VNF:access agregar un bridge y configurar IPs y rutas
echo "## 3. En VNF:access agregar un bridge y configurar IPs y rutas"

$ACC_EXEC ovs-vsctl add-br brint
$ACC_EXEC ovs-vsctl set bridge brint protocols=OpenFlow10,OpenFlow12,OpenFlow13
$ACC_EXEC ovs-vsctl set bridge brint other-config:datapath-id=0000000000000001
$ACC_EXEC ifconfig net1 $VNFTUNIP/24
#$ACC_EXEC ovs-vsctl set-fail-mode brint secure
$ACC_EXEC ip link add vxlanacc type vxlan id 0 remote $HOMETUNIP dstport 4789 dev net1
# En la siguiente línea se ha corregido el dispositivo, que debe ser eth0
$ACC_EXEC ip link add vxlanint type vxlan id 1 remote $IPCPE dstport 8742 dev eth0
$ACC_EXEC ovs-vsctl add-port brint vxlanacc
$ACC_EXEC ovs-vsctl add-port brint vxlanint
$ACC_EXEC ifconfig vxlanacc up
$ACC_EXEC ifconfig vxlanint up

$ACC_EXEC ovs-vsctl set-controller brint tcp:127.0.0.1:6633
$ACC_EXEC ovs-vsctl set-manager ptcp:6632

$ACC_EXEC ryu-manager ryu.app.rest_qos ryu.app.rest_conf_switch ./qos_simple_switch_13.py &

$ACC_EXEC ip route add $IPCPE/32 via $K8SGW
```

Figura 16. renes_start.sh

Se añade la primera línea:

```
$ACC_EXEC ovs-vsctl set bridge brint protocols=OpenFlow10,OpenFlow12,OpenFlow13
```

Con este comando se consigue que en el puente con nombre brint del KNF:access se establezca la versión del protocolo OpenFlow. Además, se especifica que el puente pueda admitir las versiones 1.0,1.2 y 1.3 del protocolo OpenFlow

Continuando con la configuración del puente brint del KNF:access con el siguiente comando se establece el controlador en la dirección 127.0.0.1 y en el puerto 6633 usando el protocolo TCP:

```
$ACC_EXEC ovs-vsctl set-controller brint tcp:127.0.0.1:6633
```

Para finalizar con la configuración del conmutador controlado por OpenFlow, se establece una conexión de gestión para el puente Open vSwitch(OVS) del KNF:access en el puerto 6632 usando el protocolo TCP sobre IP.

```
$ACC_EXEC ovs-vsctl set-manager ptcp:6632
```

11. Ryu: controlar la calidad de servicio

Para conseguir la gestión de calidad de servicio en la red de acceso mediante la API REST de Ryu, una infraestructura basada en componentes para redes definidas por software, controlando KNF:access (4). Con él, se va a limitar el ancho de banda de bajada y subida hacia la red residencial, para la red residencial se establece un máximo de 12 Mbps de bajada y 6 Mbps de subida. Para H11 y H21 se establece un mínimo de 8 Mbps en bajada y un mínimo de 4 Mbps para la subida, mientras que para H12 y H22 se establece un máximo de 4 Mbps de bajada y un máximo de 2 Mbps en la subida.

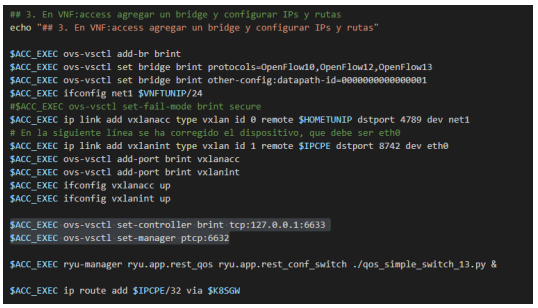


Figura 17. renes_start.sh

Para conseguirlo, se ha sustituido en primer lugar el switch de KNF:access por un conmutador OpenFlow que se conecta mediante la API REST de Ryu para controlar KNF:access mediante comandos ovs-vsctl a brint. Para ello se habilita el protocolo OpenFlow 10,12 y 13, configuramos el datapath y hacemos lo mismo con el controller y el manager, el primero en la dirección 127.0.0.1 y el puerto de Ryu 6633, y el segundo en el puerto 6632.

