UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

MÁSTER EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

ESTUDIO DEL RENDIMIENTO DE UN ENTORNO EMULADO DE VIRTUALIZACIÓN DE SERVICIOS DE RED EXTREMO A EXTREMO

TRABAJO FIN DE MÁSTER

Alumno: Enrique Sánchez Martín

Tutor: Katja Gilly de la Sierra - Ll. / Salvador Alcaraz Carrasco

A mi familia y amigos, en especial a mi perro Duque allá donde esté

Y a mi tutora Katja, por su ayuda y por guiarme a lo largo de este proyecto.

Prólogo

El presente trabajo de fin de máster titulado como “ESTUDIO DEL RENDIMIENTO DE UN ENTORNO EMULADO DE VIRTUALIZACIÓN DE SERVICIOS DE RED EXTREMO A EXTREMO” aborda de una manera práctica la implementación de un entorno de red emulado sobre el que se aplicarán funciones de red virtualizadas para la transmisión extremo a extremo de datos entre distintos hosts que integran la topología.

Para realizar dicho cometido se han seguido distintos estándares y fuentes oficiales con experiencia en el sector, destacando el estándar TOSCA desarrollado por el grupo industrial OASIS, un grupo donde distintas organizaciones y gobiernos forman alianzas para el desarrollo de código y estándares abiertos, sobre el cual se llevan a cabo distintas y numerosas aplicaciones basadas en la virtualización de funciones de red y sobre el que se realizan las distintas plantillas que se han empleado en este trabajo.

Por otro lado, esta investigación se basa y sigue la senda de conocimiento establecida por el Instituto Europeo de Normas de Telecomunicaciones (ETSI), centrándose en la tecnología o aplicación nombrada como *Open Source NFV Management and Orchestration* (MANO) sobre la cual está basado este estudio.

En la primera parte de este proyecto se exponen de una forma concisa y ordenada los conocimientos teóricos necesarios para su correcta comprensión, así como las distintas herramientas utilizadas que han hecho posible que se lleve a cabo y se hayan obtenido los resultados esperados. Tras la obtención de dichos datos y tras analizarlos debidamente se llega a una conclusión final, expuesta al final del presente documento.

Índice

Capítulo 1: Introducción 12
1.1 Antecedentes 12
1.2 Objeto y alcance 12
1.3 Metodología 13
Capítulo 2: Base teórica 14
2.1 Virtualización: Historia y definición 14
2.2 Redes definidas por software (SDN) 15
2.3 Virtualización de funciones de red (NFV) 17
2.3.1 Funciones de red virtualizadas (VNF) 18
2.3.1.1 Virtual Network Function Descriptor (VNFD) 19
2.3.2 Infraestructura de Virtualización de Funciones de red (NFVI) 21
2.3.2.1 VNF Forwarding Graphs 22
2.3.3 Network Functions Virtualisation Management and Orchestration (NFV MANO) 23
2.3.3.1 Function Virtualization Orchestrator (NFVO) 24
2.3.3.2 Virtualized Network Function Manager 24
2.3.3.3 Virtualised Infrastructure Manager (VIM) 25
2.4 OASIS TOSCA 26
2.5 NFV: Ventajas y retos 27
2.6 Escalado de VNF 28
Capítulo 3: Herramientas 30
3.1 Mininet 30
3.2 Oracle VM VirtualBox 31
3.3 Wireshark 32
3.4 Linux Ubuntu 18.04 33
3.5 Mini-nfv 33
3.6 BMON 34
Capítulo 4: Preparación del entorno 35
4.1 Instalación y configuración de VirtualBox 35
4.2 Instalación de Ubuntu 39
4.3 Instalación de Mininet 42
4.4 Instalación de Wireshark 43
4.5 Instalación Mini-nfv 44
4.6 Instalación Bmon 46

Capítulo 5: Pruebas de laboratorio 47
5.1 Estudio del ancho de banda 47
5.2 Sobrecarga del VNF 59
5.3 Comparación de tráfico entre una red con tecnología NFV y sin tecnología NFV ... 60
6. Conclusiones 63

Índice de figuras

Figura 1: Arquitectura Hosted y arquitectura Bare-Metal……..15

Figura 2: Arquitectura SDN……16

Figura 3: Arquitectura NFV…….18

Figura 4: Plantilla VNFD……..20

Figura 5: Plantilla VNFFGD……22

Figura 6: Elementos estructurales plantilla de servicio……..26

Figura 7: Flujo de creación de una VNF……..28

Figura 8: Opciones de descarga VirtualBox……34

Figura 9: Instalación de VirtualBox…….34

Figura 10: Instalación de VirtualBox 2……35

Figura 11: Instalación de VirtualBox 3……35

Figura 12: Panel de VirtualBox……36

Figura 13: Creación de máquina virtual……..36

Figura 14: Creación de disco duro virtual……...37

Figura 15: Configuración adaptador de red VirtualBox……38

Figura 16: Descarga de Ubuntu 18.04……..38

Figura 17: Configuración máquina virtual con imagen Ubuntu……...39

Figura 18: Instalación de Ubuntu…….39

Figura 19: Escritorio Ubuntu……...40

Figura 20: Escritorio Ubuntu……...40

Figura 21: Instalación de mininet……41

Figura 22: Iniciación de mininet……..42

Figura 23: Instalación de Wireshark……...43

Figura 24: Iniciación de Wireshark……43

Figura 25: Descarga de mini-nfv……..44

Figura 26: Iniciación de mini-nfv……45

Figura 27: Topología 1, árbol básico……46

Figura 28: Plantilla VNFD 1……47

Figura 29: Plantilla vnffgd-helloworld-i.yaml……48

Figura 30: Pruebas con un único flujo……49

Figura 31: Topología 2, árbol básico con 4 hosts……50

Figura 32: Plantilla flujo 2……51
Figura 33: Pruebas con dos flujos……52
Figura 34: Topología 3, árbol básico con 6 hosts……53
Figura 35: Plantilla flujo 3……54
Figura 36: Pruebas con 3 flujos……55
Figura 37: Topología 4, árbol básico con 8 hosts……55
Figura 38: Plantilla flujo 4……57
Figura 39: Pruebas con 4 flujos……58
Figura 40: Gráfica de ancho de banda frente al número de flujos……58
Figura 41: Sobrecarga de VNF……60
Figura 42: Interfaz de Wireshark……61
Figura 43: Comparación de red sin NFV y red con NFV……62

Capítulo 1: Introducción

1.1 Antecedentes

En la actualidad, las redes de comunicación forman parte de nuestro día a día. Se trata de un concepto que ha colonizado por completo el sector de las comunicaciones y que se encuentra presente en cualquier ámbito de dicho sector. Remontándonos a su significado más elemental, una red es un conjunto de recursos software, dispositivos hardware, protocolos, tecnologías y medios utilizados para la transmisión de datos o información entre los distintos elementos que integran dicha red.

Se distinguen distintos tipos de redes según su estructura, tamaño y alcance, siendo algunas de las más populares:

- Redes de área local (LAN)
- Redes de área amplia (WAN)
- Redes de área metropolitana (MAN)
- Redes de área global (GAN)

Este tipo de redes emplean infraestructuras de redes físicas, que sirven como base para las redes de comunicaciones lógicas o virtuales, como las Virtual Private Networks (VPN), las cuales vinculan dispositivos de red, que emplean un determinado medio físico de transmisión, mediante software de tunelización sin necesidad de que dichos dispositivos estén conectados físicamente entre sí.

Con el crecimiento y el avance de este tipo de redes surge una fuerte demanda en lo que a despliegues de infraestructuras de tecnología de la información (TI) se refiere. Esta demanda surge de la aparición de nuevos servicios, los cuales conllevan un aumento del tráfico, por lo que se requieren nuevos métodos de gestión de recursos que mejoren la experiencia de usuario controlando valores como la latencia y que permitan de alguna manera ahorrar costes.

Es aquí donde aparece y cobra protagonismo el concepto de virtualización de redes, el cual es un concepto enfocado en desvincular los servicios de red del hardware permitiendo su aprovisionamiento de una forma virtualizada a nivel de software, logrando así una gestión de los recursos de red más flexible, efectiva y ágil.

Con la aparición y adopción del concepto de virtualización aparecen nuevos términos subyacentes que afectan directamente a las redes, que son las redes definidas por software (SDN) y la virtualización de funciones de red (NFV). Dichos conceptos marcan las pautas y establecen las premisas que el sector de las comunicaciones está adoptando y adoptará en un futuro cada vez más próximo en busca de un rendimiento óptimo y eficaz ante las nuevas necesidades y servicios emergentes.

1.2 Objeto y alcance

El objeto y objetivo principal de este proyecto es la implementación de un entorno de red virtualizado mediante software de emulación que nos permita evaluar el rendimiento de una red

extremo a extremo utilizando e implementando funciones de red virtualizadas (VNF) conforme marca el estándar y entorno difundido por el ETSI.

El alcance de este proyecto consiste en:

- Elaboración de un entorno de red SDN con Mininet y sus herramientas.
- Esclarecer e indagar en conceptos relacionados con las redes que aplican la tecnología NFV.
- Utilizar la base teórica ofrecida por el Instituto Europeo de Normas de Telecomunicaciones, destacando y basando la estructura de este proyecto en el *framework* NFV MANO.
- Simular un entorno de red aplicando NFV y observar su comportamiento con tráfico simulado extremo a extremo.
- Aplicación de diferentes flujos de datos que pasan a través de VNFs modificando y aplicando distintas plantillas y parámetros de inicialización.
- Estudio del ancho de banda en este tipo de entornos de comunicación.

1.3 Metodología

En la realización de este proyecto se han seguido una serie de fases para su correcta ejecución, las cuales son:

- Documentación y estudio de los conceptos teóricos necesarios: Esta fase del proyecto es la más temprana debido a que es la fase en la que se han adquirido todos los conceptos teóricos necesarios para su correcta comprensión y ejecución futura.

- Estudio y selección de las herramientas empleadas para la realización del proyecto: En esta fase del proyecto se realizó un estudio en el que se seleccionaron las herramientas que formarían parte del mismo y que permitirían obtener los resultados esperados.

- Desarrollo del proyecto y ejecución de la simulación: En esta fase se aplican los conocimientos obtenidos en las fases previas y se desarrolla un escenario sobre el cual se implementará la simulación deseada para la obtención de los resultados finales sobre los cuales se podrán establecer conclusiones.

Capítulo 2: Base teórica

2.1 Virtualización: Historia y definición

La virtualización es un concepto que ha ido ganando protagonismo con el paso de los años, encontrando su auge en la época actual. No obstante, se trata de un concepto relativamente antiguo ya que su aparición se remonta a la década de 1960, teniendo en cuenta el significado que se le da actualmente a dicho concepto.

Este término nació de la mano de IBM en busca de una solución o método para que los ordenadores o mainframes que se utilizaban en dicha época pudiesen particionar o segmentar lógicamente la utilización de la CPU para que dichos equipos pudiesen realizar simultáneamente distintas tareas u operaciones. Es decir, el objetivo que se perseguía era poder crear fragmentaciones o particiones lógicas que pudiesen trabajar con independencia las unas de las otras, empleando para ello los distintos recursos que se les habían asignado [1].

Estableciendo esta premisa y en base a dicho concepto, en la década de los 1980 con la aparición de nuevas tecnologías como la arquitectura x86, la virtualización comenzó su gran expansión dentro del sector de las tecnologías de la comunicación y la información (TIC). La aparición de procesadores que empleaban dicha arquitectura fue un verdadero hito tecnológico, estableciendo las bases para el avance del concepto de virtualización. Así, en la década de 1990 el concepto de máquina virtual (VM) se encontraba notablemente extendido, alcanzando una alta popularidad y surgiendo el concepto de Cloud Computing en la década siguiente.

La entrada al nuevo milenio trajo consigo nuevos avances en este sector. La demanda de nuevos servicios y el aumento de los datos consumidos dio como resultado que ante la escasez y la dificultad de llevar a cabo expansiones de los centros de datos a nivel físico se optase por empezar a virtualizar dichos centros de datos. Así, en un mismo hardware se empezaron a ejecutar varias máquinas virtuales que efectuaban distintas funciones [2].

Uno de los conceptos más importantes sobre el que se sustenta la virtualización es el de hipervisor. Según la información facilitada por Redhat: “Un hipervisor, conocido también como monitor de máquina virtual (VMM), es un software que crea y ejecuta máquinas virtuales (VM) y que, además, aísla el sistema operativo y los recursos del hipervisor de las máquinas virtuales y permite crearlas y gestionarlas” [3].

Se distinguen dos tipos de estructura de virtualización, las cuales son la arquitectura *Hosted* y la arquitectura *Bare Metal*. La principal diferencia entre ambas es que en la arquitectura *Hosted* hay definido un sistema operativo mientras que en la arquitectura *Bare Metal* la función de virtualizar se ejecuta directamente sobre el hardware, tal y como se observa en la Figura 1.

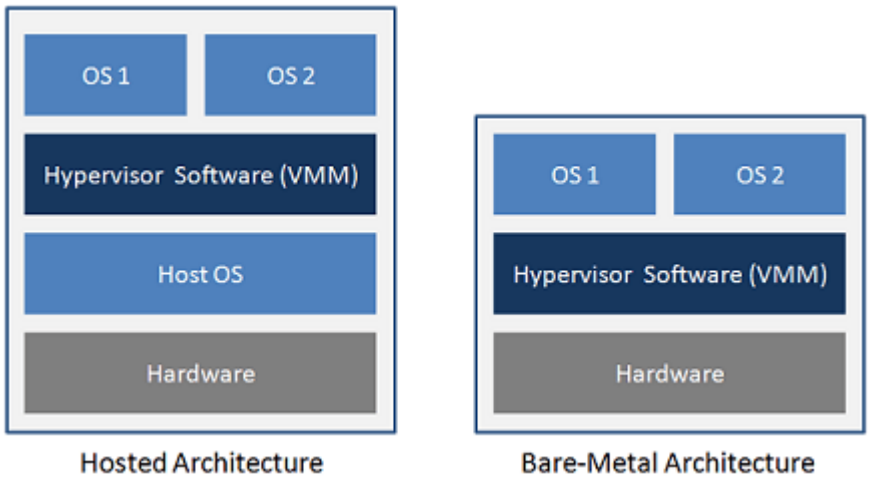

Figura 1. Arquitectura Hosted y arquitectura Bare-Metal.

Fuente: NI-White-Paper.

Establecidas dichas premisas y con la expansión del sector de las comunicaciones, la virtualización pasó a ser un concepto clave en la mayor parte de las redes que buscaban aumentar su eficiencia ante la creciente demanda de servicios que conllevaban un elevado flujo de datos. Con esta virtualización los recursos de red se desvincularon del hardware, dando paso a las redes definidas por software (SDN) y a la virtualización de funciones de red (NFV).

2.2 Redes definidas por software (SDN)

Las redes de comunicación han sufrido cambios a lo largo de los años en busca de adaptarse y mejorar su eficacia ante la aparición de nuevos servicios que exigen cada vez mayores prestaciones. Con el creciente volumen del tráfico a transmitir empezaron a surgir limitaciones en el ámbito del entorno físico al ligar dichas redes a dispositivos hardware, por lo que la virtualización empezó a plantearse como la alternativa ante dicha problemática.

Ante esta nueva perspectiva surgieron las redes definidas por software, las cuales son redes cuyo enfoque se basa en una red programada y controlada mediante aplicaciones software, aislando los recursos de red en un sistema completamente virtualizado. En este nuevo escenario se establece una distinción entre el plano de control y el plano de datos, dicha distinción permite que la red pueda diseñarse y programarse de una forma centralizada.

El plano de control se centra en la lógica de transmisión y el enrutamiento y conmutación de paquetes. Por otro lado, el plano de datos es el encargado en la transmisión y recepción de paquetes.

De estas dos capas surge una tercera que es la capa de aplicación, en la cual se encuentran todos los programas que toman decisiones sobre la red para satisfacer las necesidades de los usuarios y los requisitos de los distintos servicios.

Las distintas capas o niveles se comunican e interaccionan entre sí por medio de una interfaz de programación de aplicaciones (API), siendo el protocolo OpenFlow el estándar empleado para el

desarrollo e implementación de este tipo de arquitectura de red. La API dirección norte permite la comunicación entre el plano de control y el plano de aplicación para solicitar recursos y servicios, mientras que la API dirección sur sirve para comunicar el plano de datos con el plano de control, identificado así la topología de red y el comportamiento de los nodos.

En la figura 2 se observa esta distinción entre los planos de los que se componen las redes que presentan dicha arquitectura:

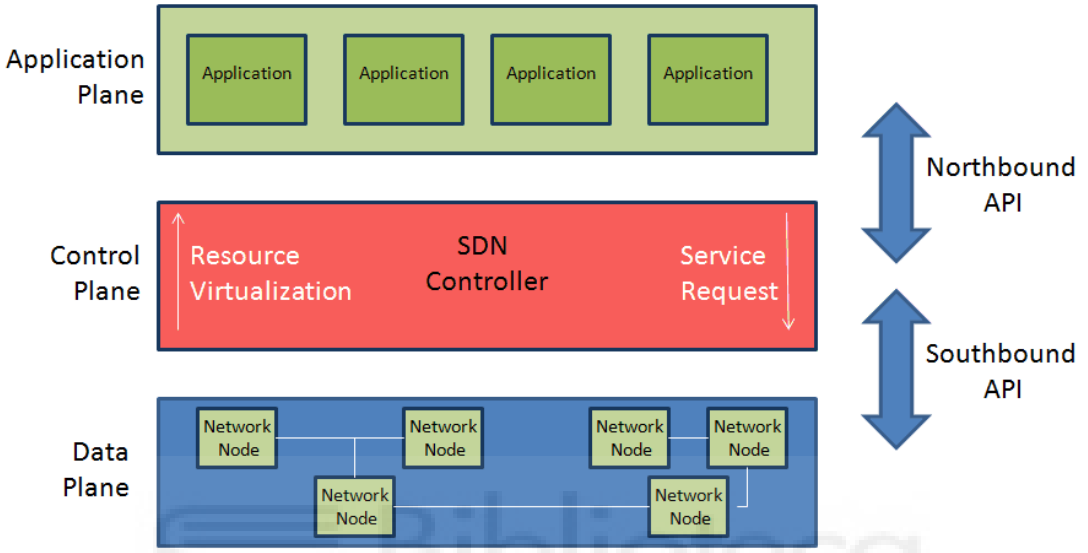

Figura 2. Arquitectura SDN. Fuente: Pandora FMS blog.

Según lo recogido por un estudio realizado por Journal de Ciencia e Ingenieria se establece que una red definida por software debe integrar los siguientes elementos [3]:

- Un protocolo software.
- Un sistema operativo de red.
- Equipo de cómputo controlador.
- Equipos con soporte SDN de *Switching* y *Routing*.
- Lenguajes de programación de alto nivel empleados para las políticas de comunicación.

El controlador será el componente central, que actuará como el cerebro de la red estableciendo contacto con las demás capas mediante las distintas APIS expuestas anteriormente. Dicho controlador tendrá instalado un sistema operativo como por ejemplo Linux, Windous o MacOS. Dicho controlador es un elemento importante en la lógica de control de la red ya que es el encargado de adaptar la red a las políticas definidas por el operador de red.

Por otro lado, en los equipos de *Switching* y *Routing* deberá comprobarse la correcta provisión de las tablas de flujo para garantizar la comunicación, ya que las decisiones de reenvíos de paquetes están basadas en flujos.

Con el uso de la virtualización y la aparición de este tipo de redes se obtuvieron numerosos beneficios, como disminuir la necesidad de innovar a nivel de hardware para adaptarse ante las nuevas necesidades de la red, mayor independencia y control sobre la infraestructura, mayor flexibilidad, mayor seguridad y una reducción de costes.

