Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Herramienta para el despliegue automático de Redes Virtuales

Autor: David Hernández Puerta

Tutor: Joaquín Álvarez Horcajo

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Herramienta para el despliegue automático de redes virtuales

Autor: David Hernández Puerta

Tutor: Joaquín Álvarez Horcajo

Tribunal:

Presidente: Juan Antonio Carral Pelayo

Vocal 1º: Elisa Rojas Sánchez

Vocal 2º: Joaquín Álvarez Horcajo

1 5 de Julio de 2022

Dedico mi Trabajo de Fin de Grado a todas aquellas personas que no confían en sí mismas.

Agradecimientos

Quisiera agradecer:

A Joaquín Álvarez Horacajo, mi tutor, por haber confiado en mi desde el principio para la realización de este Trabajo Fin de Grado, y además por su continua ayuda, sin la cual no hubiera sido viable el desarrollo de este proyecto.

A todos los compañeros del laboratorio LE34 por haber aportado su miguita de pan en este TFG.

A toda mi familia, en especial a mis padres y a mi hermana, por ser los principales pilares y haberme guíado a lo largo de mi vida para convertirme en la persona que soy hoy en día.

A mis amigos, que gracias a todos vosotros habéis conseguido que estos cuatro años de Universidad hayan sido increíbles, pasando por momentos de máxima felicidad y por otros de absoluta frustración.

Y en especial, a mi novia Nerea, la persona que más feliz me hace en este mundo porque me apoya, me motiva, me inspira y me saca mis mejores sonrisas.

Gracias por apoyarme y formar parte de esto, de cierta manera un pedacito de todos vosotros está reflejado en este proyecto.

Resumen

Este Trabajo Final de Grado (TFG) se enfoca en el desarrollo de una herramienta de automatización para el despliegue de redes virtuales heterogéneas, emulando dispositivos de arquitecturas diferentes, con el objetivo de interconectar los nodos según la información topológica generada mediante una herramienta denominada Boston University Representative Internet Topology Generator (BRITE).

Se diseña un modelo de red virtual y se comprueba su funcionamiento con el propósito de conseguir la implementación automática de numerosas topologías de red. Así como, una aplicación web para seleccionar las características de los sistemas finales y obtener información acerca de la red desplegada.

Palabras clave: redes virtuales heterogéneas, automatización, nodos, BRITE, aplicación web

Abstract

This Bachelor's Degree Final Project is focused on the development of an automation tool for the deployment of heterogeneous virtual networks, emulating devices of different architectures, with the aim of interconnecting nodes according to a topological information which is generated through a tool called BRITE.

A virtual network model is designed and its performance is tested in order to achieve an automatic deployment of multiples network topologies, as well as, a web-based application is developed to select the features of the end devices and to get information about the deployed network.

Keywords: heterogeneous virtual networks, automation, nodes, BRITE, web-based application

"Lo que importa verdaderamente en la vida
no son los objetivos que nos marcamos,
sino los caminos que seguimos para lograrlo"

Peter Bamm

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

A mediados del siglo XVIII, la sociedad comenzó un proceso de transformación económica, social y educativa que favoreció al origen de la *Primera Revolución Industrial*. Uno de los acontecimientos más importantes fue la invención de la máquina de vapor. A partir de este momento los avances tecnológicos y científicos comenzaron a tomar un importante papel. Es gracias a ello que acontecen la *Segunda* y *Tercera Revolución Industrial* con el uso de la electridad y la transformación de la información y de las telecomunicaciones para automatizar los procesos de producción.

En los últimos años, la conectividad de los dispositivos ha crecido exponencialmente, aumentando, por consiguiente, el tráfico en la red. Para desplegar una red, se necesita disponer de diversos componentes: dispositivos finales, routers, switches y un medio de transmisión, que pueden ser guiados o no guíados. Los guíados se transmite y dirigen a través medios sólidos, por ejemplo, el par trenzado, la fibra óptica y el coaxial. En los no guiados, las ondas electromagnéticas se propagan por aire o vacío. Además, para el correcto funcionamiento de una red, son necesarios otros elementos como son los protocolos.

En la actualidad existen varios modelos que dividen la red en capas. El primero, estandarizado por ISO, es el modelo OSI que define una estructura de 7 capas: Aplicación, Presentación, Sesión, Transporte, Red, Enlace y Física. El segundo, el modelo TCP/IP, que divide a la red en 4 capas: Aplicación, Transporte, Interred y Nodo a red. En este modelo, no es obligatorio el uso de todas las capas, ya que, la capa Nodo a red no está especificada en TCP/IP. El tercero, el modelo de Internet, que consta de 5 capas: Aplicación, Transporte, Red, Enlace y Física [1]. Cada capa necesita protocolos para su funcionamiento. Basándose en el modelo TCP/IP, un protocolo de la capa de Aplicación puede ser DNS, de la capa de transporte: TCP, de la capa de Interred: IPv4 y de la capa de Nodo a red: Ethernet.