Una vez configurado, se programan las reglas que se ha mencionado anteriormente con las limitaciones, que vienen recogidas en los archivos renes1qosdown.sh, renes1qosup.sh para el primer renes en bajada y subida respectivamente, y lo mismo para renes2 en los archivos renes2qosdown.sh y renes2qosup.sh.

Asimismo, para conseguir realizar el proceso de subida en la carpeta /rdsv-final/vnx/nfv3_home_lxc_ubuntu64.xml se realizan los siguientes cambios en brg1 y brg2, la única diferencia sería la ip que cada uno tendría una distinta:

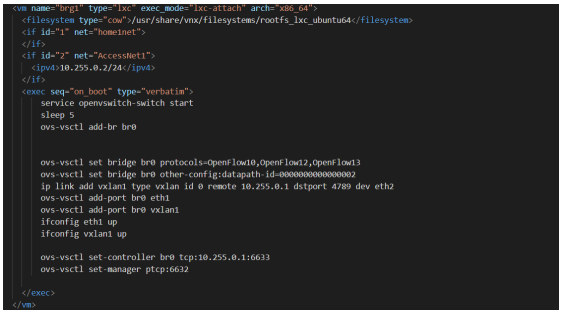


Figura 18. nfv3_home_lxc_ubuntu64.xml brg1

Para las pruebas, se ejecutan los comandos que se pueden ver en las capturas, los cuales también están accesibles desde el README del proyecto.

En primer lugar, se prueba el tráfico contenido por el renes1, es decir, el correspondiente a los hosts h11 y h12. Para ello se prueba la descarga de archivos y se comprueba que su ha limitado a un máximo de 4 Mbps y a un máximo 12 Mbps marcado por la red residencial.

Cuando se prueba el tráfico de bajada en el KNF:cpe es menor que 12Mbps cuando se manda a h12:

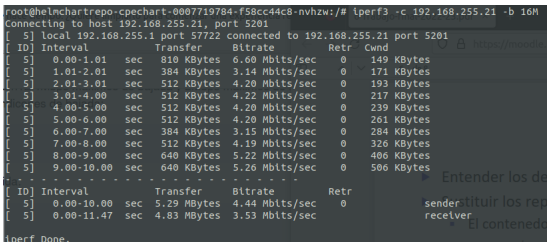


Figura 19. Tráfico en CPE1 que envía CPE1 a h12

Y se comprueba que el tráfico recibido se encuentra por debajo del límite de 4 Mbps en el host.

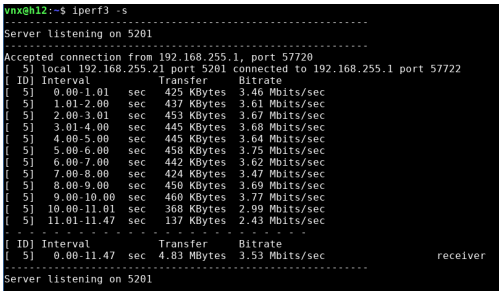


Figura 20. Tráfico en h12 en bajada

A continuación, se comprueba que el tráfico se ha limitado bien en el proceso de subida entre el host h11 y el KNF:cpe, que, como se puede observar, transmite con una tasa menor que 6Mbps, que es el máximo establecido en las condiciones de QoS.

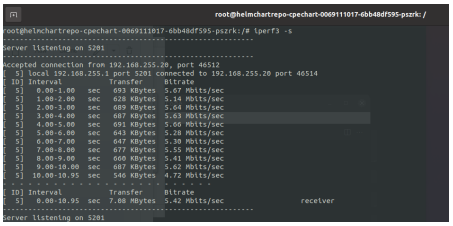


Figura 21. Tráfico en CPE1 que envía h11 a CPE1

Y cuando manda tráfico el host h11 se comprueba que el tráfico siempre es mayor que el mínimo establecido de 4Mbps.