Por otro lado, este tipo de redes ofrecen una centralización de la lógica de control, lo cual permite que las políticas de la red puedan ser modificadas mediante lenguajes de alto nivel y componentes software de una forma más sencilla, pudiendo además reaccionar de una forma más eficiente, ágil y automática ante cambios en el estado de la red pudiendo así mantener las políticas de alto nivel intactas.

2.3 Virtualización de funciones de red (NFV)

Con la evolución de la virtualización y con la aparición de las redes definidas por software surgieron a su vez nuevos conceptos como el de la virtualización de funciones de red o NFV. Los conceptos de SDN y NFV están estrechamente relacionados entre sí y en multitud de redes coexisten. Sin embargo, existen diferencias entre ellos y la más notoria es que el concepto de SDN suele aplicarse a la virtualización de la red en su conjunto, mientras que NFV está más enfocado en la virtualización de los servicios de red de forma individual.

Con la aparición de NFV se avanza hacia la sustitución de los dispositivos hardware dedicados por máquinas virtuales capaces de aplicar y de virtualizar dichas funciones. Es decir, la virtualización de funciones de red es aquella tecnología en la cual las funciones que antes estaban asociadas a un hardware específico pasan a ejecutarse sobre servidores como máquinas virtuales.

Esta tecnología posibilitó a los proveedores de servicios de red ofrecer dichos servicios sin la necesidad de incorporar nuevos recursos hardware, consiguiendo de esta manera dotar a la red de una mayor agilidad y escalabilidad en la adopción de nuevos servicios.

Por otro lado, algunas de las ventajas que ofrece la aplicación de esta tecnología son la reducción en los costos del mantenimiento de la red, la facilidad para actualizar y aplicar cambios sobre dichas funciones de red y un menor consumo de energía. La implementación de esta tecnología favorecerá al usuario en cuanto que las redes podrán atender nuevas peticiones de servicios y adaptarse a estos de una forma más eficiente y ágil al ser un entorno completamente virtualizado y que goza de una alta programabilidad [4].

Esta tecnología presenta una arquitectura característica, la cual se encuentra estandarizada por el ETSI, lo cual permite una mayor interoperabilidad y definir un estándar de implementación. En concreto, en Noviembre de 2012 se creó la ETSI NFV ISG (Industry Specification Group) la cual actualmente la conforman más de 150 miembros pertenecientes al sector de las telecomunicaciones [5].

El concepto de NFV está fuertemente ligado al de *Cloud computing*, ya que se basa en muchas de las tecnologías de virtualización empleadas en este campo, como la virtualización del hardware por medio de hipervisores y el uso de switches virtuales de los que se sirve para obtener una fuerte capacidad de procesamiento de paquetes y asegurar la conexión entre las máquinas virtuales y las interfaces físicas que intervienen en la comunicación.

Según el estándar planteado por la ETSI, la arquitectura vigente en esta tecnología está dividida en diferentes bloques o niveles funcionales, los cuales se pueden observar en la Figura 3. Estos niveles son:

- VNF
- NFVI
- NFV MANO

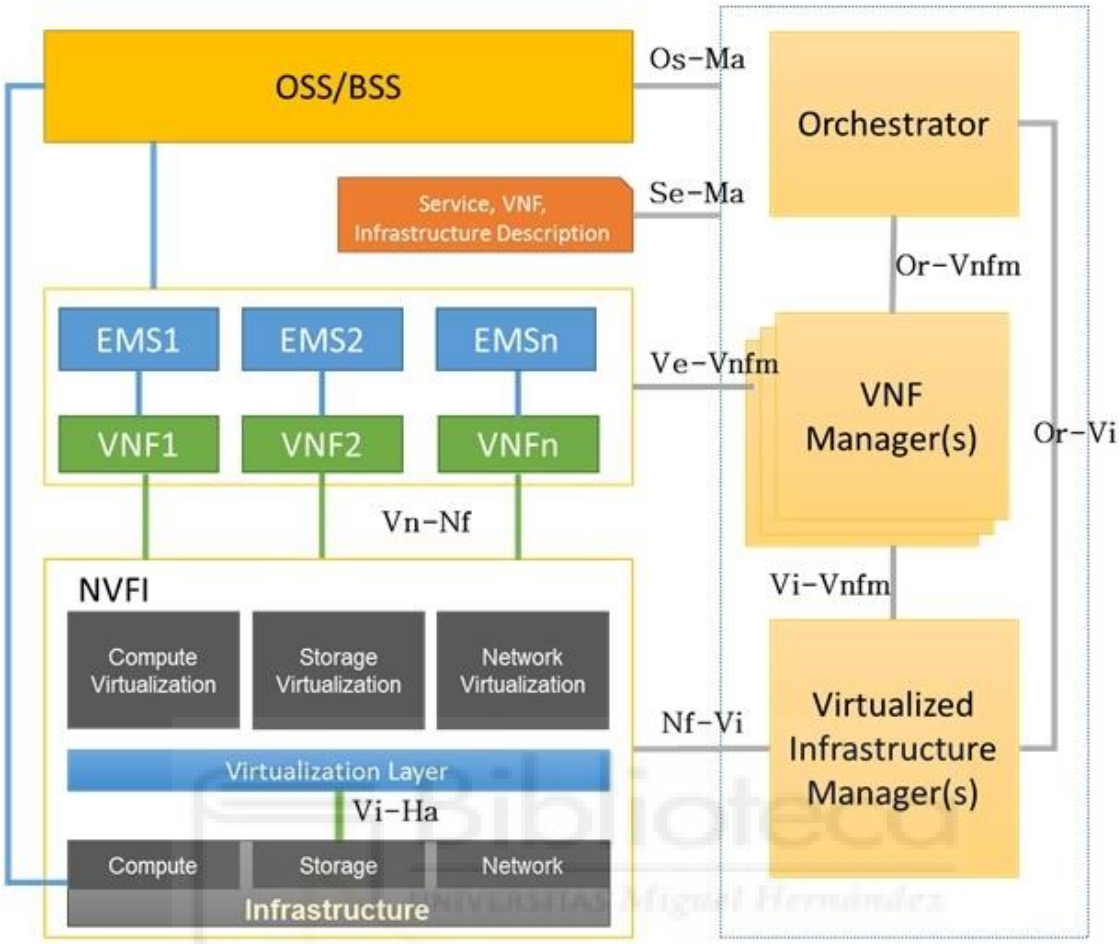

Figura 3. Arquitectura NFV. Fuente: ETSI NFV ISG.

2.3.1 Funciones de red virtualizadas (VNF)

Uno de los conceptos clave sobre los cuales se sustenta cualquier arquitectura NFV es el de las funciones VNF. Una función virtualizada de red es aquella función que antes se ejecutaba en hardware dedicado y propietario pero que ahora se ejecuta sobre el software de máquinas virtuales, controladas por un hipervisor.

Cada VNF por sí mismo ofrece una funcionalidad limitada. No obstante, cuando se desarrollan módulos de VNF estos operan conjuntamente y procesan el tráfico de datos para obtener y ejecutar la funcionalidad de red deseada por el proveedor de servicios. Es decir, una sola VNF por sí misma no constituye una NFV en su totalidad, es el conjunto de varias de estas, inteligentemente organizado para realizar una determinada función de red, lo que constituye una red de estas características. Esto es conocido como *service chaining*, concepto utilizado para referirse a la concatenación de estas funciones de red virtualizadas en busca de proveer el servicio correcto.

Con la aparición de esta tecnología se consiguió una mayor flexibilidad a la hora de aplicar estos sistemas que antes requerían de hardware específico. En su lugar, dicha tecnología permitió la

aplicación de dichas funcionalidades de red en plataformas compartidas o hardware genérico de distintos fabricantes.

La mayoría de VNF son productos que se pueden adquirir en el mercado bajo contrato gubernamental, pudiendo ser combinadas por el proveedor de servicios de la forma más eficiente para conseguir el servicio deseado con un rendimiento óptimo. Para ello, el proveedor deberá configurar dichas VNF, las interconexiones entre las mismas para garantizar una correcta comunicación entre las distintas unidades y como interaccionan entre ellas controlando a su vez el ciclo de vida de las mismas.

Por otro lado, el proveedor de servicios será el responsable de garantizar una correcta comunicación e interacción de los VNF con sus sistemas propietarios de OSS (Operations Support Systems) y BSS (Bussines Support Systems).

Dentro de las VNF se pueden distinguir las Virtualisation Deployment Unit (VDU), las cuales son unidades utilizadas en los modelos de información que describen la implementación y el comportamiento operativo de un subconjunto de un VNF, o de todo el VNF si no forma parte de un subconjunto [6]. Por otro lado, una VNF está siempre compuesta por componentes conocidos como Virtual Network Function Component (VNFC), los cuales son componentes que otorgan a la función virtualizada de red un subconjunto definido de funcionalidades. Ambos conceptos están relacionados.

Por otro lado, existen los Element Management System (EMS), que no son más que son unas entidades que se encargan de gestionar la configuración, el control de fallos, el rendimiento, las funciones y la seguridad de las VNF. Estos elementos pueden estar asociadas en una asociación 1-1 con un VNF o por el contrario estar asociadas a múltiples VNF. A su vez, los EMS pueden implementarse como VNF.

2.3.1.1 Virtual Network Function Descriptor (VNFD)

Para implementar un VNF se emplean determinadas plantillas donde están definidos los atributos de dicho VNF. A su vez, esta plantilla será utilizada por la capa NFV MANO para instanciar dicho VNF. Cada VNFD se almacenará en el catálogo de VNF el cuál será consultado cada vez que se instancie una VNF. Los recursos serán asignados por la capa de infraestructura (NFVI) en base a dicha plantilla. No obstante, los requisitos establecidos en el VNFD pueden ser anulados por los requisitos establecidos previamente en la solicitud de creación de la instancia, es decir, cuando se instancie una VNF por parte del orquestador de NFV, éste podrá definir otros requisitos que no necesariamente serán iguales a los definidos en la plantilla VNFD.

Tal y como se observa en la figura 4 una plantilla VNFD consta de VDUs, Connection Points (CP) y Virtual Links (VL). Dichos elementos son considerados como nodos, los cuales están recogidos en el subapartado *node_templates* dentro del apartado *topology_template*. En dichos apartados se definirán las distintas características y atributos de estos nodos [7].

- VDU: El VDU es la máquina virtual donde se alojará la función de red y donde se define la imagen, los drivers que manejarán dichas VDU así como el *flavor*. El flavor será la parte encargada de definir el hardware y los recursos físicos tales como la RAM y el disco, por su parte, la imagen es la plantilla que se utiliza para crear las nuevas instancias, dichas imágenes pueden constar de un sistema operativo y de software instalado. La

imagen y el flavor se definen en las secciones *image* y *flavor* dentro del apartado *properties*.

En la sección *nfv_compute* se describe como se configurará la máquina virtual que reside en la VDU pudiendo seleccionar el tamaño de memoria, el número de cpus y el tamaño del disco en las secciones *num_cpus*, *mem_size* y *disk_size*.
Por otro lado, se encuentran los parámetros *user_data* y *user_data_format*, los cuales definen los comandos que se ejecutan en la VDU así como el formato de los mismos.

- CP: Los puntos de conexión son aquellos puntos empleados para conectar los distintos VL definidos en la plantilla, pudiendo ser NIC virtuales o una NIC SR-IOV. Los CP siempre requieren de VL y VDU conectados a él.

  Dentro de los parámetros en la plantilla se distingue el de *management* dentro de la sección *properties*, el cual si se encuentra en *true* permitirá que el punto de conexión sea accesible para el usuario. Por otro lado se observan el parámetro *order*, el cual permite definir el orden de un CP dentro de un VDU cuando hay varios CPs presentes, y el parámetro *anti_spoofing_protection*, el cual solo es aplicable cuando el CP es un NIC virtual.

  Por otro lado, en el apartado *requirements* se definirán los VL y los VDU a los que va conectado dicho CP en los apartados *virtualLink* y *virtualBinding* respectivamente.
- VL: Las conexiones virtuales son la parte encargada de proporcionar conectividad entre las distintas VDU. Los parámetros más importantes a tener en cuenta en la plantilla en lo referente a los VL son *network_name*, el cual proporciona el nombre de la red donde se encuentra conectado dicho VL, y el *vendor*, en el cual se detalla el proveedor que genera dicho VL.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Demo with user-data

metadata:
  template_name: sample-vnfd-userdata

topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            num_cpus: 1
            mem_size: 512 MB
            disk_size: 1 GB
      properties:
        image: cirros-0.3.5-x86_64-disk
        config: |
          param0: key1
          param1: key2
        mgmt_driver: noop
        user_data_format: RAW
        user_data: |
          #!/bin/sh
          FILE=$(ifconfig | head -n 1 | cut -f 1 -d ' ')
          mkdir exp
          watch -n 60 "date >> exp/$FILE; ifconfig | grep eth0 -C 5 | egrep 'RX|TX' > exp/$FILE" &

    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        order: 0
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1

    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net_mgmt
        vendor: ACME
```

Figura 4: Plantilla VNFD. Fuente: Mini-nfv

2.3.2 Infraestructura de Virtualización de Funciones de red (NFVI)

Uno de los bloques definidos por la ETSI para esta tecnología es el encargado de la infraestructura (NFVI). Dicha capa es en la que se localiza el hardware sobre el que se realiza la virtualización y del que se obtienen los recursos tanto hardware como software utilizados por las máquinas virtuales instanciadas.

La NFVI de un sistema puede estar compuesta por numerosos dispositivos físicos, los cuales son implementados y administrados como una única entidad que sustenta el entorno de ejecución e instanciación de las VNF.

La capa NFVI está compuesta por múltiples puntos conocidos como los *NFVI Point of Presence* (NFVI-PoP), en estos puntos es donde los proveedores de servicios implementarán los VNF, teniendo en cuenta tanto los recursos de almacenamiento, redes y computación [8].

Una de las principales funcionalidades de la capa NFVI es la de realizar una abstracción de los recursos hardware creando una capa de virtualización que puede ser manejada o administrada lógicamente para que los VNF puedan instanciar máquinas virtuales y realizar las funciones designadas.

La red NFVI administrará mediante dispositivos de enrutamiento y conmutación tanto los recursos de almacenamiento como los recursos software contenidos en un NFVI-PoP, permitiendo a su vez la conectividad externa. La capa NFVI trabaja directamente con las VNF y con el Virtual Infrastructure Manager (VIM) junto con el Network Function Virtualization Orchestrator (NFVO).

2.3.2.1 VNF Forwarding Graphs

Según el estándar definido por TOSCA, VNF Forwarding Graphs (VNFFG) es una función empleada para administrar y orquestar el tráfico que pasa a través de las VNF. Es decir, dicha función crea grafos que enlazan múltiples VNF mediante connection points (CP) y virtual links (VL) para orquestar y determinar que el tráfico deberá pasar por dicha cadena de VNFs, creando las cadenas conocidas como Service Function Chains (SFC).

Estas cadenas de servicios de red virtuales presentan como ventaja la automatización en la configuración de las conexiones de red virtual para manejar diferentes tipos de flujos de tráfico, utilizando de manera eficiente la capacidad de programación proporcionada por las redes definidas por software.

Los VNFFG se instancian y se definen en los VNF Forwarding Graphs Descriptor (VNFFGD), cuyos parámetros se pueden observar en la figura 5.

En las plantillas VNFFGD el único elemento o tipo de nodo son las *Forwarding_graphs*. Este elemento es el encargado de describir el flujo o cadena de VNF que se seguirá en la transmisión de datos, así como el clasificador que se encargará de formar una ruta determinada entre un conjunto de VNF.

Dentro de los *Forwarding graphs* una de las secciones más importantes en una plantilla es la de *properties*. Dentro de esta sección encontramos *id*, que describe el identificador de la ruta, *policy*, que describirá las políticas de coincidencia de tráfico, y *path* que definirá la ruta y la cadena de VNF.

Dentro del subapartado *policy* se encuentran algunas de las políticas aplicables en el flujo VNFFG, como por ejemplo políticas aplicadas por IP en los parámetros *ip_dst_prefix* e *ip_src_prefix*, políticas por protocolo con *ip_proto* o políticas por puerto con *destination_port_range*.

Por otro lado, dentro del subapartado *path* se encuentran los nodos que se deben atravesar en el flujo creado. Cada nodo irá definido en el apartado *forwarder* e irá asociado a un CP en el apartado *capability*, pudiendo seleccionar 2 CP, uno para la entrada y otro para la salida de tráfico.

Dentro de la sección *groups* es donde se definirá el VNFFG propiamente dicho. En el subapartado *properties* se definen parámetros como el *vendor*, la versión o el número de puntos de conexión. No obstante, algunos de los más importantes son el de *constituent_vnfs* y el de *members*, donde se describen los VNF constituyentes y las rutas o *Fowarding graphs* que forman parte de este VNFFG, respectivamente.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP12->CP12)
      properties:
        id: 51
        policy:
          type: ACL
          criteria:
            - network_src_port_id: 05754e35-828a-4f06-8d46-35c4d333d05f
            - network_id: 3d582b05-b996-4df2-9b3f-a1a26b2f1383
            - ip_proto: 6
            - destination_port_range: 80-80
            - ip_dst_prefix: 192.168.120.2/24
            - ip_src_prefix: 192.168.120.1/24
        path:
          - forwarder: vnfUD
            capability: CP111

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL11]
        connection_point: [CP111]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path1]
```

Figura 5: Plantilla VNFFGD. Fuente: Mini-nfv

2.3.3 Network Functions Virtualisation Management and Orchestration (NFV MANO)

La tercera capa o bloque definido por la ETSI referente a la estructura de una red o solución NFV es la de NFV MANO. Dicha capa recoge las funcionalidades de los elementos NFVO, VIM y Virtual Network Functions Manager (VNFM).

Según lo establecido por la ETSI se trata de un marco ideado para la administración y la orquestación de todos los recursos disponibles de un centro de datos virtualizado, en los cuales se incluyen el almacenamiento, las redes y la computación. Este bloque funcional será el encargado de gestionar todos los recursos de red de las aplicaciones basadas en la nube y de coordinar y administrar el ciclo de vida de las distintas funciones de red VNF.

Para la correcta conexión y comunicación de los distintos elementos que componen esta capa es necesaria la integración con Application Programming Interfaces (API). Para la estandarización de esta capa se creó la división Open Source MANO (OSM) como parte de la ETSI. Por otro lado,

la Fundación Linux creó otra plataforma de código abierto referente en el sector, la Open Platform for NFV Project (OPNFV) [9].

2.3.3.1 Function Virtualization Orchestrator (NFVO)

El orquestador forma una parte fundamental de la arquitectura presentada por el marco NFV MANO. Se trata de la pieza clave en la administración de los Network Services (NS), VNF y NFVI.

Por lo tanto, las funciones del orquestador podrían resumirse en tres funciones principales:

- El orquestador será el encargado de incorporar nuevos paquetes y plantillas de implementación de servicios de red (NS), coordinando a su vez el ciclo de vida de este servicio. Esto incluye la gestión de sus actualizaciones, escalado, consultas, medición de rendimiento y finalización de la instancia sobre la que se sustenta dicho servicio. Para la adopción e implementación de una determinada plantilla el orquestador siempre validará la autenticidad de la misma así como la validez de la información que esta contiene, cumpliendo con el estándar establecido.

- Como ocurría con los NS, el orquestador será el encargado de administrar los distintos paquetes de VNF que surgen a su vez como consecuencia de las plantillas de implementación de NS. Del mismo modo, se necesita una validación previa de dichos paquetes, comprobando su integridad y su autenticidad. Por otro lado, mediante el uso del repositorio de instancias NFV se velará por la correcta comunicación entre los NS y las VNF, evaluando sus políticas y coordinando junto con los VNFM la instanciación de las VNF necesarias y gestionando sus ciclos de vida.