Para montar una red en un entorno real es nesario disponer de elementos físicos como ordenadores, routers, switches y otros dispostivos, lo que dificulta su despliegue o una vez montada, modificarla o reestructurarla conlleva un proceso complejo, además de costes, a veces, inasumibles.

El desarrollo de tecnologías como la virtualización hacen posible, que los elementos de la red se puedan desplegar en un solo sistema de manera más rápida y sencilla. Gracias a la virtualización, la sociedad está caminando hacia una nueva transformación: La *Cuarta Revolución Industrial*, que genera un mundo en el que los sistemas físicos y virtuales cooperan entre sí de una manera más flexible [2].

La virtualización es una tecnología que permite dividir y compartir el hardware de un dispositivo para simular o emular otros dispositivos o incluso funcionalidades, por ejemplo, la división del disco de almacenamienro físico en discos de almacenamientos lógicos, otro ejemplo puede ser la ejecución de máquinas virtuales. Al ser un concepto abstracto y muy

amplio, la virtualización se divide en varios tipos: virtualización de escritorio, virtualización de aplicaciones, virtualización de recursos y virtualización de plataforma [3].

Existen diferentes grados de aproximación a la realidad: la simulación y la emulación. La simulación consiste en la representación de un sistema físico mediante un módulo abstracto [4], es decir, que modela toda la realidad. También se puede entender como un simulacro de cómo opera el sistema, ya que el funcionamiento coincide con la realidad, sin embargo, el modelo intenta aproximarse a la realidad, mientras que, la emulación es un modelo más real en el que la representación del sistema es más completa en su conjunto, porque imita parte de la realidad y otra parte la modela [5].

La creciente interconexión entre los dispositivos, y el auge de la virtualización y de los sistemas simulados y emulados, promovió el desarrollo de una nueva herramienta para desplegar redes virtuales heterogéneas de manera automática. La información sobre la topología a desplegar: nodos y enlaces, se obtendrá de un archivo generado por BRITE.

BRITE es una herramienta que permite crear topologías aleatorias bajo unos criterios. El archivo de salida contiene una serie de parámetros que podrán ser utilizados para el posterior despliegue de la red, como el ID del nodo y los nodos que se tienen que inteconectar, es decir, los enlaces. A partir de dicha información, se crean los elementos de red virtuales necesarios, por ejemplo, dispositivos *bridge*, interfaces *tap* y *veth*, máquinas virtuales, etc.

En el mundo real, no solamente existen dispostivos de una sola arquitectura *hardware*, sino que, cooperan dispositivos de muchas arquitecturas. Por este motivo, las máquinas virtuales que se van a desplegar necesitarán arquitecturas diferentes, por ejemplo *Intel* y *ARM*. Esto va a proporcionar dos cararterísticas a la herramienta: la heterogeneidad y la aproximación a la realidad.

Existen herramientas que pueden ejecutar máquinas virtuales, por ejemplo VirtualBox y VMware son un software para virtualizar y, QEMU, en cambio, es un software que permite tanto emular como virtualizar. Se selecciona QEMU para el desarollo de la herramineta, ya que es muy interesante que pueda emular máquinas con diferentes arquitecturas.

Por último, con el fin de facilitar la manejabilidad de la herramieta se incorporá una interfaz gráfica. Por ejemplo, en la interfaz gráfica de *Packet Tracer* se puede escoger y configurar los dipositivos, desplazarlos y situarlos en un punto del lienzo, unirlos con otros dispositivos, etc. Sin embargo, la interfaz gráfica que se va a desarrollar será una interfaz web basada en un modelo cliente-servidor. En la página web se podrán introducir archivos de configuración con el objetivo de desplegar la red con la información contenida en estos.

1.1 Campos de aplicación

En la actualidad, con el aumento del uso de las tecnologías desarrolladas en la *Cuarta Revolución Industrial*, está comenzando un nuevo concepto denominado “Smart City” en la que muchos países están invirtiendo mucho capital para su crecimiento, España es uno de ellos.