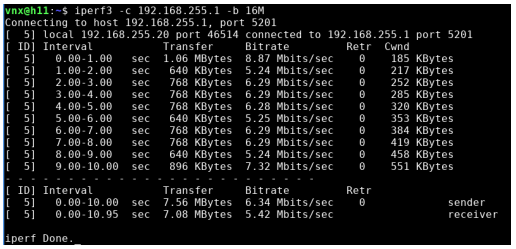


Figura 22. Tráfico en h11 en subida

También se comprueba que el tráfico se ha limitado bien en el proceso de subida desde h12 hacia KNF:cpe. Se puede observar que es menor que 6 Mbps, que es el máximo establecido en las condiciones de QoS.

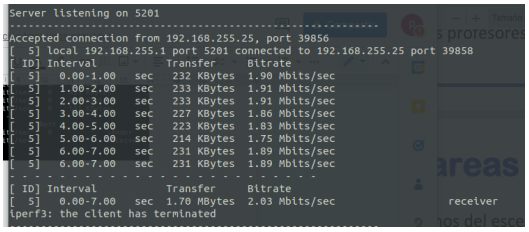


Figura 23. Tráfico en CPE1 que envía h12 a CPE1

Al mismo tiempo, en H12 vemos que cumple con el límite máximo de subida de 2Mbps

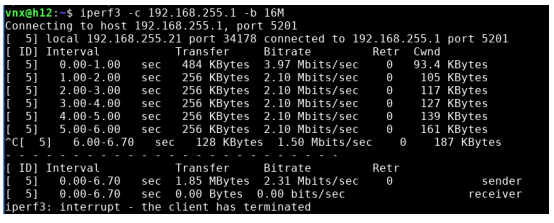


Figura 24. Tráfico en h12 en subida

Tras esto, procedemos a ejecutar las mismas comprobaciones para el renes2, que ha de cumplir las mismas condiciones, en este caso para sus hosts h21 y h22. Vienen recogidas en el Anexo I.

12. ANEXO I

En este caso, comenzamos con el envío de paquetes desde CPE2 hacia el host h22, que como podemos observar cumple con el límite máximo de 6 Mbps de la red residencial.

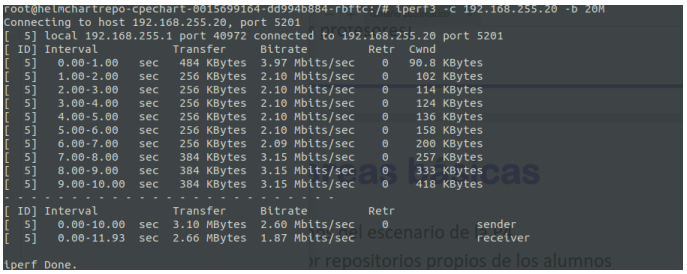


Figura 25. Tráfico en CPE2 que envía CPE2 a h22

De igual forma, vemos que se configuran correctamente en h22 y que se recibe por debajo del máximo de 4 Mbps establecido en las reglas.

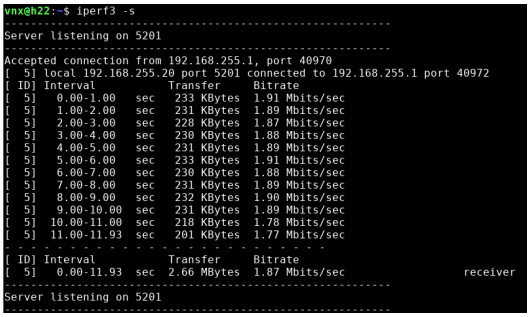


Figura 26. Tráfico en h22 en bajada

Sin abandonar h22, procedemos a comprobar en este caso el enlace de subida, comprobamos que recibe por debajo del máximo de subida de 6Mbps establecido para la red residencial.