- El orquestador será el encargado de gestionar los recursos del NFVI, atendiendo y validando las peticiones de solicitud de dichos recursos mediante uno o varios VIM. Generalmente dichas peticiones vendrán de los VNFM, las cuales si son aprobadas se asignarán recursos entre los múltiples NFVI PoPs. Para realizar dicha función con la mayor eficiencia se facilitará el VIM más adecuado a cada VNFM a la hora de atender una de estas solicitudes de asignación de recursos. Finalmente, el orquestador obtendrá información del uso de los recursos en uso por las VNF instanciadas, velando por el cumplimiento de las distintas políticas.

2.3.3.2 Virtualized Network Function Manager

El VNFM es el siguiente componente dentro del bloque de NFV MANO. El manager será aquel elemento encargado de la gestión del ciclo de vida de las distintas VNF instanciadas y bajo el control del NFVO interactuando a su vez con el VIM mediante el envío de comandos.

Cada VNF instanciada estará asociada a un VNFM, pero dicho VNFM podrá administrar varias VNF. No obstante, las funciones realizadas por el VNFM son las siguientes:

- Instanciación y configuración de los VNF.
- Actualizar, escalar y analizar el rendimiento de las VNF instanciadas.

- Controlar el ciclo de vida de las distintas VNF y finalizarlas cuando sea preciso.
- Interaccionar con NFVO y VIM para el control de los recursos asignados a cada una de las VNF instanciadas.
- Comprobar la disponibilidad de los recursos y si es factible la instanciación de las VNF solicitadas.
- Control de los distintos Key Performance Indicator (KPI) que miden el desempeño de la red.

Dentro del grupo de VNF que pueden ser instanciadas se conocen funcionalidades genéricas. No obstante, el VNFM debe adecuarse al marco referencial para poder soportar aquellas VNF que requieran de funcionalidades específicas, las cuales vienen especificadas en el paquete de VNF que se ha utilizado en su instanciación.

2.3.3.3 Virtualised Infrastructure Manager (VIM)

El tercer elemento que forma parte de la capa NFV MANO es el Virtualised Infrastructure Manager (VIM). Este elemento es el responsable de gestionar la capa NFVI, concretamente de administrar los recursos de almacenamiento, de red y de software dentro del dominio del proveedor de servicios.

Este elemento puede presentar distintas funcionalidades dependiendo de su especialización, así puede estar dedicado a un tipo exclusivo de recurso del NFVI o bien puede tener la capacidad de gestionar distintos tipos.

Las principales funciones de este elemento son las siguientes:

- Gestionar los recursos de virtualización y su capacidad, asociándolos a los recursos físicos de red, almacenamiento y cómputo de software, manteniendo un control de esta asignación de los recursos virtuales a los recursos físicos.

- Control del rendimiento de los recursos software, hardware y virtuales notificando los fallos que puedan afectar a las asociaciones vigentes con los recursos virtualizados.

- Creación y gestión de los enlaces virtuales, redes, subredes y puertos de los VNFFG así como de las políticas de seguridad que estos emplean para mantener un correcto control de acceso de red o tráfico de reenvío.

- Control sobre recursos útiles para otros elementos de la capa NFV MANO, como por ejemplo las imágenes software, gestionando los repositorios que contienen estos datos y recursos para agilizar así la asignación de los mismos.

2.4 OASIS TOSCA

Con el auge de este tipo de tecnología y con la computación en la nube surgió la necesidad de estandarizar dichos servicios se hizo cada vez mayor. Es por ello que surgió el estándar TOSCA desarrollado por OASIS. Con la aparición de dicho estándar lo que se buscaba era un aumento en la escalabilidad y flexibilidad de estos servicios, pudiendo portarlos entre diferentes entornos con independencia del proveedor, mejorando así la automatización y la interoperabilidad.

OASIS es una organización sin ánimo de lucro cuya tarea es la de desarrollar estándares abiertos, los cuales son adoptados y respaldados por la mayor parte de los proveedores y del mercado de las telecomunicaciones.

Con la aparición de TOSCA se definieron plantillas de servicio que definían los componentes y las relaciones entre estos a la hora de establecer o definir los servicios e infraestructuras que se emplean en la nube, así como las diferentes políticas que rigen sobre dicha tecnología. TOSCA ha pasado por distintas versiones que usaban lenguajes de programación diferentes, siendo el primero de ellos el lenguaje XML para posteriormente ser adoptado el lenguaje YAML en el año 2016.

Por otro lado, a parte de las especificaciones expuestas en el estándar surgen a su vez proyectos de código abierto que ayudan a reducir la brecha en las especificaciones, lo que conlleva una rápida corrección y crecimiento del estándar y del sector de las redes NFV.

El ETSI, se encuentra actualmente trabajando en TOSCA Simple Profile for NFV, habiendo lanzado en el año 2018 la primera versión de ETSI GS NFV-SOL 001 [10], proporcionando con dicho estándar las bases para un ecosistema abierto.

Con la utilización del lenguaje de programación YAML se implementarán las distintas plantillas VNFD así como los VNFFGD que se encuentran integrados y definidos en la plantilla Network Service Descriptor (NSD).

En la figura 6 se puede observar la estructura clásica de un servicio de red definido mediante el lenguaje TOSCA, en la que se observa que los servicios tienen una estructura única donde los nodos forman parte de la plantilla de nodos y donde se definen las relaciones entre ellos.

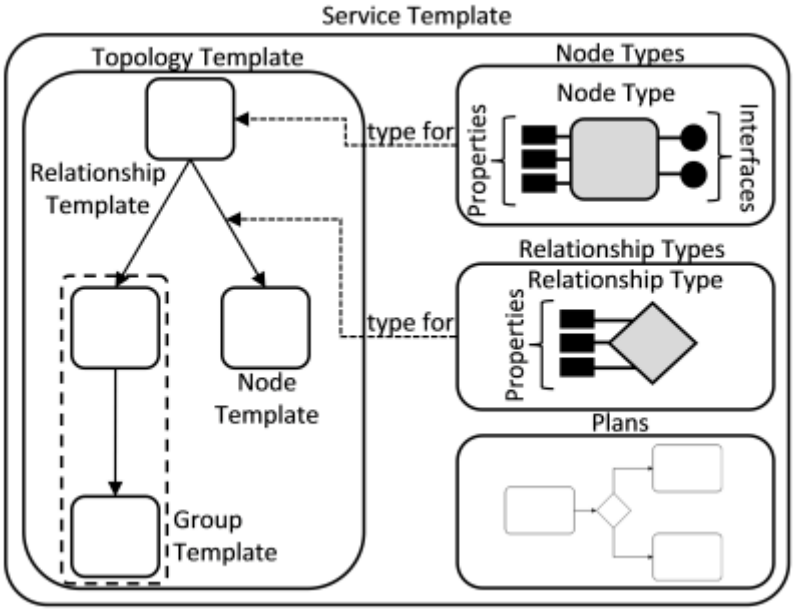

Figura 6: Elementos estructurales plantilla de servicio de red. Fuente: Cloudify.co.

2.5 NFV: Ventajas y retos

Las redes con estructura NFV suponen una alternativa competitiva y una clara evolución en el desempeño de las redes de comunicación. Se trata de una serie de tecnologías que están siendo cada vez más adoptadas en el sector debido a sus numerosas ventajas y beneficios.

Los principales beneficios que han incrementado la popularidad de este tipo de redes quedan recogidos en la lista siguiente:

- Las redes NFV proporcionan una mayor independencia en cuanto a hardware propietario o dedicado para la implementación de servicios, lo que aumenta considerablemente las posibilidades y conlleva a una reducción del uso del espacio físico y la necesidad de personal especializado.

- Reducción en el consumo de recursos, tanto eléctricos como materiales que conllevan a una reducción de los costes asociados al mantenimiento y puesta en marcha de la red y de los servicios.

- Entorno más dinámico y escalable, con una mayor capacidad de adaptación a los nuevos avances tecnológicos. A su vez, esta tecnología proporciona una mayor rapidez en la implementación de nuevos servicios.

- Posibilidad de ejecutar numerosas funciones y aplicaciones de red sobre un único dispositivo, consiguiendo a su vez una mayor automatización en la incorporación, aprovisionamiento y activación de nuevas funcionas de red consiguiendo así una mayor eficiencia operativa.

- Alta fiabilidad con un alto rendimiento, el cual puede ser aprovechado por aplicaciones que requieren de una baja latencia y de un funcionamiento óptimo en tiempo real.

- Flexibilidad en la ejecución de VNF, pudiendo ser ejecutados en diferentes servidores y trasladados a otros según las necesidades de ejecución y los recursos disponibles en la red.

La aparición de nuevas tecnologías y el avance de la virtualización suponen grandes retos a los que se busca enfrentar mediante la tecnología NFV. Algunos de los retos de esta tecnología vienen marcados por las dificultades en la interoperabilidad y fiabilidad de estas redes durante el periodo de migración hacia las mismas.

Es por ello que algunos de los objetivos que se persiguen con la aplicación de este nuevo concepto en las redes de comunicaciones están fuertemente relacionados con el análisis predictivo de estas redes, lo que permitiría un mayor control sobre las mismas y lograr una mayor eficiencia en cuanto a la utilización y asignación de recursos y la seguridad, pudiendo monitorizar a tiempo real y de una forma unificada todo lo que sucede en la red.

2.6 Escalado de VNF

Uno de los mayores retos que presenta cualquier red que implemente la tecnología NFV es el de la gestión de los recursos y como gestionar estos de la forma más eficiente ante la fluctuación del tráfico.

Para la consecución de dichos objetivos el concepto que se plantea es el de la escalabilidad de las VNF en función de las necesidades de la red, haciendo así un uso más eficiente de los recursos presentes en la misma.

Usando un modelo de red predictivo la red será capaz de reaccionar instanciando VNFs en función de los índices de flujo estimados, lo que permitirá reaccionar de manera proactiva ante los cambios que se produzcan en la red.

Los recursos de las VNF vienen definidos en las plantillas VNFD tal y como se ha expuesto en apartados anteriores. Sin un sistema de escalabilidad de los VNF, estos presentarían dos funcionamientos distintos, por un lado, si los VNF se contemplasen para cubrir una determinada función de red, pero sus recursos asignados se viesen superados esto supondría una suspensión del servicio, por otro lado, si estos VNF fuesen concebidos para exprimir al máximo sus recursos se conseguiría el efecto contrario, habiendo recursos infrautilizados.

Por ello nace la necesidad de escalabilidad de las VNF, pudiendo ser instanciadas bajo demanda y dependiendo del análisis predictivo de la red, pudiendo hacer frente a las distintas necesidades de la misma de una forma eficaz reduciendo los costos de operación.

Este escalado sería controlado por el NFVO y el VNFM, los cuales serían los responsables de instanciar las VNF y de reservar y solicitar los recursos necesarios dinámicamente a través de los VIM a la capa NFVI, la cual contestará a dichas peticiones otorgando la cantidad de recursos necesarios para la instanciación de dichas VNF.

En el sentido inverso, para finalizar el ciclo de vida de una determinada VNF, el VNFM se comunica con el NFVO y este a su vez con el VIM y la NFVI para liberar los recursos de dicha VNF, terminando así el ciclo de vida de dicha VNF y produciéndose la liberación de dichos recursos de red. En la figura 7 puede observar el flujo entre las distintas partes de la arquitectura para la creación y posterior eliminación de las VNF.

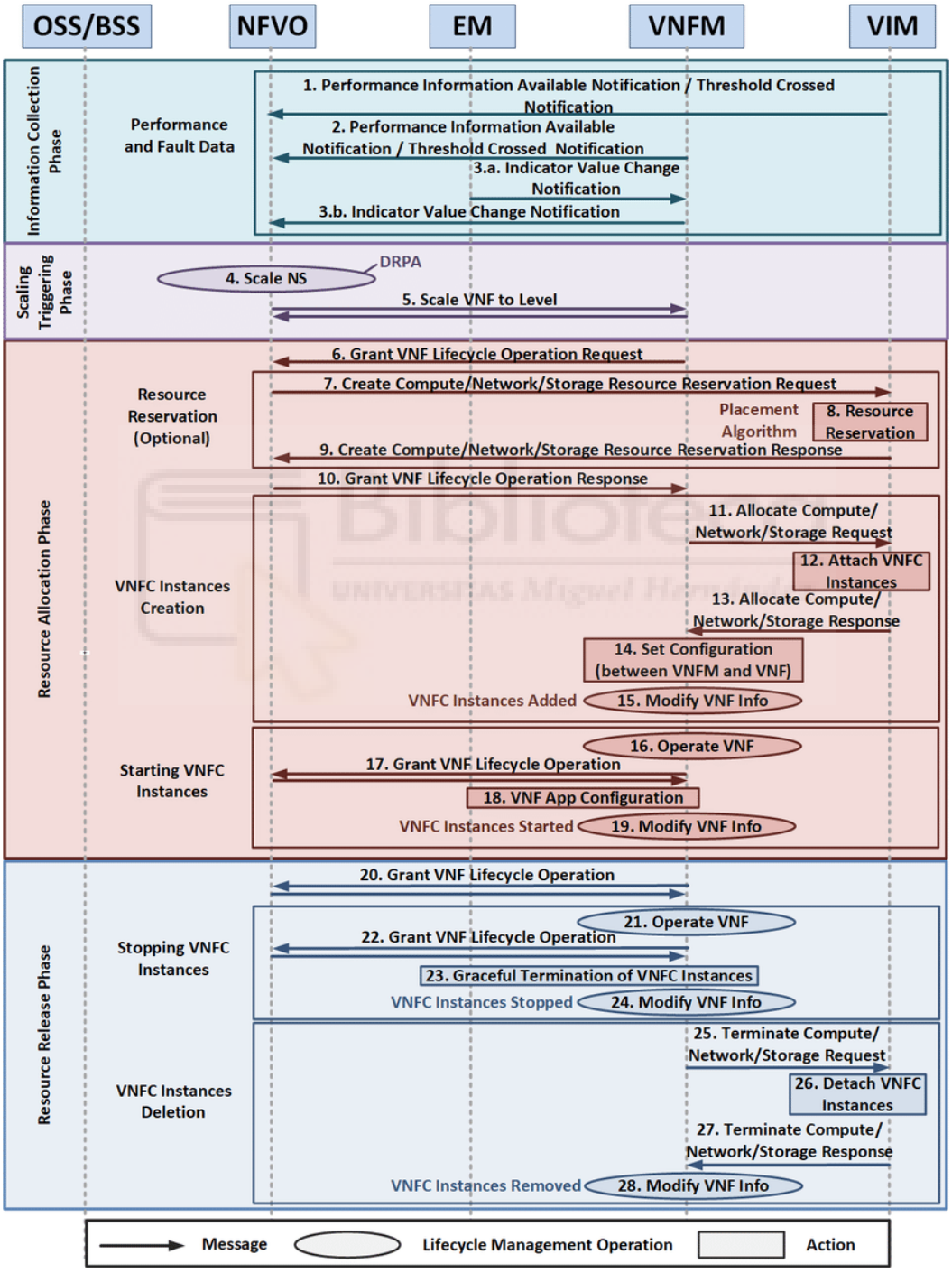

Figura 7: Flujo de creación de una VNF. Fuente: Oscar Adamuz-Hinojosa

Capítulo 3: Herramientas

3.1 Mininet

Para la emulación y creación de la red SDN la herramienta empleada es Mininet. Se trata de un poderoso emulador empleado para el despliegue de redes sobre los recursos que el hardware o máquina virtual hospedadora ofrece.

Mediante Mininet, se pueden emular redes SDN definiendo hosts virtuales, conmutadores, controladores y enlaces. Los hosts instanciados por Mininet utilizan software de red Linux mientras que sus conmutadores posibilitan el uso de OpenFlow, permitiendo así un enrutamiento flexible y personalizado.

Mininet posibilita la creación y la experimentación con distintos tipos de topología en un marco que emula de manera realista el funcionamiento correcto de cualquier tipo de sistema, ejecutando código real como las aplicaciones de red estándar de Linux/Unix. A su vez, se trata de una alternativa rápida, económica, de fácil instalación y que permite hacer pruebas con un mayor ancho de banda.

Según lo recogido en el portal oficial de Mininet la aplicación se caracteriza por [11]:

- Ofrece un banco de pruebas de red simple y económico que permite desarrollar aplicaciones OpenFlow.
- Posibilita realizar pruebas de regresión a nivel del sistema.
- Permite el trabajo simultáneo de varios desarrolladores sobre la misma topología de forma independiente.
- Proporciona una API de Python que permite crear y experimentar con redes de forma sencilla y extensible.
- Posibilidad de probar topologías complejas sin necesidad de conectar una red física.
- La herramienta cuenta con una CLI que permite ejecutar pruebas en la red de forma dinámica ya que reconoce tanto la topología como OpenFlow.

Mininet hace uso de un solo kernel para aplicar la virtualización basada en procesos, además con la versión de Linux 2.2.26 se admitieron los espacios de nombres de red para proporcionar a los procesos individuales interfaces de red, tablas ARP independientes y tablas de enrutamiento. Por otro lado, Mininet hace uso de conexiones ethernet virtuales para conectar los distintos hosts y conmutadores que formen parte de la topología.

Sin embargo, Mininet también presenta sus limitaciones, las cuales son que no puede sobrepasar la CPU o el ancho de banda disponible en un solo servidor además de que no podrá utilizar aplicaciones que no sean compatibles con el sistema operativo Linux. Estas limitaciones realmente estarían más estrechamente relacionadas a la infraestructura donde se despliega Mininet, mostrando una gran dependencia de la misma.

Por otro lado, Mininet es una herramienta capaz de ejecutar cualquier script de Python, pudiendo así hacer uso de diferentes funcionalidades aplicables a la topología o entorno de red simulado.

3.2 Oracle VM VirtualBox

El software de virtualización que se empleará para ejecutar el entorno desarrollado en este proyecto es Oracle VM VirtualBox. Se trata de una herramienta de virtualización x86 y AMD64/Intel64 de código abierto bajo los términos de la GNU General Public License (GPL) versión 2. Dicho software se ejecutará sobre un host Windows, aunque también es compatible con otros sistemas como Linux y Macintosh.

Según lo recogido en su web oficial VirtualBox es un virtualizados completo de propósito general para hardware x86, dirigido a servidores, computadoras de escritorio y uso integrado [12].

Algunas de las ventajas y características que ofrece esta herramienta son las siguientes [13]:

- Simplificación de las operaciones: Se trata de un software fácil de adoptar y de usar, con una interfaz gráfica intuitiva y que funciona en cualquier escritorio, reduciendo a su vez el coste de operativo de los equipos.

- Automatización de las implementaciones en la nube: Esta herramienta mediante su interfaz gráfica facilita a los desarrolladores la importación y exportación de máquinas virtuales en formato OVF en la nube.

- Acceso remoto seguro: VirtualBox ofrece claves de cifrado de 256 bits para proteger las conexiones remotas a aplicaciones limitadas o restringidas debido a su carácter sensible, proporcionando de esta forma entornos de trabajo completamente seguros y cifrados.

A la hora de configurar el adaptador de red de una máquina virtual VirtualBox ofrece las siguientes opciones:

- Modo solo anfitrión: El sistema operativo anfitrión que aloja las máquinas virtuales podrá conectarse a ellas. Por otro lado, también podrá conectar distintas máquinas virtuales pero no proporcionará acceso a internet.

- Modo adaptador puente: La máquina virtual se conectará a la misma red que el sistema anfitrión, pudiendo así acceder a internet bidireccionalmente.

- Modo NAT: Las máquinas virtuales instanciadas tendrán acceso a internet pero para establecer la conexión desde internet se realizará el mapeo de puertos mediante NAT. Las máquinas virtuales no tendrán contacto entre sí. Para conectar las máquinas virtuales entre sí se necesitaría el modo de red NAT.

- Modo interno: Este modo solo permite la conexión entre máquinas virtuales.

3.3 Wireshark

La herramienta que se ha empleado en este proyecto para la captura y el análisis del tráfico es Wireshark. Esta herramienta es un analizador de protocolos utilizada principalmente para análisis de datos y protocolos.

Con esta herramienta puede observarse en tiempo real el tráfico que pasa por la red, pudiendo capturarlo, filtrarlo y guardarlo en disco con un formato *pcap*. Los datos capturados pueden analizarse utilizando varias métricas que la herramienta pone a disposición del usuario.

Según lo recogido en la web oficial de Wireshark, algunas de sus características son las siguientes [14]:

- Herramienta multiplataforma capaz de ejecutarse en sistemas como Windows, Linux, macOS, entre otros.
- Proporciona un análisis completo de VoIP.
- Permite una inspección profunda de un gran número de protocolos, el cual aumenta progresivamente en las nuevas revisiones de la aplicación.
- Permite la posibilidad de capturar paquetes a tiempo real y poder analizarlos posteriormente fuera de línea.
- Permite la aplicación de filtros de pantalla y reglas de coloración de paquetes.
- La salida se puede exportar a XML, PostScript, CSV o texto sin formato.
- Permite la lectura y escritura en múltiples formatos de archivo de captura diferentes: tcpdump (libpcap), Pcap NG, Catapult DCT2000, Cisco Secure IDS iplog, Microsoft Network Monitor, Network General Sniffer® (comprimido y sin comprimir), Sniffer® Pro y NetXray®, Network Instruments Observer, NetScreen snoop, Novell LANalyzer, RADCOM WAN / LAN Analyzer, Shomiti / Finisar Surveyor, Tektronix K12xx, Visual Networks Visual UpTime, WildPackets EtherPeek / TokenPeek / AiroPeek y muchos otros.
- Soporte de descifrado para muchos protocolos como IPsec, ISAKMP, Kerberos, SNMPv3, SSL / TLS, WEP y WPA / WPA2.
- Posibilidad de leer datos a tiempo real desde ethernet, IEEE 802.11, PPP / HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI y otros (según su plataforma).
- Los datos capturados se pueden analizar a través de la interfaz gráfica proporcionada por la herramienta o a través de la utilidad TShark en modo TTY.

3.4 Linux Ubuntu 18.04

El sistema operativo escogido para llevar a cabo el desarrollo de la parte práctica de este proyecto es Linux en concreto Ubuntu en su versión 18.04 debido a que ofrece una compatibilidad completa con el resto de herramientas utilizadas.

Linux es un sistema operativo de tipo UNIX y de código abierto que cuenta con multitud de distribuciones distintas, en su mayoría gratuitas. Se trata de un sistema multiusuario, multitarea y multiplataforma que cuenta a su vez con una interfaz gráfica y con un modo consola. La mayor actividad a la que se destina Linux es al control de superodenadores de alto rendimiento que cuentan con numerosos procesadores y servidores.

Linux posee multitud de elementos y componentes del entorno o proyecto GNU, como por ejemplo compiladores y entornos de escritorio. Por otro lado, destacan los repositorios, que son plataformas donde se aloja software que puede ser descargado posteriormente para el sistema operativo.

La distribución escogida es Ubuntu en su versión 18.04. Ubuntu es una distribución de Linux que está basada en Debian. Es una de las distribuciones más populares debido a su facilidad de uso y experiencia de usuario, lo que le ha llevado a alcanzar una cuota de mercado del 52% [15].

3.5 Mini-nfv

La herramienta que permitirá la simulación de un entorno NFV en mininet es mini-nfv [16]. Dicha herramienta ofrece un entorno para la simulación de un orquestador y manager de NFV, pudiendo implementar y gestionar VNFs sobre Mininet, basándose en el marco del framework ETSI MANO.

Mini-nfv mediante el uso de plantillas de OASIS TOSCA será capaz de gestionar el ciclo de vida, implementar, monitorizar, escalar y eliminar VNFs en Mininet. Además, mediante el uso de TOSCA se definirán los metadatos de VNF y se pondrán en marcha máquinas virtuales en OpenStack.

Además de lo expuesto anteriormente, mini-nfv es compatible con el motor de plantillas Jinja2 el cual destaca por su alta integración con Python, proporcionando un entorno de ejecución de espacio aislado integrado.

Las características que ofrece mini-nfv según su web oficial son las siguientes:

- Ofrece un catálogo de NFV en el que destacan descriptores de VNF, descriptores de servicios de red y descriptores de VNFFG.

- Ofrece la funcionalidad de VNFM, pudiendo controlar el ciclo de vida básico de cualquier VNF así como facilitar la configuración inicial de cualquier VNF instanciado.

- Ofrece la funcionalidad de NFVO implementado los servicios de red extremo a extremo mediante plantillas. Gestiona la política de colocación de VNF de la forma más eficiente así como el service chaining SFC descrito en el VNFFGD.

- Controla el tráfico entre las VNF pudiendo crear tráfico simétrico o asimétrico.
- Compatibilidad con platillas Jinja2 y TOSCA.
- Definición de la red a través de virtual links (VL) y connection points (CP), así como la definición de IP y MAC.
- Emulación de CPU y propiedades de *flavor* a través de CPULimitedHOst de Mininet.
- Mini-nfv ignora la RAM y el disco disponibles así como las IP flotantes.
- Por el momento no es capaz de aplicar descriptores de servicio de red (NSD) ni de monitorizar las VNF así como tampoco facilitar soporte explicito para contenedores/NETCONF/Click-based VNFs.

3.6 BMON

Una de las herramientas que se emplearán para el análisis de la red es Bmon, la cual resulta de gran utilidad a la hora de analizar parámetros como el ancho de banda y poder depurar errores, pudiendo observar los picos y la pérdida de ancho de banda en la red en tiempo real. Algunas de sus características más destacables son:

-Permite monitorizar varias interfaces de red simultáneamente.

-Monitorización en tiempo real.

-Monitorización del ancho de banda, pudiendo recoger los picos y las caídas de este de forma gráfica.

-Ofrece detalles más concretos sobre el rendimiento de la red.

Capítulo 4: Preparación del entorno

4.1 Instalación y configuración de VirtualBox

El entorno donde se desarrollará este proyecto se ejecutará sobre una máquina virtual instanciada en VirtualBox. Para la obtención de dicha aplicación se ha de acceder a la página oficial de la misma donde se encuentra para descargar de manera gratuita [17].

Las opciones de descarga son amplias dependiendo del sistema operativo en el que se vaya a instalar la aplicación. En el caso de este proyecto la máquina anfitriona es un ordenador con el sistema operativo Windows 10. En la figura 8 pueden observarse las distintas opciones de descarga.

Figura 8: Opciones de descarga VirtualBox. Fuente: virtualbox.org

Tras obtener la versión correcta según el sistema operativo anfitrión se ejecuta el instalador de la aplicación y se instala siguiendo los pasos y las opciones que este facilita:

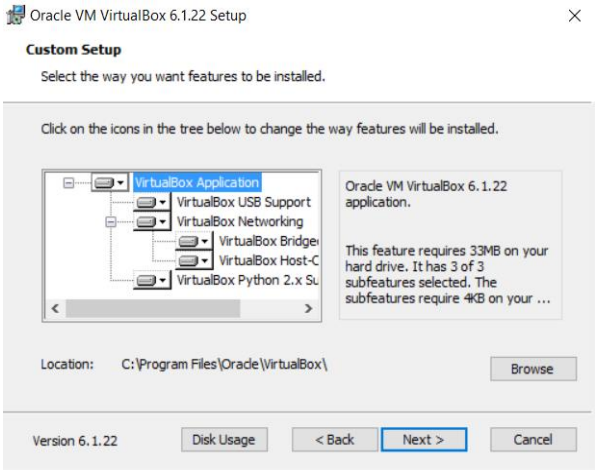

Figura 9: Instalación de VirtualBox. Fuente: VirtualBox.

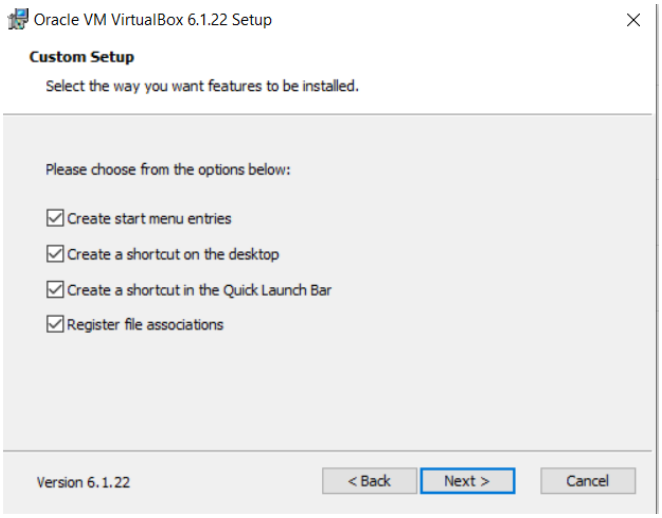

Figura 10: Instalación de VirtualBox 2. Fuente: VirtualBox

En el proceso de instalación saldrá una ventana emergente que informará de la instalación de un controlador de BUS que permitirá que VirtualBox conecte las máquinas virtuales instanciadas con los puertos USB.

Figura 11: Instalación de VirtualBox 3. Fuente: VirtualBox

Una vez instalada la aplicación al acceder a ella observaremos un panel donde se pueden observar las máquinas virtuales creadas, así como diferentes opciones para crear otras nuevas o configurar las ya existentes, tal y como se puede observar en la figura 12.

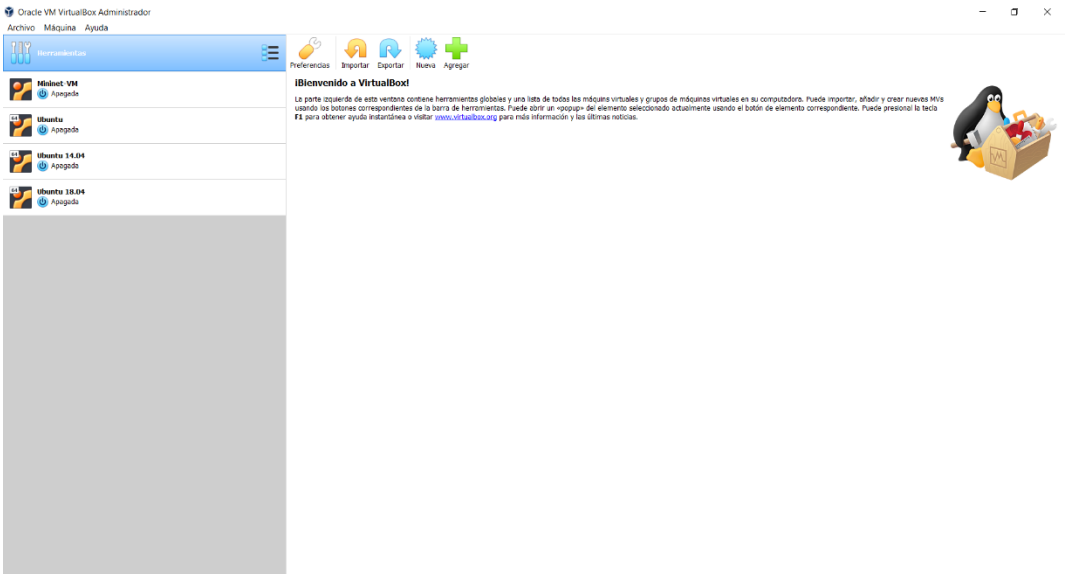

Figura 12: Panel de VirtualBox. Fuente: VirtualBox.

Para crear una nueva máquina virtual se debe pulsar en el icono azul que indica “Nueva”, tras esto obtendremos el menú que se observa en la figura 13, donde se selecciona un nombre para la máquina virtual, así como el tamaño de la memoria y el disco duro. En este caso se ha seleccionado un tamaño de memoria de 2962MB y la creación de un disco virtual.

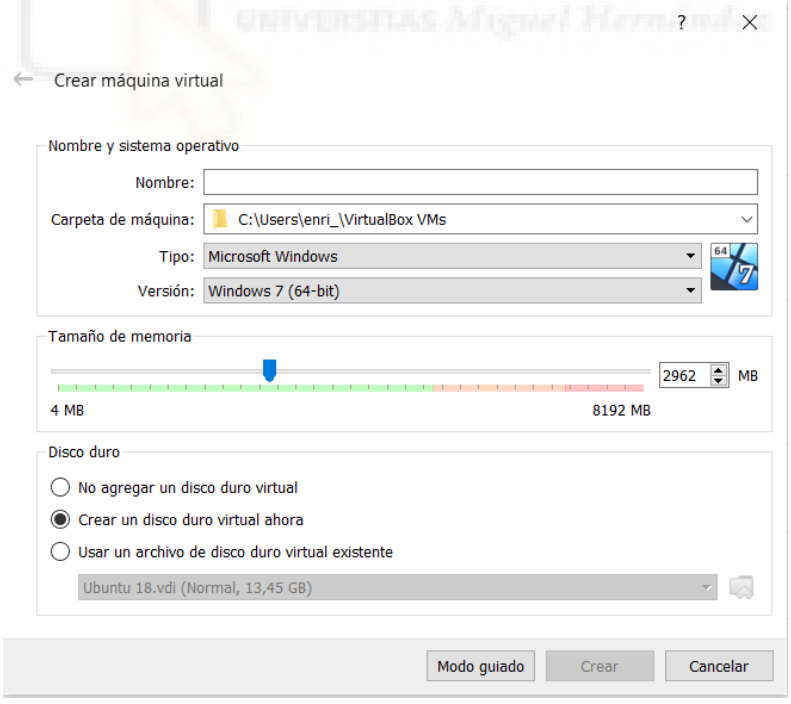

Figura 13: Creación de máquina virtual. Fuente: VirtualBox

El siguiente paso en la creación de la máquina virtual es la creación del disco duro virtual. Como puede observar en la figura 14 VirtualBox permite seleccionar el tamaño del disco que se reserva para esta máquina, pudiendo a su vez seleccionar el tipo de archivo de disco duro y si el almacenamiento es reservado dinámicamente o si por el contrario es fijo. En este caso se ha seleccionado un tamaño de 13,45GB, VDI y reservado dinámicamente.

Algunos de los sectores que se podrían ver beneficiados con la adopción de este tipo de tecnologías podrían ser el de las redes móviles, pudiendo aprovechar lo que ofrece NFV para el núcleo EPC de lTE o para las redes IMS.

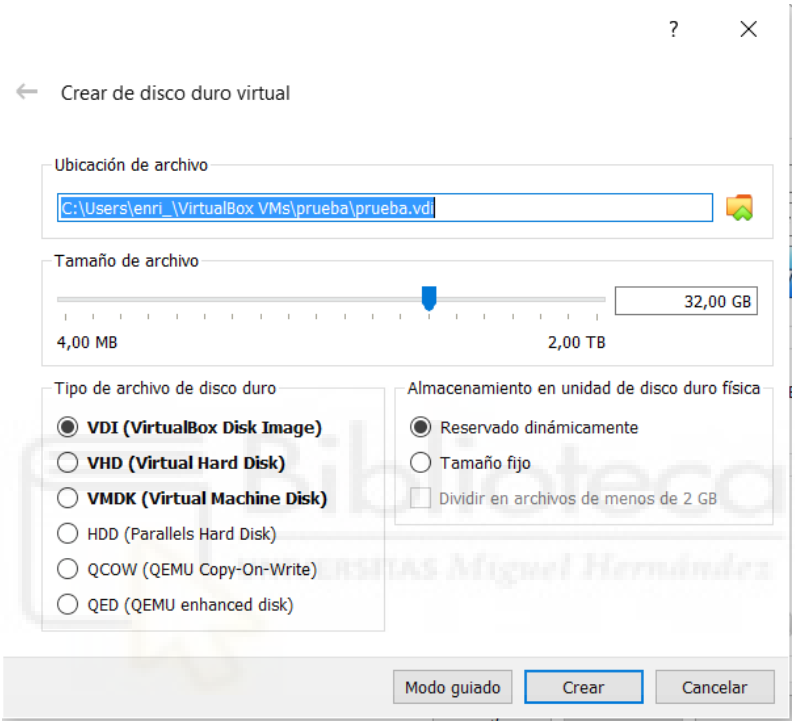

Figura 14: Creación de disco duro virtual. Fuente: VirtualBox.

Por último, se ha de configurar el adaptador de red que utilizará la máquina virtual. Para ello, una vez creada la máquina se ha de pulsar sobre ella y sobre el icono configuración. En la pantalla de configuración se ha de ir a la sección “Red”, y en esta se ha de configurar el adaptador que usará dicha máquina. En este caso se ha optado por un adaptador de tipo puente, tal y como se observa en la Figura 15.

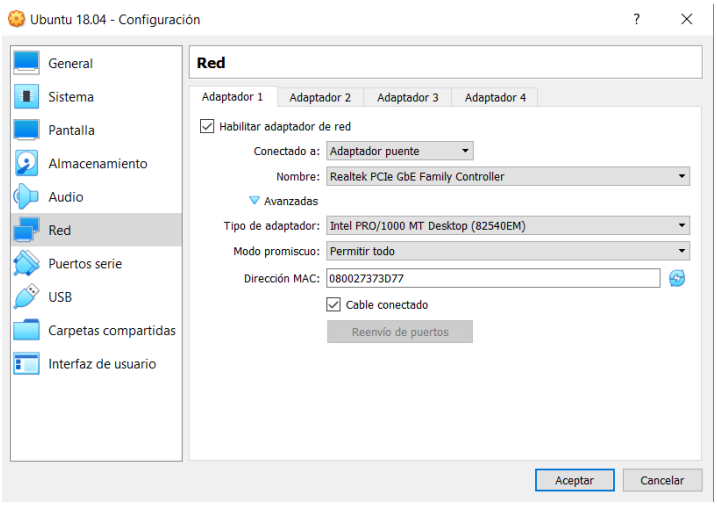

Figura 15: Configuración adaptador de red VirtualBox. Fuente: VirtualBox.

4.2 Instalación de Ubuntu

El sistema que se instalará sobre la máquina virtual creada en VirtualBox será Ubuntu en su versión 18.04. Para obtener la imagen de este sistema es preciso acceder a la página oficial de Ubuntu donde hay dos opciones de descarga, la versión de escritorio o la versión servidor, tal y como se observa en la figura 16. En este proyecto la imagen seleccionada es la versión de escritorio.