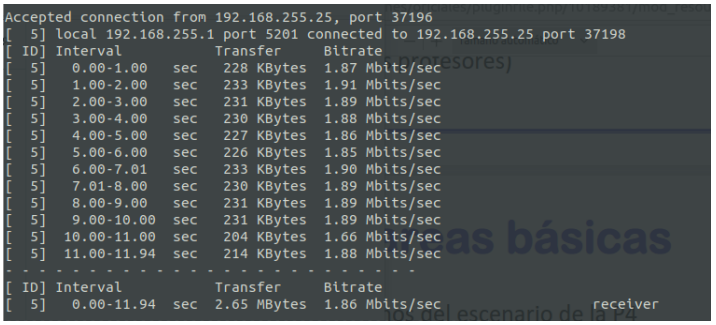


Figura 27. Tráfico en CPE2 que envía h22 a CPE2

De igual forma, h22 está enviando por debajo del límite impuesto de 2Mbps.

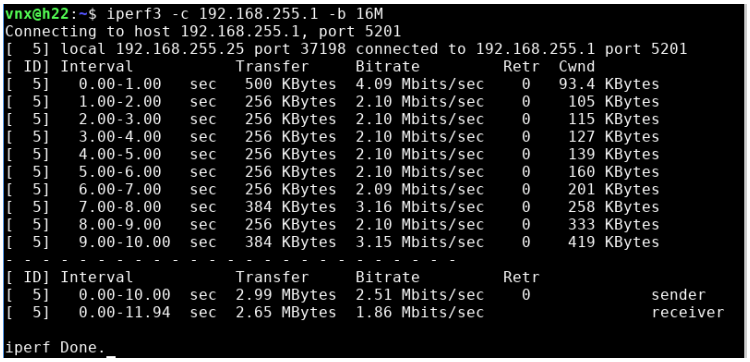


Figura 28. Tráfico en h22 en subida

Finalmente, pasamos a analizar el tráfico en h21, que como podemos ver en las imágenes a continuación, cumple también con lo establecido, enviando al CPE por debajo del máximo de 6 Mbps para la red residencial.

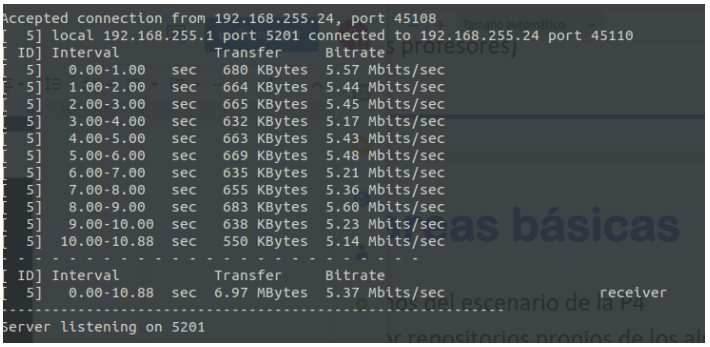


Figura 29. Tráfico en CPE2 que envía h21 a CPE2

Y por el otro lado, enviando por encima de lo 4 Mbps de mínimo.

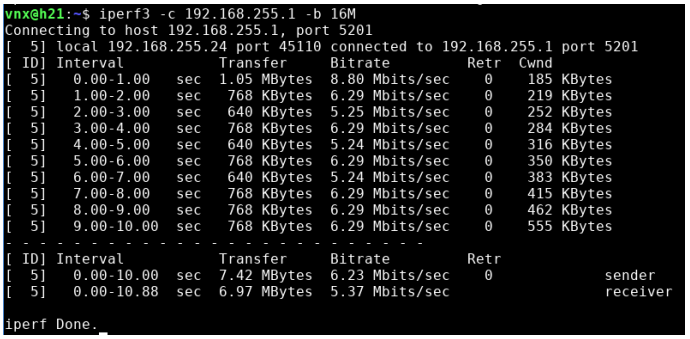


Figura 30. Tráfico en h21 en subida

Finalmente, ejecutamos la bajada hacia h21 desde CPE con la que nos hemos encontrado con un fallo, ya que CPE envía por debajo de máximo de la red residencial, pero no cumple con el mínimo de 8 que se exige en la regla de subida.