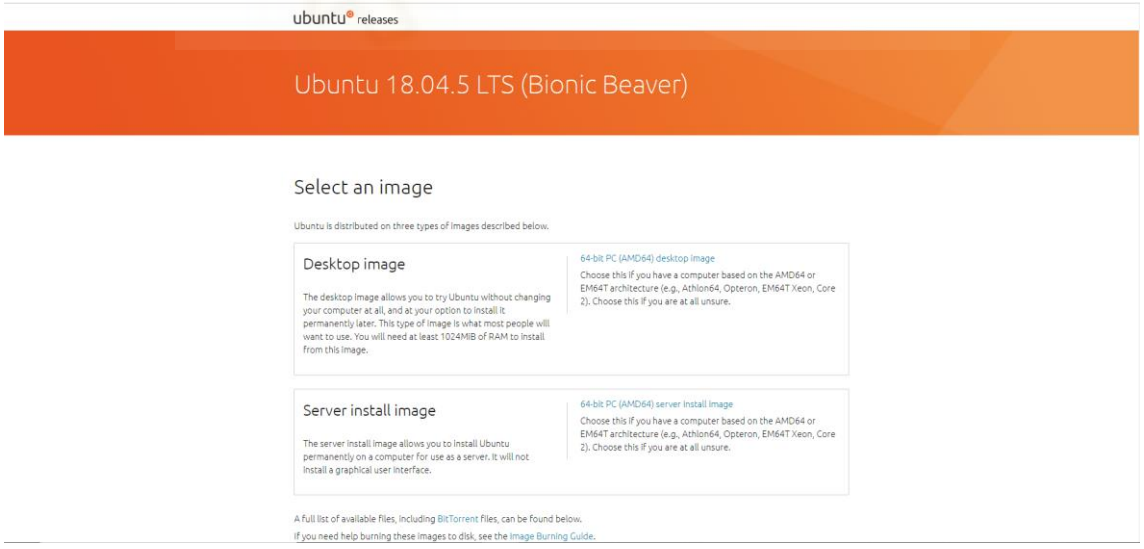

Figura 16: Descarga de Ubuntu 18.04. Fuente: Ubuntu.com

Una vez descargada la imagen hay que añadirla a la máquina virtual. Para ello se accede nuevamente a la configuración de la máquina virtual y se pulsa sobre la sección "Almacenamiento". En dicha sección se ha de ir a "Controlador: ATA" y seleccionar "Añadir disco óptico". Es en el siguiente paso donde se ha de añadir la imagen de Ubuntu descargada, tal y como se observa en la figura 17.

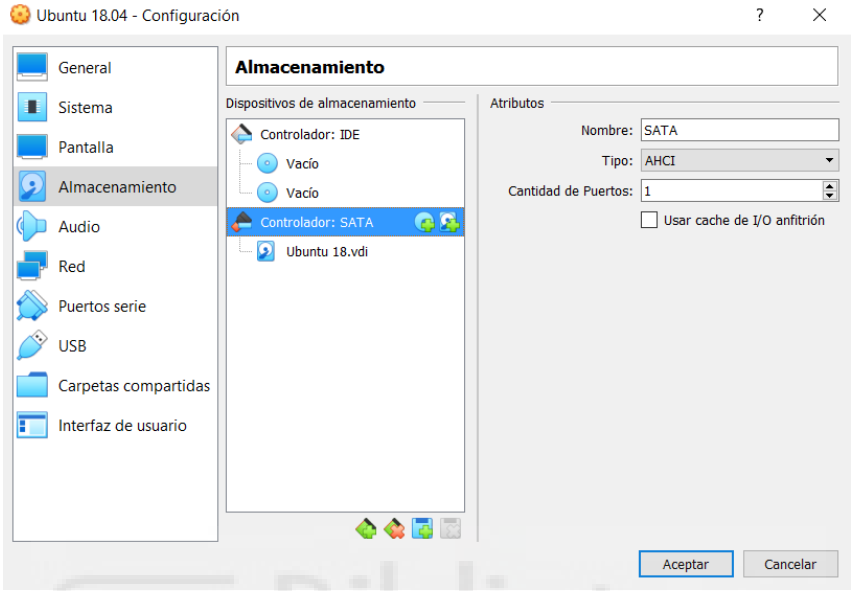

Figura 17: Configuración máquina virtual con imagen Ubuntu. Fuente: VirtualBox

Tras esto se inicia la máquina seleccionando a "Iniciar" en el menú de VirtualBox. Seguidamente se iniciará el instalador de Ubuntu, el cual facilita la instalación mostrando las diferentes opciones de una forma intuitiva en la que el usuario podrá seleccionar los ajustes que desee tal y como se observa en las figuras 18 y 19.

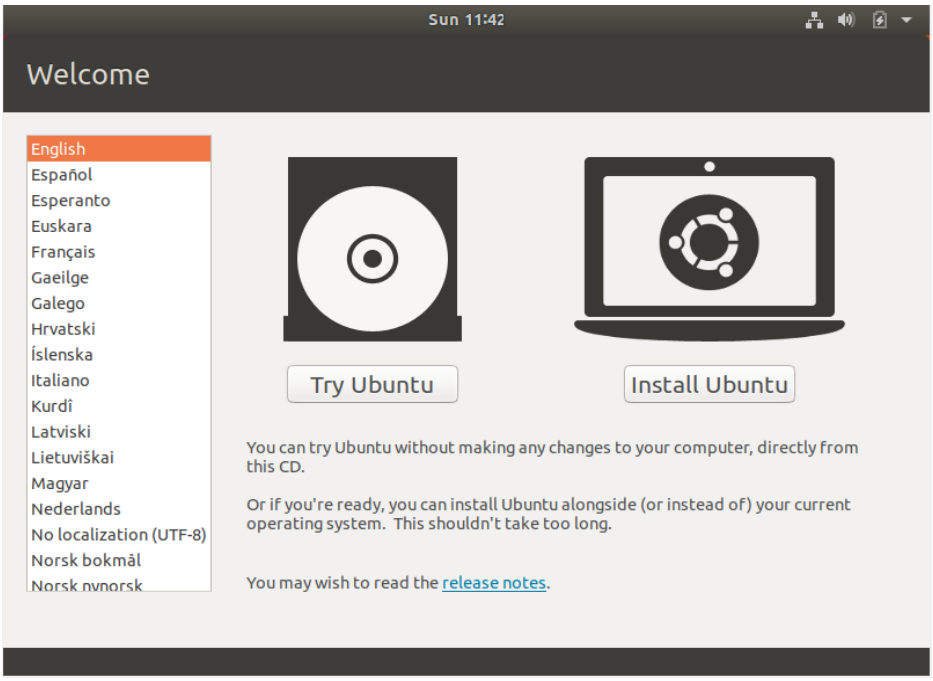

Figura 18: Instalación de Ubuntu. Fuente: Ubuntu

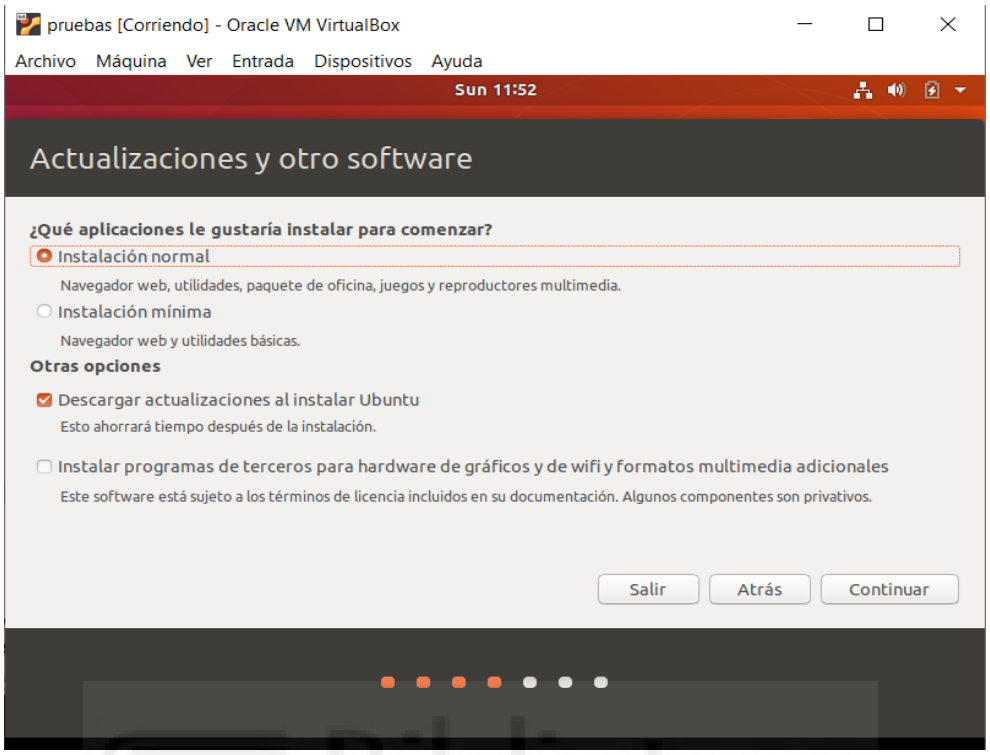

Figura 19: Escritorio Ubuntu. Fuente: Ubuntu.

Una vez finalizada esta instalación se podrá acceder al escritorio del sistema por medio del usuario configurado, tal y como se observa en la figura 20.

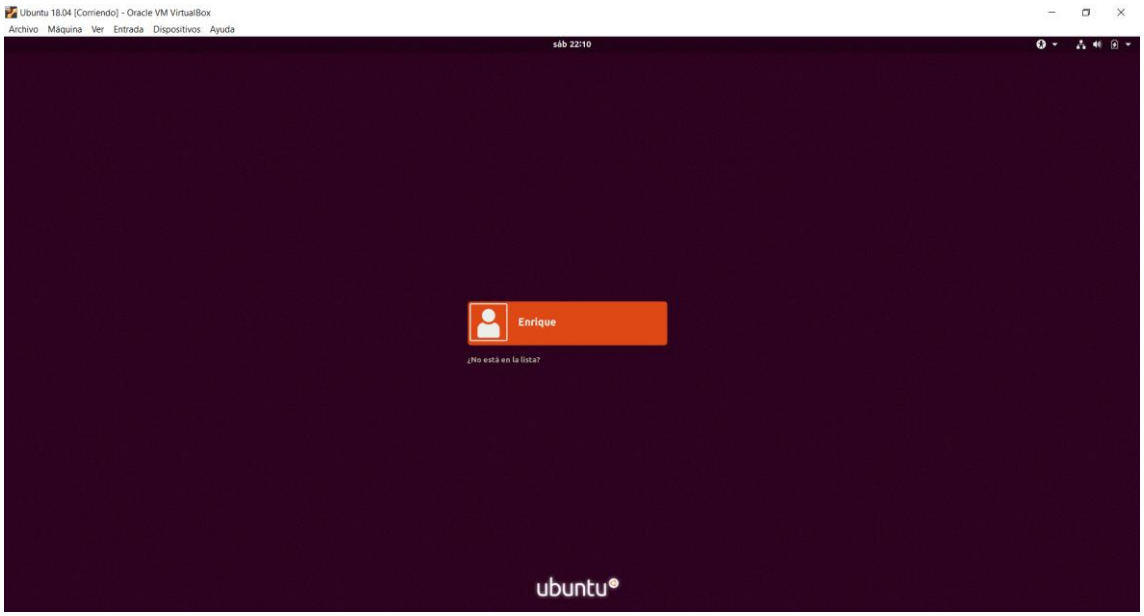

Figura 20: Escritorio Ubuntu. Fuente: Ubuntu.

4.3 Instalación de Mininet

El siguiente paso en la preparación del escenario es la instalación de Mininet, aplicación que permitirá la simulación de una red SDN pudiendo generar topologías personalizadas.

La instalación de Mininet es un proceso relativamente sencillo. El primer paso es abrir la terminal del sistema, una vez abierta hay que ejecutar el comando de descarga. En el caso de Ubuntu el comando empleado para descargar es:

```
$ sudo apt-get install mininet
```

Al ejecutar dicho comando mininet será instalado, así como todas sus dependencias, tal y como se observa en la figura 21.

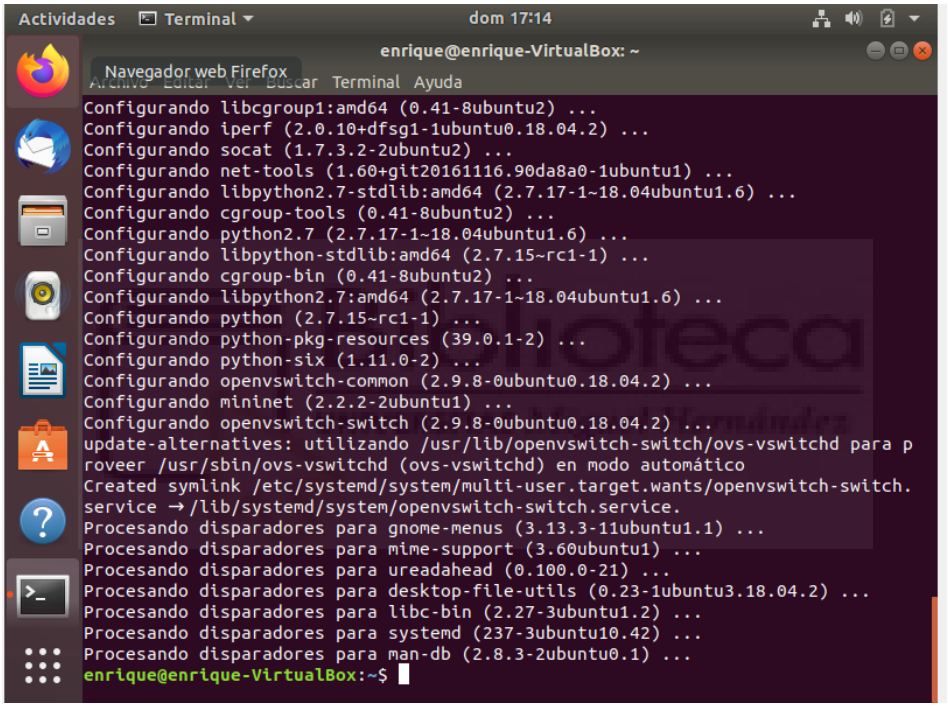

Figura 21: Instalación de mininet. Fuente: mininet

Tras esto, mininet habrá quedado instalado. Para testear su funcionamiento se ejecuta el comando:

```
$ sudo mn
```

Tras esto, mininet se ejecutará con una topología básica que viene por defecto tal y como se observa en la figura 22.

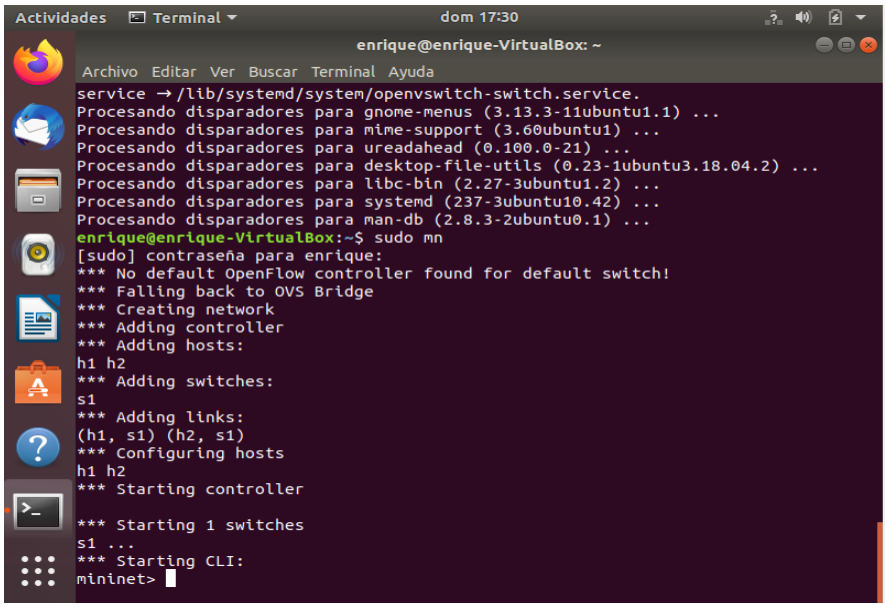

Figura 22: Iniciación de mininet. Fuente: mininet

4.4 Instalación de Wireshark

La instalación del analizador de protocolos Wireshark consta de pocos sencillos pasos. En primer lugar, para instalar dicha herramienta es preciso ejecutar el siguiente comando:

```
$ sudo apt-get install Wireshark
```

Tras esto, se instalará la herramienta junto con todas sus dependencias, tal y como se observa en la figura 23:

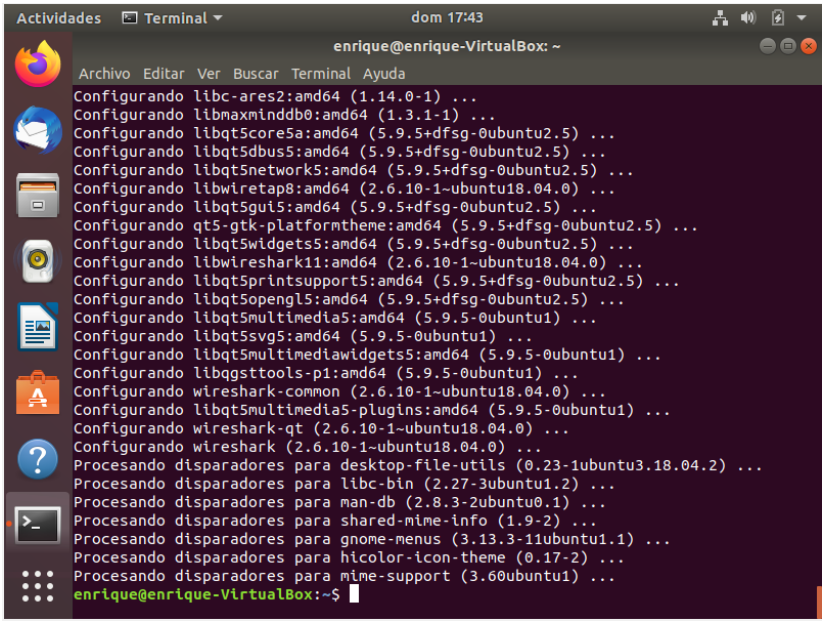

Figura 23: Instalación de Wireshark. Fuente: Wireshark

Para iniciar la aplicación se ha de ejecutar el siguiente comando:

```
$ sudo wireshark
```

Tras la aplicación del comando anterior, se abrirá la interfaz gráfica de Wireshark, tal y como se observa en la Figura 24.

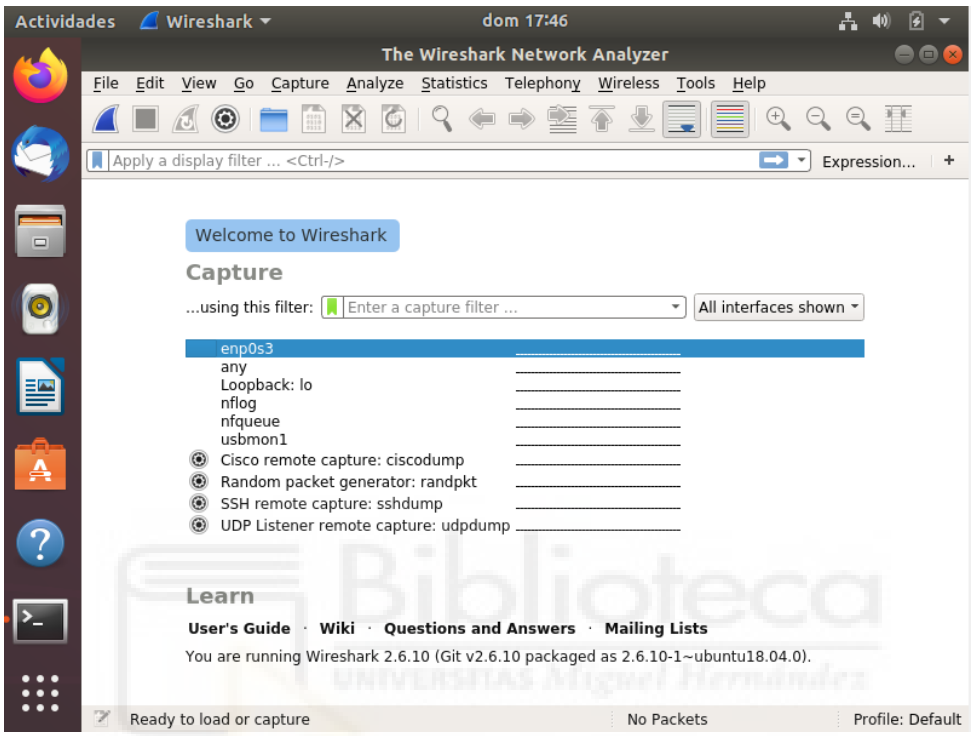

Figura 24: Iniciación de Wireshark. Fuente: Wireshark

4.5 Instalación Mini-nfv

Tras la instalación de todas las herramientas, el último paso en la preparación del entorno de pruebas es la descarga e instalación de mini-nfv. Para obtener dicha aplicación y descargar su código es preciso acceder a su página oficial, tal y como se observa en la figura 25.

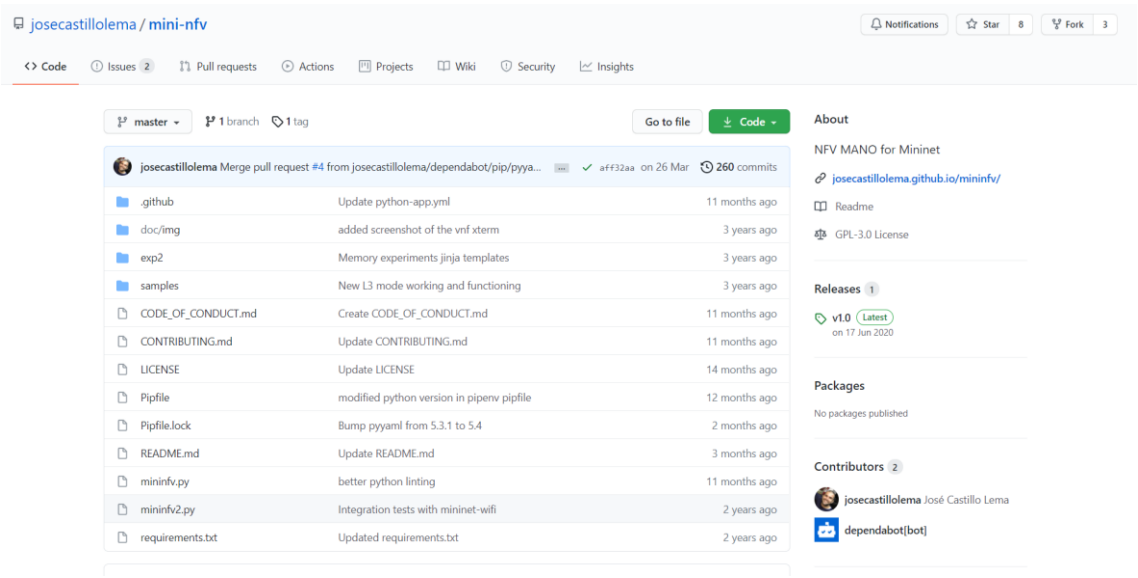

Figura 25: Descarga de mini-nfv. Fuente: github.com/josecastillolema

La aplicación requiere de varias dependencias especificadas en el fichero "requirements.txt". Para poder instalar directamente dichas dependencias desde dicho fichero será necesario el comando pip. Para poder hacer uso de dicho comando se ha de emplear el siguiente comando:

```
$ sudo apt install python-pip
```

Tras la instalación de dicho comando se procede a instalar las dependencias recogidas en el fichero indicado anteriormente desde la carpeta en la que se encuentra dicho fichero. El comando de instalación es:

```
$ pip install -r ./requirements.txt
```

Tras esto, ya se podrá acceder a la aplicación y operar sobre ella, ejecutando el siguiente comando dentro del directorio en el que se encuentra el ejecutable:

```
$ sudo ./mininfv.py
```

No obstante, es recomendable que en Ubuntu 18.04 se pare el controlador *openvswitch-testcontroller*, como se indica en el siguiente comando:

```
$ sudo systemctl stop openvswitch-testcontroller.service
```

Tras la ejecución de dicho comando se abrirá la aplicación mini-nfv, pudiendo operar completamente sobre ella, tal y como se observa en la figura 26.

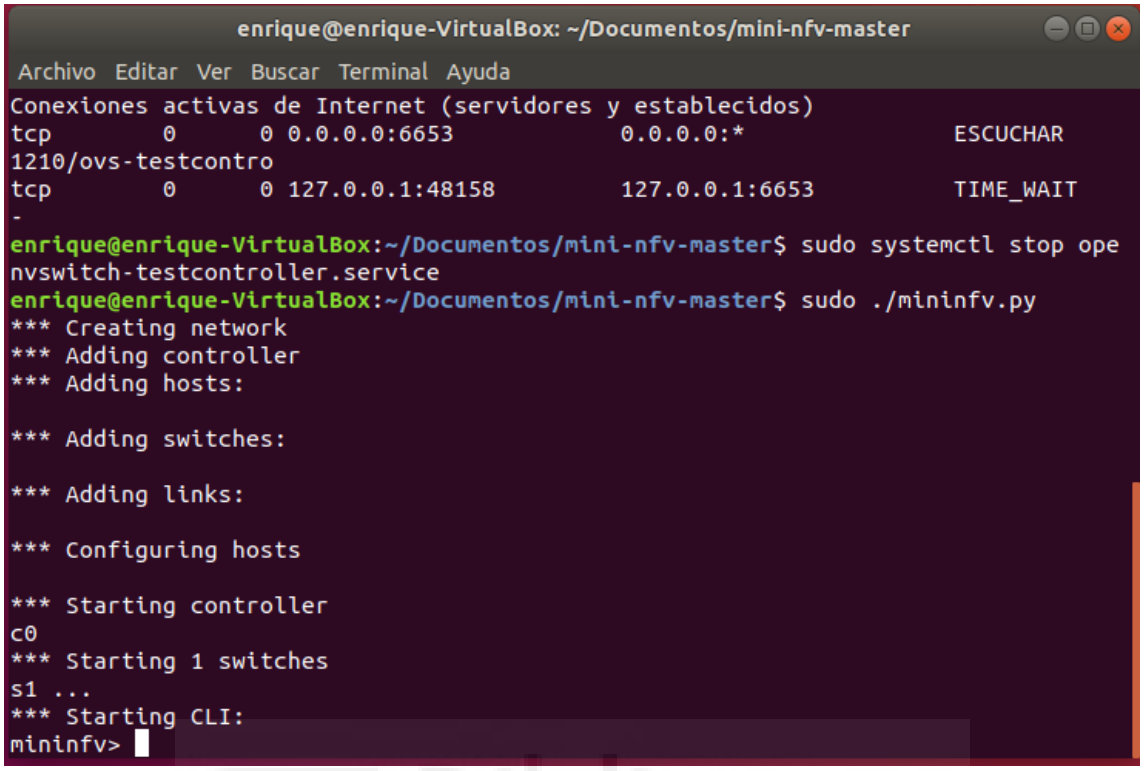

Figura 26: Iniciación de mini-nfv. Fuente: mini-nfv

4.6 Instalación Bmon

La instalación del analizador de red Bmon es relativamente sencilla. Para instalar el programa en el equipo es preciso ejecutar el siguiente comando:

```
$ sudo apt-get install bmon
```

Tras esto se solicitará instalar una serie de dependencias para el correcto funcionamiento de dicha herramienta, a lo que se ha de aceptar.

Capítulo 5: Pruebas de laboratorio

5.1 Estudio del ancho de banda

En este capítulo se expondrá de una forma práctica como se lleva a cabo el proceso de creación de funciones virtualizadas de red y de distintos flujos utilizando la herramienta mini-nfv y plantillas VNFFGD y VNFD escritas en yaml. Una vez definidos los distintos flujos se procederá a producir tráfico entre los distintos hosts que componen la topología para realizar distintas mediciones sobre dicho entorno de red.

La finalidad de este experimento es poder observar y simular como se lleva a cabo el proceso de creación de los VNF y VNFFG. Para más información respecto a la red y las pruebas realizadas revisar los Anexos I, II y III.

La topología que se va a emplear en primera instancia es la de un árbol básico compuesto por un switch y dos hosts, tal y como se observa en la figura 27.

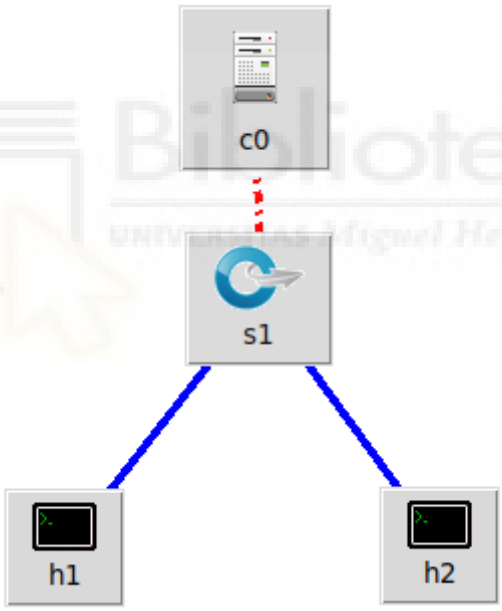

Figura 27: Topología 1, árbol básico. Fuente: Elaboración propia con miniedit.

Para iniciar la simulación en primer lugar se ha de ejecutar el script de mini-nfv, en este caso mininfv.py, con el siguiente comando:

```
$ sudo ./mininfv.py
```

Una vez iniciada la aplicación el primer paso es la creación del VNF que formará parte del primer flujo VNFFG. En este caso, la plantilla escogida fue la *tosca-vnfd-countpackets.yaml*, la cual puede observarse en la figura 28.

Como puede observarse en dicha figura, dentro de la VDU1 la memoria seleccionada son 512 MB, tamaño de disco 1 GB y 1 CPU. Por otro lado, se han definido el CP1 y el VL1. Al ser únicamente un CP el parámetro *order* estará en 0 y únicamente conectado a VDU1 y VL1.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Demo with user-data

metadata:
  template_name: sample-vnfd-userdata

topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            num_cpus: 1
            mem_size: 512 MB
            disk_size: 1 GB
      properties:
        image: cirros-0.3.5-x86_64-disk
        config: |
          param0: key1
          param1: key2
        mgmt_driver: noop
        user_data_format: RAW
        user_data: |
          #!/bin/sh
          FILE=$(ifconfig | head -n 1 | cut -f 1 -d ' ')
          mkdir exp
          watch -n 60 "date >> exp/$FILE; ifconfig | grep eth0 -C 5 | egrep 'RX|TX' > exp/$FILE" &

    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        order: 0
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1

    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net_mgmt
        vendor: ACME
```

Figura 28. Plantilla VNFD 1. Fuente: mini-nfv

Para instanciar dicha VNF el comando empleado es:

```
$ vnf_create --vnfd-file samples/vnfd/tosca-vnfd-countpackets.yaml vnfUD
```

Con dicho comando se creará el VNF con nombre vnfUD, el nombre escogido es de gran importancia debido a que será el nombré que figurará en la plantilla VNFFGD para la creación del flujo. La herramienta mini-nfv implementa los VNF como hosts virtuales dentro de una red aislada.

Las distintas plantillas utilizadas serán administradas y validadas por el NFVO siendo a su vez el responsable de gestionar la reserva de recursos del NFVI a través de los distintos VIM. El VNFM por su parte ha sido el encargado de realizar la petición de dicho VNF a través del VIM, en este caso el VIM será mininet, el cual realizará las peticiones al NFVI, que en este caso es el hardware del ordenador sobre el que se ha simulado el escenario. Mini-nfv realiza la función de NFVO y VNFM.

Una vez creado el VNF es preciso crear los dos hosts que forman parte de la topología y que serán los encargados de producir el tráfico que curse por la red. Para la creación de dichos hosts los comandos son los siguientes:

```
$ add_host h2 10.0.30.10/24
```

```
$ add_host h1 10.0.40.10/24
```

Con dicho comando se selecciona el nombre del host así como la red y dirección IP. En este experimento se generará tráfico, por lo que es necesario ejecutar un script de Python en uno de los hosts para otorgarle la función de servidor HTTP por el puerto 80. El comando es:

```
$ py h1.cmdPrint('python -m SimpleHTTPServer 80 &')
```

En el caso del primer flujo no se carga una plantilla de las disponibles en el catálogo de mininfv, sino que directamente se emplea el motor de plantillas *jinja 2* pudiendo cargar las variables de dicha plantilla por medio de comandos de forma dinámica. En este caso, lo que se define por comando son las direcciones IP de origen y destino con el comando:

```
$ px import yaml; net.values=yaml.load('---\nip_src: 10.0.30.10/24\nip_dst: 10.0.40.10/24')
```

Tras esto, se cargará la plantilla *vnffgd-helloworld-i.yaml* la cual se observa en la figura 29.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP12->CP12)
      properties:
        id: 51
        policy:
          type: ACL
          criteria:
            - destination_port_range: 80-65000
            - ip_dst_prefix: '{{ ip_dst }}'
            - ip_src_prefix: '{{ ip_src }}'
        path:
          - forwarder: vnfUD
            capability: CP111

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL11]
        connection_point: [CP111]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path1]
```

Figura 29. Plantilla vnffgd-helloworld-i.yaml. Fuente: mini-nfv

Como puede observarse, las políticas por las que se rige dicho flujo serán, por un lado, el rango de puertos del 80 al 65000 y por otro las IP de origen y destino, las cuales pueden ser modificadas dinámicamente vía comando. En este caso, las IP fueron definidas en el comando anterior. Seguidamente, se observa que el VNF por el que circulan los datos de este flujo es el vnfUD. En el caso de no hallarse coincidencias en las IPs o en el nombre del VNF indicado en el *forwarder* la instanciación del flujo falla.

Para instanciar e iniciar dicho flujo se ejecuta el siguiente comando:

```
$ vnffg_create_jinja --vnffgd-template exp2/tosca-vnffgd-helloworld-i.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample
```

En *vnffgd-template* se indica la plantilla que se emplea para el flujo, en *vnf-mapping* se establece cual es el VNF constituyente en dicho flujo, en *symmetrical* se indica si el tráfico es simétrico en lo referente al tráfico de subida y bajada y por último se indica el nombre del flujo.

Para poner la red y activar su conectividad es preciso encender el switch mediante el comando:

```
$ switch s1 start
```

Tras la puesta en marcha es preciso activar el analizador de red *Bmon*. Para ello se abre otra terminal y se ejecuta el siguiente comando:

```
$ sudo bmon
```

En este experimento el cliente h2 realizará peticiones cada segundo al servidor h1 mediante el siguiente comando:

```
$ watch -n 1 curl 10.0.40.10
```

Al realizar capturas de tráfico sobre la interfaz que une al switch con el VNF vnfUD se monitoriza el ancho de banda consumido en la red en dicha interfaz, obteniéndose los resultados que se observan en la figura 30:

Figura 30. Pruebas con un único flujo. Fuente: Elaboración propia

Tras la repetición de las pruebas y la realización de la media de los distintos datos obtenidos se obtiene:

- Rx = 1,36 (bytes/segundos)
- Tx = 1,37 (bytes/segundos)

Una vez obtenidos los resultados del primer flujo el experimento plantea el mismo escenario pero añadiendo otros dos hosts para conformar un nuevo flujo que atraviese el mismo VNF, en este caso el vnfUD, tal y como se observa en la figura 31.

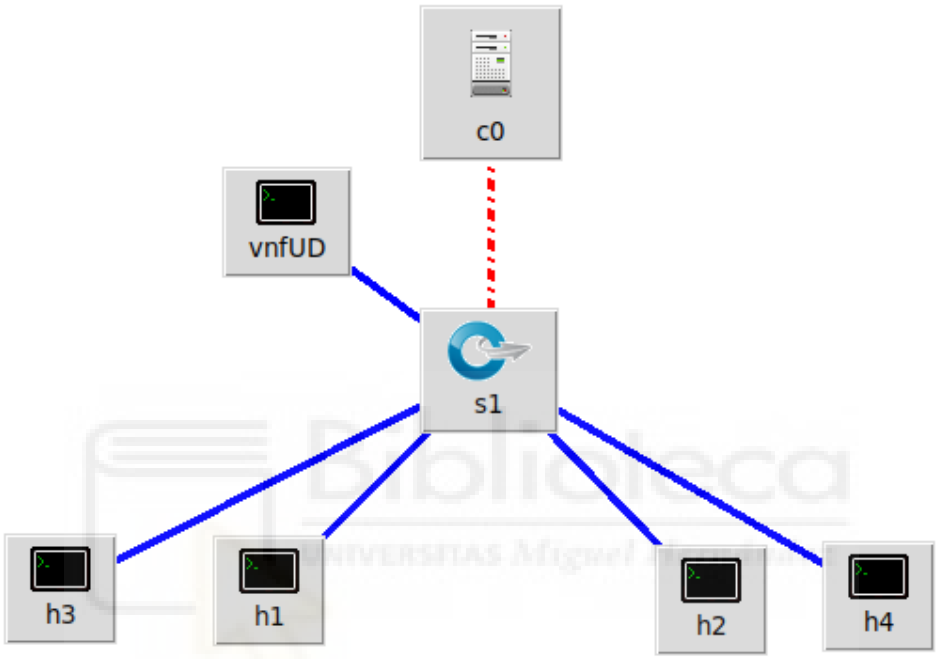

Figura 31: Topología 2, árbol básico con 4 hosts. Fuente: elaboración propia con miniedit.