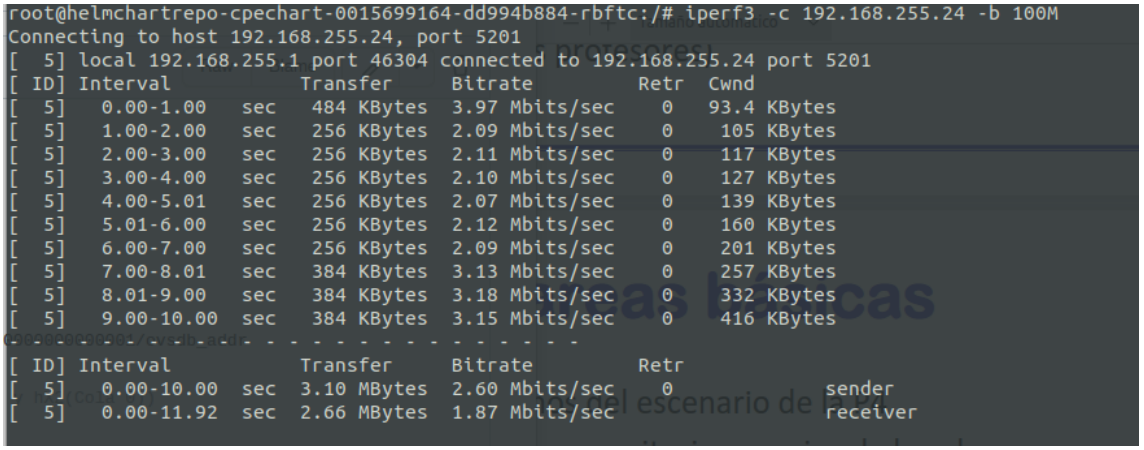


Figura 31. Tráfico en CPE2 que envía CPE2 a h21

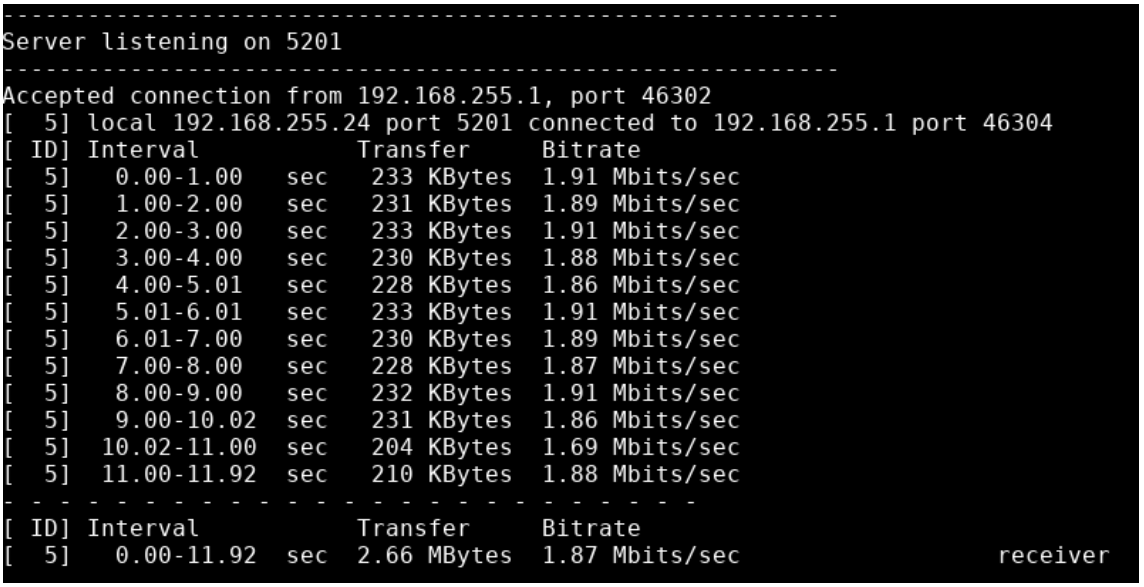


Figura 32. Tráfico en bajada en h21