Lo que se pretende es medir el incremento del consumo del ancho de banda en la interfaz del VNF. Para ello, los comandos empleados para añadir los nuevos nodos y flujo son los siguientes:

```
$ add_host h4 10.0.30.11/24
```

```
$ add_host h3 10.0.40.11/24
```

Se define al host h3 como un servidor http:

```
$ py h3.cmdPrint('python -m SimpleHTTPServer 80 &')
```

Para la definición del segundo flujo se opta por una plantilla *yaml* dentro del catálogo ofrecido por mini-nfv dentro del orquestado NFVO, la cual se puede observar en la figura 32. En este caso las IP vienen establecidas dentro de la plantilla y no se pueden modificar vía comando, por lo que serán un valor fijo. Por otro lado, en este caso nuevamente se define el rango de puertos activos.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP12->CP12)
      properties:
        id: 51
        policy:
          type: ACL
          criteria:
            - destination_port_range: 80-65000
            - ip_dst_prefix: 10.0.40.11/24
            - ip_src_prefix: 10.0.30.11/24
        path:
          - forwarder: vnfUD
            capability: CP111

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL11]
        connection_point: [CP111]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path1]
```

Figura 32. Plantilla flujo 2. Fuente: mini-nfv

Para la creación del flujo se emplea el siguiente comando:

```
$ vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-helloworld2.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample2
```

Nuevamente se ejecuta *Bmon* y se monitoriza el rendimiento de la red y el consumo del ancho de banda en la interfaz del VNF, tal y como se observa en la figura 33 tras realizar ambos clientes peticiones a sus respectivos servidores.

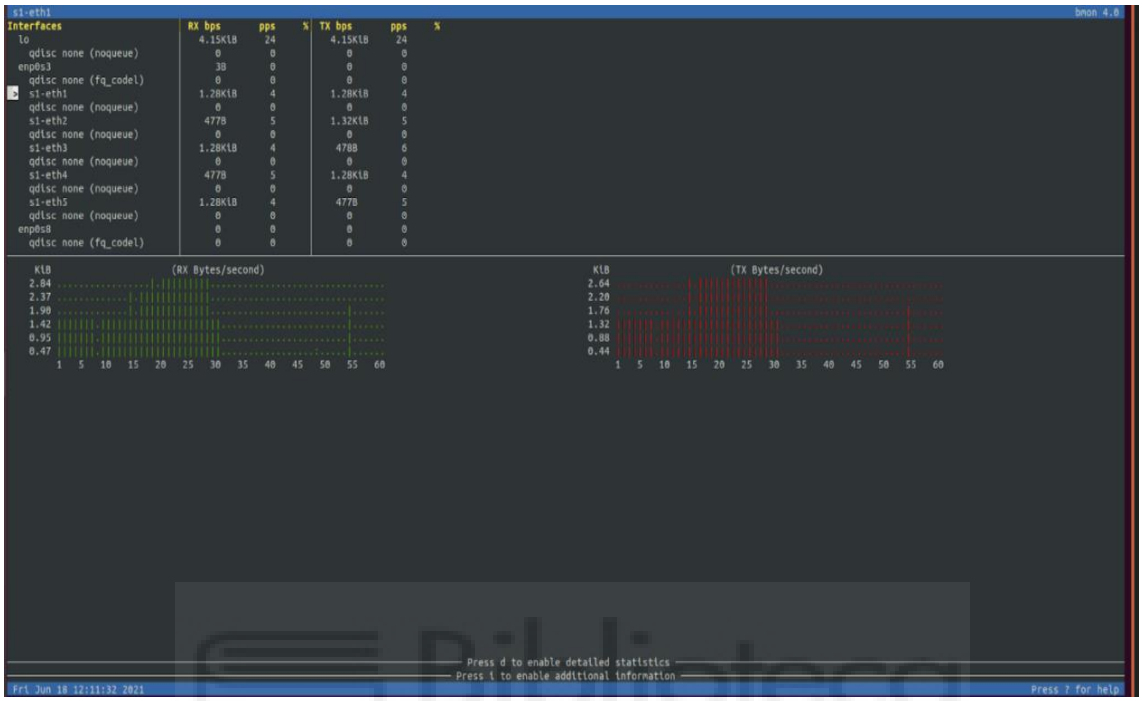

Figura 33. Pruebas con dos flujos. Fuente: Elaboración propia

Tras la realización del experimento un número elevado de veces se obtiene la siguiente media en los datos obtenidos:

- Rx = 2,97 (bytes/segundos)
- Tx = 2,91 (bytes/segundos)

El siguiente paso en el experimento es seguir añadiendo pares cliente-servidor http cuyo tráfico pase a través de la VNF establecida. La figura 34 ilustra como quedaría el entorno de red tras añadir el par.

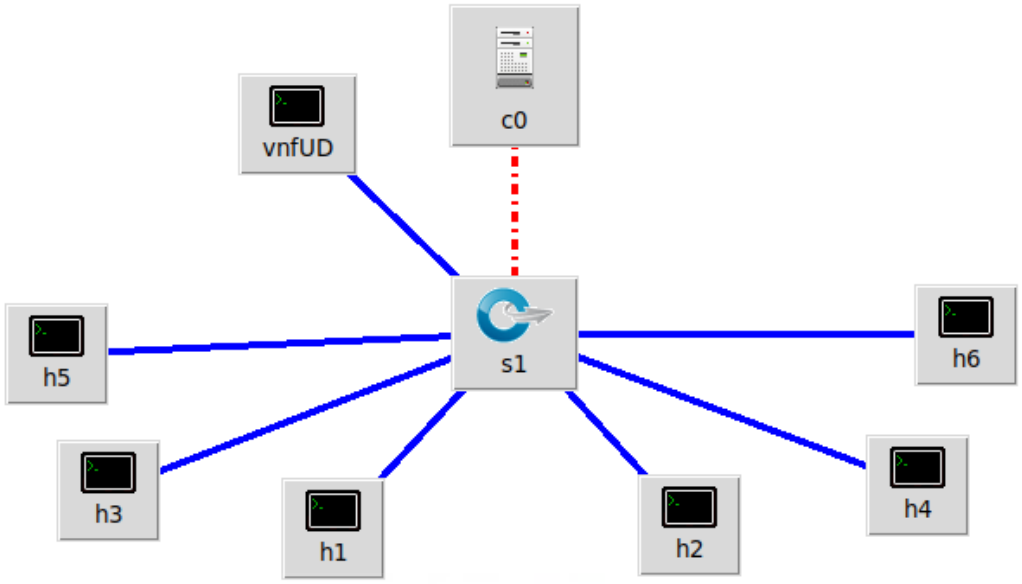

Figura 34: Topología 3, árbol básico con 6 hosts. Fuente: elaboración propia con miniedit

Para añadir dichos hosts se usa nuevamente el mismo comando usado en los dos casos anteriores, lo mismo ocurre con la creación del servidor http.

```
$ add_host h6 10.0.30.12/24
```

```
$ add_host h5 10.0.40.12/24
```

```
$ py h5.cmdPrint('python -m SimpleHTTPServer 80 &')
```

Tras la creación de los hosts que actuaran de cliente y servidor el siguiente paso es crear el flujo estableciendo las políticas sobre las que se regirá y se tratará el tráfico que lo atraviese. Para ello, se emplea nuevamente el comando visto anteriormente, usando la plantilla que se observa en la figura 35.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP13->CP13)
      properties:
        id: 52
        policy:
          type: ACL
          criteria:
            - destination_port_range: 80-65000
            - ip_dst_prefix: 10.0.40.12/24
            - ip_src_prefix: 10.0.30.12/24
        path:
          - forwarder: vnfUD
            capability: CP112

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL12]
        connection_point: [CP112]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path2]
```

Figura 35. Plantilla flujo 3. Fuente: mini-nfv

Para la creación del flujo se emplea el mismo comando utilizado en la creación de los dos flujos anteriores:

```
$ vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-helloworld3.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample3
```

Una vez creados los tres flujos se realizarán peticiones simultaneas cada segundo desde los distintos clientes a sus respectivos servidores empleando el comando:

```
$ watch -n 1 curl (IP Destino)
```

Nuevamente, mediante la herramienta *Bmon* se medirá el ancho de banda de la interfaz del VNF para comprobar cuál ha sido su incremento al añadir el nuevo flujo, pudiendo observar los resultados en la figura 36.

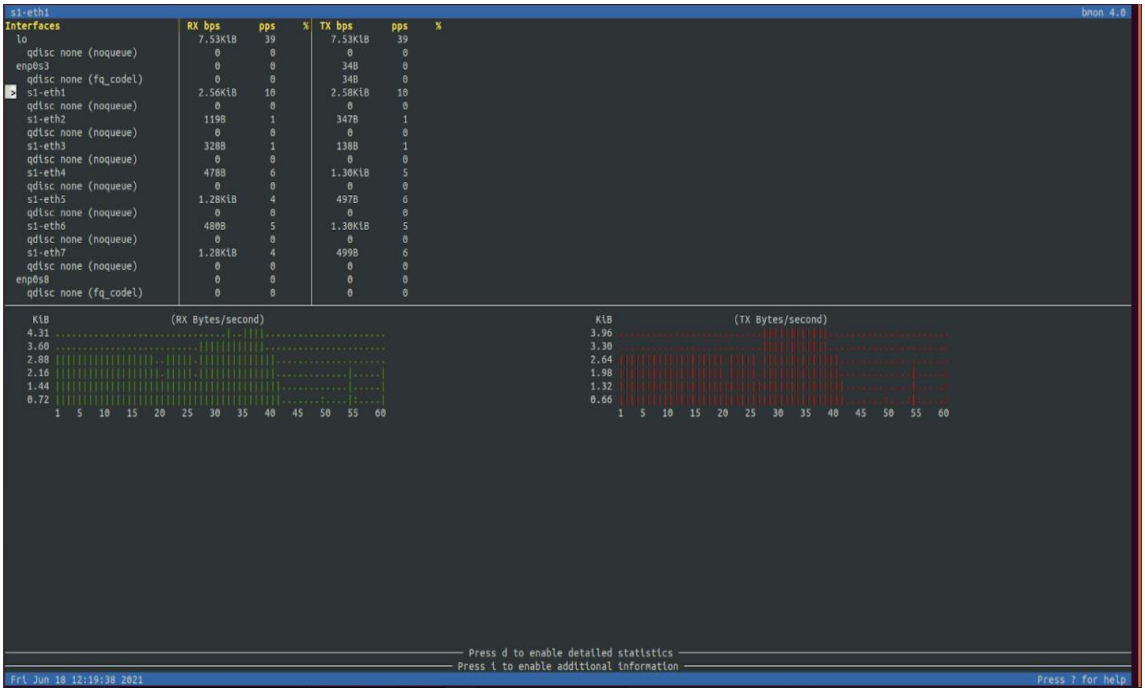

Figura 36: Pruebas con tres flujos. Fuente: Elaboración propia

Después de la repetición de dichas pruebas y tras realizar una media de los resultados obtenidos se ha determinado:

- Rx = 4,08 (bytes/segundos)
- Tx = 4,1 (bytes/segundos)

Por último, se realizan nuevamente todos los pasos anteriores para añadir un último flujo, quedando la topología según lo que se observa en la figura 37, añadiendo un nuevo par cliente-servidor cuyo tráfico atravesará el VNF.

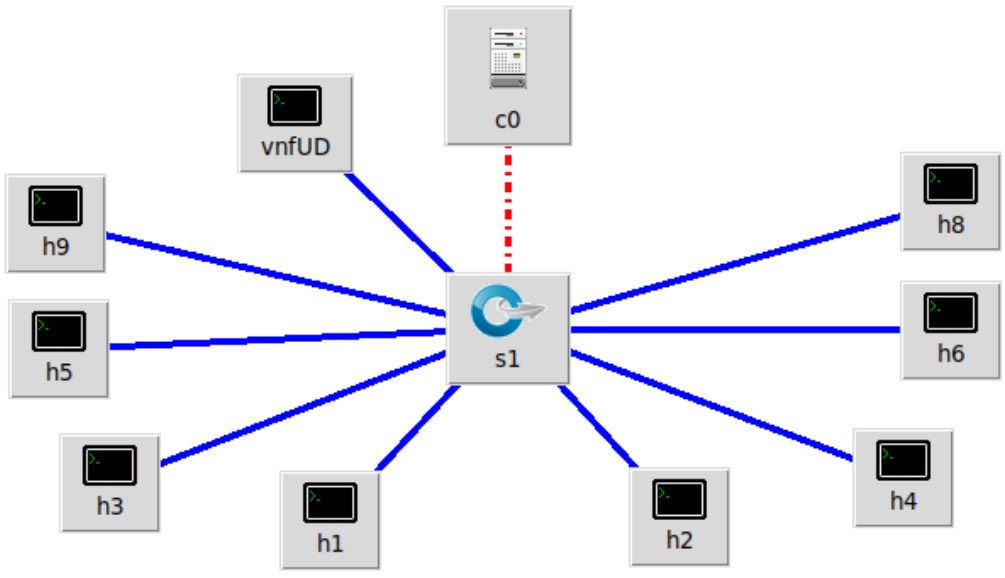

Figura 37: Topología 4, árbol básico con 8 hosts. Fuente: elaboración propia con miniedit

Nuevamente, el primer paso es añadir los dos hosts que se sumarán a la topología y crear el nuevo servidor http:

$ add_host h8 10.0.30.13/24

$ add_host h7 10.0.40.13/24

$ py h7.cmdPrint('python -m SimpleHTTPServer 80 &')

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP13->CP13)
      properties:
        id: 53
        policy:
          type: ACL
          criteria:
            - destination_port_range: 80-65000
            - ip_dst_prefix: 10.0.40.13/24
            - ip_src_prefix: 10.0.30.13/24
        path:
          - forwarder: vnfUD
            capability: CP113

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL13]
        connection_point: [CP113]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path3]
```

Figura 38. Plantilla flujo 4. Fuente: mini-nfv

Tras esto, nuevamente se instanciará la plantilla VNFFGD que se observa en la figura 38 con el siguiente comando:

$ vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-helloworld4.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample4

Tras la instanciación del nuevo flujo se procede a que los clientes realicen peticiones simultáneas cada segundo a sus respectivos clientes, cursando dicho tráfico por los distintos flujos que atraviesan el VNF creado, obteniendo los resultados observados en la figura 39.

Figura 39: Pruebas con cuatro flujos. Fuente: Elaboración propia

En base a la repetición de dicho experimento, finalmente se obtienen los datos:

- Rx = 5,30 (bytes/segundos)
- Tx = 5,30 (bytes/segundos)

En base a los resultados obtenidos en las diferentes pruebas anteriores se obtiene una gráfica ascendente tanto para los paquetes en el ámbito de Rx como en el de Tx, tal y como se observa en la figura 40.

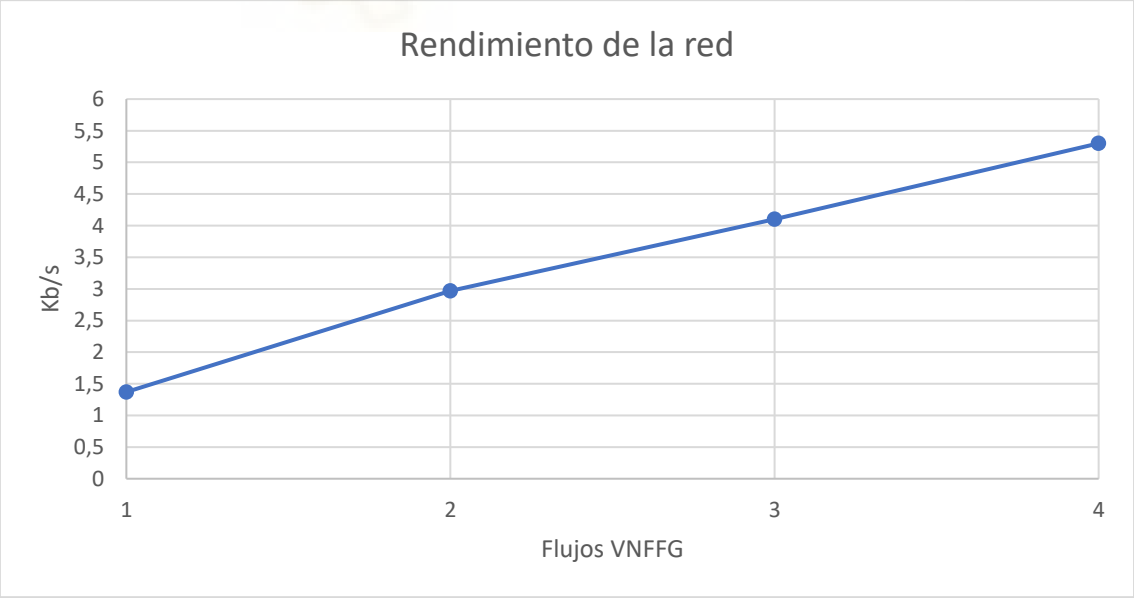

Figura 40: Gráfica de ancho de banda frente al número de flujos. Fuente: Elaboración propia

Mediante esta prueba se puede observar que los VNF consumen los recursos en función de la necesidad de los mismos, aumentando progresivamente este consumo si las circunstancias de tráfico en la red lo requieren. Uno de los retos planteados es la capacidad de que dichas funciones de red virtualizadas sean capaces de hacer un análisis predictivo y que aprendan en tiempo real sobre el comportamiento de la red para ofrecer cada vez resultados más precisos y eficientes en base a dichas predicciones.

5.2 Sobrecarga del VNF

Una vez estudiado el comportamiento de los VNF con un tráfico creciente el siguiente paso en el experimento es forzar la red al máximo para lograr que el VNF creado llegue al límite de los recursos que tiene asignados y observar su comportamiento.

En esta parte del experimento se emplearán la misma plantilla VNFD empleada en el experimento de medición del ancho de banda, así como los mismos flujos y hosts. Tras la creación y preparación del mismo entorno siguiendo los mismos comandos del experimento anterior se procede a abrir un terminal por cada uno de los clientes que realizarán consultas al servidor. Para ello se emplean los siguientes comandos en mini-nfv:

```
$ xtem h2
$ xterm h4
$ xterm h6
$ xterm h8
```


Una vez abiertas las consolas se emplea el mismo comando para emitir las peticiones visto en el experimento anterior:

```
$ watch -n 1 curl (IP Destino)
```

Esta vez, para forzar al límite los recursos asignados al VNF se dejará que dicho tráfico curse por tiempo ilimitado, esperando a que los recursos del VNF lleguen al límite establecido en la plantilla VNFGD.

Tras llegar al límite de sus recursos se procede a descartar los flujos progresivamente, cortando así parte del tráfico, quedando en evidencia la necesidad de la escalabilidad de los VNF ya que en este caso al haber sobrepasado los recursos que la VNF tenía asignados el tráfico de la red se ha visto afectado tal y como queda patente en la figura 41, en la que se observa como en primera instancia el ancho de banda ha alcanzado límites extremos durante un periodo de tiempo para finalmente disminuir casi por completo debido al descarte de los flujos en su totalidad.

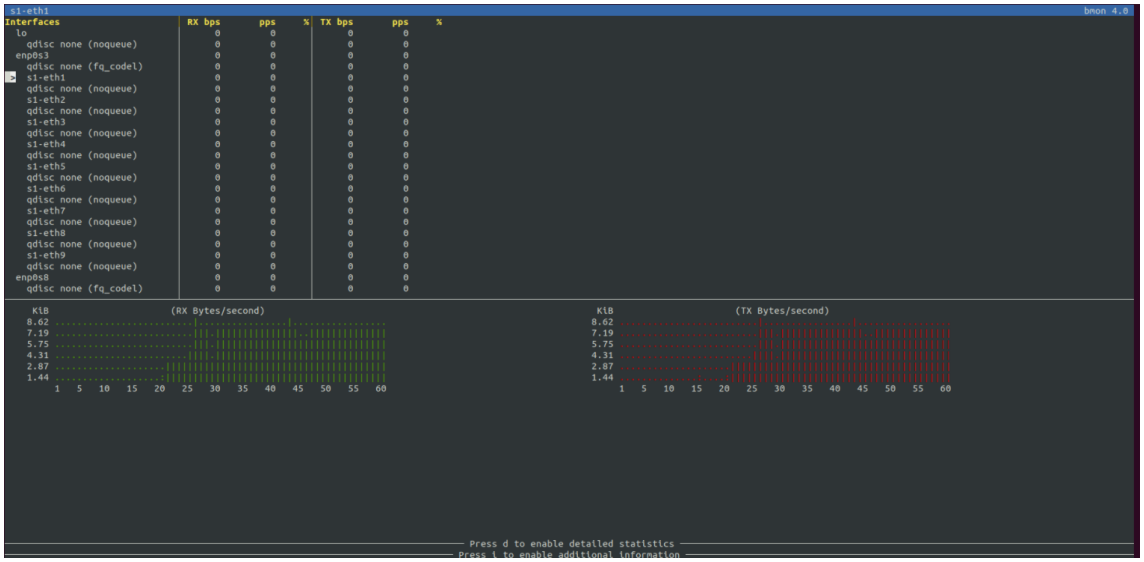

Figura 41: Sobrecarga de VNF. Fuente: Elaboración propia

En un entorno de red real esta pérdida de datos podría causar daños severos en dicha red de comunicación, ofreciendo una fiabilidad baja frente al crecimiento exponencial del flujo de datos. En el caso concreto de la red simulada en este experimento el VNF creado presenta uno de los comportamientos descritos en el apartado en el que se aborda el tema de la escalabilidad de los VNF. Este VNF al haber recibido un tráfico cada vez más creciente ha alcanzado su máximo para luego descartar todo el tráfico al haber sido superados los recursos asignados.

Aquí es donde se plantea otro de los retos en la industria de las comunicaciones dedicada a la tecnología NFV. Se espera que los VNF adquieran funcionalidades cada vez más inteligentes que les permitan gestionar los recursos de una forma más eficiente en función de las demandas de la red, aumentando así la fiabilidad de la misma y reduciendo el riesgo de que el tráfico se vea afectado.

5.3 Comparación de tráfico entre una red con tecnología NFV y sin tecnología NFV

En el último experimento se provocará la misma situación de tráfico para dos redes distintas, una empleando tecnología NFV habiendo instanciado un VNF y otra sin dicha tecnología.

Para la preparación de dicho entorno se planea crear 4 hosts, dos de los cuales actuarán de servidor y los dos restantes de clientes. La comunicación entre uno de los pares cliente-servidor se llevará a cabo a través de un flujo VNFFG que pasará el tráfico a través de la VNF creada. En este caso, se opta nuevamente por la misma plantilla VNFGD para crear la VNF.

Los comandos utilizados han sido:

```
$ vnf_create --vnfd-file samples/vnfd/tosca-vnfd-countpackets.yaml vnfUD
```

```
$ add_host h2 10.0.30.10/24
```

```
$ add_host h1 10.0.40.10/24
```

$ add_host h4 10.0.30.11/24

$ add_host h3 10.0.40.11/24

Se define al host h3 como un servidor http:

$ py h3.cmdPrint('python -m SimpleHTTPServer 80 &')

Se crea el flujo VNFGGD:

$ vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-helloworld2.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample2

Una vez preparado el entorno se pone en marcha el switch mediante el comando:

$ switch s1 start

Tras esto, los clientes realizarán las peticiones mediante el mismo comando empleado en los experimentos anteriores en un intervalo de tiempo de aproximadamente 10 segundos.

$ watch -n 1 curl (IP Destino)

En este caso se opta por la herramienta Wireshark dada su capacidad de capturar y graficar el tráfico obtenido de distintas interfaces de forma simultánea.

Para poner en funcionamiento Wireshark en el sistema mediante línea de comandos se emplea el siguiente comando en el bash de Linux y fuera de mini-fv:

$ sudo wireshark

Tras la ejecución del comando se abrirá la interfaz gráfica de la aplicación, tal y como se observa en la figura 42.

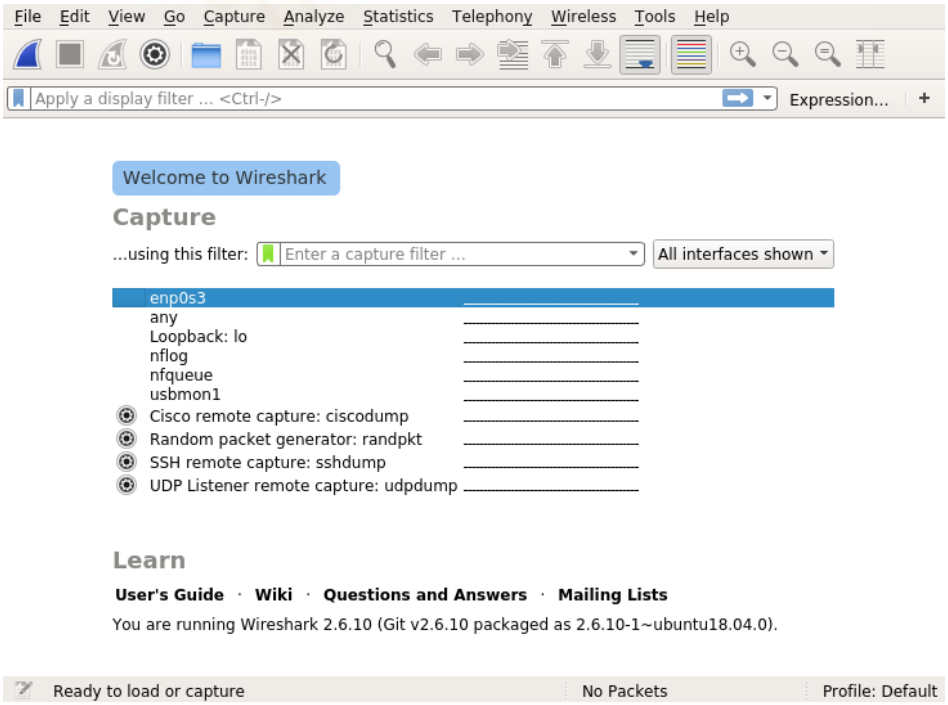

Figura 42: Interfaz de Wireshark. Fuente: Elaboración propia con Wireshark.

Tras esto se procede a capturar sobre las interfaces de los servidores para medir el consumo de ancho de banda comparando ambos escenarios. Una vez detenido el tráfico y empleando la herramienta *I/O Graph* de mininet se obtienen los resultados observados en la figura 42, en la cual la línea de tendencia roja representa el tráfico de la red sin tecnología NFV y el negro el que emplea tecnología NFV.

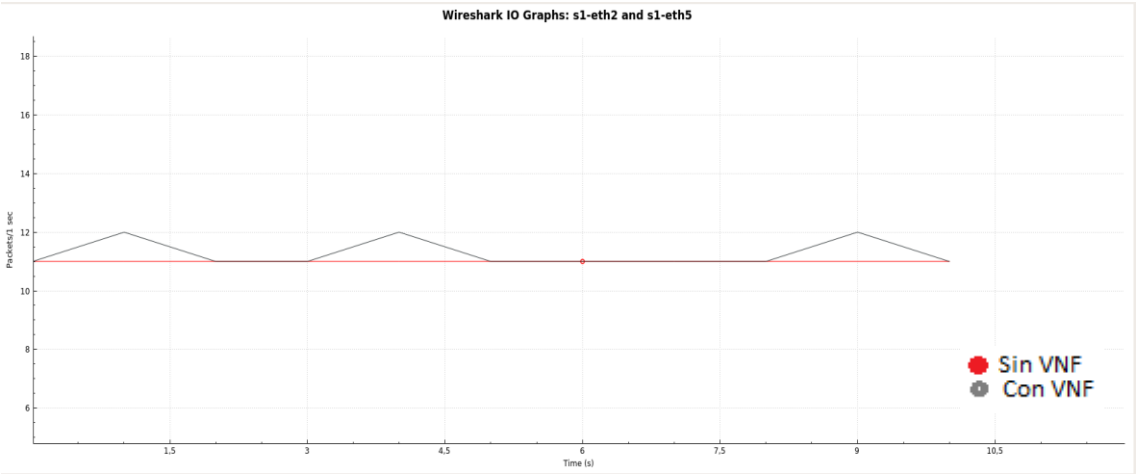

Figura 43: Comparación entre red sin NFV y red con NFV: Elaboración propia con Wireshark.

Como puede observarse, el consumo de ancho de banda es muy similar entre ambos escenarios, aunque en determinados momentos la red con NFV presenta picos en el consumo. Esto es debido a que la VNF según su funcionamiento en mininfv envía regularmente paquetes ICMP de redirección, produciendo estos picos en el consumo. Este consumo de ancho de banda depende en gran parte del switch que otorga la conectividad. Otro de los retos de la tecnología NFV es la mejora del rendimiento de Open vSwitch (OVS) para que satisfaga con una mayor eficiencia las peticiones del NFVI, lo que conllevaría finalmente a una mejor gestión de los recursos de la red.

6. Conclusiones

A lo largo de este trabajo se han expuesto de una forma didáctica y explicativa cual es el funcionamiento y las partes de las que está compuesta la tecnología NFV. Se trata de un concepto que presenta grandes expectativas de futuro, abarcando la mayor parte de los retos que presentan las redes actuales de comunicación.

Con la aparición y adopción de esta tecnología por las distintas redes de comunicación se abre un gran abanico de posibilidades que permite que los cimientos que sustentan dichas redes dependan menos del hardware, otorgando un mayor protagonismo al sector del software. Las redes NFV brindan a su vez un escenario de innovación que otorgará una mayor eficiencia operativa a los operadores que inicien la migración hacia dichos sistemas.

Algunas de las tecnologías que podrían verse beneficiadas de dicha migración son las redes LTE, pudiendo migrar sus núcleos EPC hacia sistemas que empleen tecnología NFV, revolucionando así la forma en la que los proveedores de servicios entregan dichos servicios a los clientes finales, logrando un mayor control operativo sobre los mismos.

Según un estudio publicado por it Reseller [17] el 79% de los profesionales que trabajan en el sector de las comunicaciones considera que NFV supone una importante evolución en el sector, que favorecerá mayormente a la flexibilidad de la red y los servicios.

Se prevé que la adopción de dicha tecnología crezca considerablemente en los próximos cinco años. No obstante, se presenta diversas dificultades que entorpecen la adopción de esta tecnología, retrasando dicho momento. Una de las causas de este entorpecimiento es el alto coste que presentan las soluciones NFV, así como la alta complejidad que conlleva el despliegue de dicha tecnología.

Para una aplicación correcta de un entorno NFV los operadores deberán planificar, desarrollar y optimizar previamente una solución que integre dicha tecnología para posteriormente poder aplicarla sin mayores complicaciones.

En conclusión, con el rápido avance hacia la digitalización y virtualización y con la aparición de conceptos como el de *Internet of Things* (IoT), la tecnología NFV cobrará cada vez mayor importancia y los operadores y organismos de estandarización colaborarán para hacer frente a los retos que esta tecnología plantea, como por ejemplo la optimización de los recursos empleados por las VNF tal y como se ha tratado de exponer en los experimentos realizados en este trabajo.

En definitiva, las posibilidades que plantea la tecnología NFV son infinitas, lo que sienta las bases para la consecución de una red de comunicación completamente eficiente y virtualizada.

Referencias

1. *Espeso, Pablo. Arquitectura x86, una historia imprescindible de la informática. 22 Enero 2013 en el sitio Xataka.*
2. *Lemus, Isaac. Historia de la virtualización. 10 de Marzo 2020 en el sitio Conocimiento libre.*
3. *Red Hat. ¿Qué es un hipervisor?*
4. *Sitio web Ciena. ¿Qué es NFV?*
5. *Millán Tejedor, Ramón Jesús; Efandiari, Shirin. Qué es… NFV (Network Functions Virtualization). COIT & AEIT, 2014.*
6. *Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV. ETSI GS NFV 003 V1.2.1. France 2014.*
7. *OpenStack. VNF Descriptor Template Guide. July 2020.*
8. *SdxCentral Studios. What is NFV Infrastructure (NFVI)? Definitive. April 27 2016.*
9. *SdxCentral Studios. What is NFV MANO?.October 9 2014.*
10. *Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors base don TOSCA specification. ETSI GS NFV-SOL 001 V2.5.1. FRANCE 2018.*
11. *Mininet Overview. Mininet official site.*
12. *About VirtualBox. Oracle.*
13. *Oracle VM VirtualBox. Oracle.*
14. *About Wireshark. Wireshark official site.*
15. *Stat Owl Statistics.*
16. *Castillo Lema, José. Mini-nfv. Github. 26 Mar, 2018.*
17. *it Reseller. Las telcos prevén invertir en virtualización de las funciones de red. 20 Feb 2019.*

Anexos

1. Anexo I: Guía de comandos.
2. Anexo II: Plantillas utilizadas.
3. Anexo III: Datos topología.

Anexo I: Guía de comandos

Experimentos 1 y 2:

$ sudo systemctl stop openvswitch-testcontroller.service

$ sudo ./mininfv.py

mininfv> vnf_create --vnfd-file samples/vnfd/tosca-vnfd-countpackets.yaml vnfUD

mininfv> add_host h2 10.0.30.10/24

mininfv> add_host h1 10.0.40.10/24

mininfv> py h1.cmdPrint('python -m SimpleHTTPServer 80 &')

mininfv> px import yaml; net.values=yaml.load('---\nip_src: 10.0.30.10/24\nip_dst: 10.0.40.10/24')

mininfv> vnffg_create_jinja --vnffgd-template exp2/tosca-vnffgd-helloworld-i.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample

mininfv> add_host h4 10.0.30.11/24

mininfv> add_host h3 10.0.40.11/24

mininfv> py h3.cmdPrint('python -m SimpleHTTPServer 80 &')

mininfv> vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-helloworld2.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample2

mininfv> add_host h6 10.0.30.12/24

mininfv> add_host h5 10.0.40.12/24

mininfv> py h5.cmdPrint('python -m SimpleHTTPServer 80 &')

mininfv> vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-helloworld3.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample3

mininfv> add_host h8 10.0.30.13/24

mininfv> add_host h7 10.0.40.13/24

mininfv> py h7.cmdPrint('python -m SimpleHTTPServer 80 &')

mininfv> vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-helloworld4.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample4

mininfv> switch s1 start

mininfv> xterm h2

mininfv> xterm h4

mininfv> xterm h6

mininfv> xterm h8

$ h%> watch -n 1 curl {IP destino}

Experimento 3:

$ sudo systemctl stop openvswitch-testcontroller.service

$ sudo ./mininfv.py

mininfv> vnf_create --vnfd-file samples/vnfd/tosca-vnfd-countpackets.yaml vnfUD

mininfv> add_host h2 10.0.30.10/24

mininfv> add_host h1 10.0.40.10/24

mininfv> add_host h4 10.0.30.11/24

mininfv> add_host h3 10.0.40.11/24

minifv> py h3.cmdPrint('python -m SimpleHTTPServer 80 &')

mininfv>vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-helloworld2.yaml --vnf-mapping vnfd-helloworld:'vnfUD' --symmetrical false vnffg-sample2

mininfv> switch s1 start

mininfv> xterm h2

mininfv> xterm h4

$h%> watch -n 1 curl {IP Destino}

$ sudo wireshark

Anexo II: Plantillas utilizadas

Plantilla VNFD empleada:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Demo with user-data

metadata:
  template_name: sample-vnfd-userdata

topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            num_cpus: 1
            mem_size: 512 MB
            disk_size: 1 GB
      properties:
        image: cirros-0.3.5-x86_64-disk
        config: |
          param0: key1
          param1: key2
        mgmt_driver: noop
        user_data_format: RAW
        user_data: |
          #!/bin/sh
          FILE=$(ifconfig | head -n 1 | cut -f 1 -d' ')
          mkdir exp
          watch -n 60 "date >> exp/$FILE; ifconfig | grep eth0 -C 5 | egrep 'RX|TX' > exp/$FILE" &

    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        order: 0
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1

    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net_mgmt
        vendor: ACME
```

Plantilla VNFFGD 1:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP12->CP12)
      properties:
        id: 51
        policy:
          type: ACL
          criteria:
            - destination_port_range: 80-65000
            - ip_dst_prefix: '{{ ip_dst }}'
            - ip_src_prefix: '{{ ip_src }}'
        path:
          - forwarder: vnfUD
            capability: CP111

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL11]
        connection_point: [CP111]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path1]
```

Plantilla VNFFGD 2:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP12->CP12)
      properties:
        id: 51
        policy:
          type: ACL
          criteria:
            - destination_port_range: 80-65000
            - ip_dst_prefix: 10.0.40.11/24
            - ip_src_prefix: 10.0.30.11/24
        path:
          - forwarder: vnfUD
            capability: CP111

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL11]
        connection_point: [CP111]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path1]
```

Plantilla VNFFGD 3:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP13->CP13)
      properties:
        id: 52
        policy:
          type: ACL
          criteria:
            - destination_port_range: 80-65000
            - ip_dst_prefix: 10.0.40.12/24
            - ip_src_prefix: 10.0.30.12/24
        path:
          - forwarder: vnfUD
            capability: CP112

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL12]
        connection_point: [CP112]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path2]
```

Plantilla VNFFGD 4:

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP13->CP13)
      properties:
        id: 53
        policy:
          type: ACL
          criteria:
            - destination_port_range: 80-65000
            - ip_dst_prefix: 10.0.40.13/24
            - ip_src_prefix: 10.0.30.13/24
        path:
          - forwarder: vnfUD
            capability: CP113

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 1
        dependent_virtual_link: [VL13]
        connection_point: [CP113]
        constituent_vnfs: [vnfd-helloworld]
      members: [Forwarding_path3]
```

Anexo III: Datos topología

Las topologías creadas en este experimento han seguido siempre el patrón de cliente-servidor. Es decir, por cada nodo creado que actúa como cliente se crea un nodo que actúa como servidor. De este modo al establecer cuatro flujos VNFFG se han creado cuatro pares cliente-servidor, dando lugar a 8 hosts.

H1: 10.0.40.10/24

H2: 10.0.30.10/24

H3: 10.0.40.11/24

H4: 10.0.30.11/24

H5: 10.0.40.12/24

H6: 10.0.30.12/24

H7: 10.0.40.13/24

H8: 10.0.30.13/24

La red integrará un único switch, el cual estará enlazada con cada uno de estos hosts. Las interfaces del switch con todos los flujos y el VNF instanciados son:

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

ether 08:00:27:37:3d:77 txqueuelen 1000 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 297 bytes 49213 (49.2 KB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.1.99 netmask 255.255.255.0 broadcast 192.168.1.255

inet6 fe80::94a7:329:8ba6:80a0 prefixlen 64 scopeid 0x20<link>

ether 08:00:27:1b:85:22 txqueuelen 1000 (Ethernet)

RX packets 226032 bytes 341305744 (341.3 MB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 46748 bytes 3238377 (3.2 MB)

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 9351  bytes 619103 (619.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9351  bytes 619103 (619.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e823:9ff:fe3c:64a8  prefixlen 64  scopeid 0x20<link>
        ether ea:23:09:3c:64:a8  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 3230 (3.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::4c9b:7bff:fe09:6d47  prefixlen 64  scopeid 0x20<link>
        ether 4e:9b:7b:09:6d:47  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 3230 (3.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::940e:2cff:feec:915d  prefixlen 64  scopeid 0x20<link>
        ether 96:0e:2c:ec:91:5d  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 3230 (3.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

s1-eth4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet6 fe80::38cd:ecff:fee9:c6c prefixlen 64 scopeid 0x20<link>

ether 3a:cd:ec:e9:0c:6c txqueuelen 1000 (Ethernet)

RX packets 11 bytes 866 (866.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 27 bytes 3230 (3.2 KB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet6 fe80::1801:73ff:febf:b4 prefixlen 64 scopeid 0x20<link>

ether 1a:01:73:bf:00:b4 txqueuelen 1000 (Ethernet)

RX packets 11 bytes 866 (866.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 27 bytes 3230 (3.2 KB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth6: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet6 fe80::e858:e6ff:fe8d:4db4 prefixlen 64 scopeid 0x20<link>

ether ea:58:e6:8d:4d:b4 txqueuelen 1000 (Ethernet)

RX packets 11 bytes 866 (866.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 27 bytes 3230 (3.2 KB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth7: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet6 fe80::2068:64ff:fe81:1aea prefixlen 64 scopeid 0x20<link>

ether 22:68:64:81:1a:ea txqueuelen 1000 (Ethernet)

RX packets 11 bytes 866 (866.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 27 bytes 3230 (3.2 KB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet6 fe80::2482:3dff:fe42:29cf prefixlen 64 scopeid 0x20<link>

ether 26:82:3d:42:29:cf txqueuelen 1000 (Ethernet)

RX packets 11 bytes 866 (866.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 27 bytes 3230 (3.2 KB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth9: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet6 fe80::b898:d2ff:fe94:1aa7 prefixlen 64 scopeid 0x20<link>

ether ba:98:d2:94:1a:a7 txqueuelen 1000 (Ethernet)

RX packets 11 bytes 866 (866.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 27 bytes 3230 (3.2 KB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

Para medir el ancho de banda disponible en dichos enlaces se emplea el comando Iperf de Linux:

Client connecting to 10.0.30.11, TCP port 5001

TCP window size: 85.3 KByte (default)

---

[13] local 10.0.30.12 port 60966 connected with 10.0.30.11 port 5001

[ID] Interval Transfer Bandwidth

[13] 0.0-10.0 sec 20.7 GBytes 17.8 Gbits/sec

Por otro lado se emplea nuevamente el comando Iperf para el tráfico UDP:

Client connecting to 10.0.30.11, UDP port 5001

Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)

UDP buffer size: 208 KByte (default)

---

[13] local 10.0.30.12 port 48184 connected with 10.0.30.11 port 5001

[ID] Interval   Transfer   Bandwidth

[13] 0.0-10.0 sec 1.25 MBytes 1.05 Mbits/sec

[13] Sent 893 datagrams

[13] Server Report:

[13] 0.0-10.0 sec 1.25 MBytes 1.05 Mbits/sec   0.000 ms   0/ 893 (0%)