Una “Smart City” se define como un conjunto de tecnologías y personas con el propósito

de crear mejores infraestructuras para facilitar la comodidad de los ciudadanos en ámbitos de servicios públicos, eficiencia energética, medioambiente, seguridad, ocio, turismo, interconectividad, etc.

Instituciones como la Universidad de Alcalá están investigando con el objetivo de desarrollar nuevos protocolos para su implementación en "Smart Cities". Esta herramienta puede servir para la realización de pruebas enfocadas a "Smart Cities", ya que permite desplegar redes virtuales heterogéneas emulando dispositivos de arquitecturas diferentes. Por ejemplo, los nodos podrían ser diferentes sensores ubicados en distintos puntos de una ciudad para recoger ciertos parámetros y enviárselos entre ellos o al servidor central, donde se procese la información.

1.2 Objetivos

El principal objetivo de este proyecto, es el despliegue de redes virtuales de manera automatizada dentro de un equipo, en la que los nodos que conforman dicha red, implementen arquitecturas diferentes. Dentro de este se desarrollan objetivos más pequeños:

- **Ejecución de máquinas virtuales**. Se quiere ejecutar máquinas virtuales de diferentes arquitecturas para que la red virtual sea heterogénea: *Intel* y *ARM*. Por la limitación de los recursos, se determinará el número máximo de máquinas virtuales a desplegar.
- **Interconexión de nodos**. Se ha de conectar interfaces virtuales con las máquinas virtuales para lograr la comunicación entre los nodos según indica la configuración topológica generada por BRITE.
- **Automatización**. Cada vez que se quiera desplegar una red, el usario no ha de configurar manualmente los parámetros de la red, indicados en los archivos generados por BRITE, sino que, se desarrollará un software para recoger la información contenida en los archivos y aunar todos los procesos de la red para desplegarla de manera automatizada.

El segundo objetivo de este proyecto es el desarollo de una aplicación web, con el fin de facilitar al usuario el despliegue de la red. Este, también se divide en subobjetivos:

- **Página web para enviar archivos**. Una vez adquiridas las bases teóricas, se creará una página web sencilla en la que existan campos para introducir los archivos generados por la herramienta BRITE con el fin de enviarlos al servidor.
- **Servidor**. Desarrollar un servidor para la recepción, almacenamiento, y procesamiento de los archivos enviados desde la página web y así poder ejecutar la automatización de despliegue.

1.3 Estructura y contenidos de la memoria

La estructura de este TFG se desarrolla en base a los siguientes capítulos:

Capítulo 1: Introducción. En este primer capítulo, se realiza un breve resumen del proyecto a modo de introducción, así como, la enumeración de los diferentes obejetivos y la esquematización de la estrucutra del proyecto.

Capítulo 2: Estado del arte. Este capítulo está centrado en las explicaciones teóricas de los temas tratados de este proyecto. Se exponen cinco temas: Virtualización, interfaces Linux para virtualización de redes, protocolos, la Web y herramientas.

Capítulo 3: Desarrollo e implementación.. En este capítulo se presenta el diseño de la herramienta y cómo se ha ido construyendo paso a paso hasta conseguir la aplicación final. Primero se explica el desarrollo de la red virtual y en segundo lugar la construcción de la interfaz web

Capítulo 4: Pruebas y Análisis de resultados. Se expone la realización de las pruebas, así como, su estudio para determinar el funcionamiento correcto de las acciones llevadas a cabo a lo largo del desarrollo.

Capítulo 5: Conclusiones y trabajo futuro. Es el último capítulo, para finalizar se exponen las conclusiones, así como, los trabajos futuros del proyecto.

2 Estado del arte

En este capítulo se expondrá el marco teórico, comenzando por los conceptos de virtualización, continuando por algunos protocolos de red tales como DHCP y STP, para finalizar con la explicación de las herramientas más importantes utilizadas en el desarrollo del proyecto.

2.1 Virtualización, emulación y simulación

Es esta sección se explica qué es la virtualización, así como los tipos de virtualización, de ellos muy importante la simulación y la emulación. Además, para introducir la vitualización se narra sus inicios.

2.1.1 Historia de la virtualización

La virtualización comienza a desarrollarse en la década de *1960* [6]. IBM inició un proyecto, al que denominaron *Atlas*, con el objetivo de solucionar los graves problemas surgidos del uso común de un único ordenador para muchos trabajadores . Este proyecto consistió en la evolución de la segmentación de las supercomputadoras, también llamadas *"mainframes"*, con el fin de mejorar el rendimiento de la CPU [7]. Así nació el ordenador *IBM 7044* , ver figura 2.1, capaz de repartir los recursos y su uso al mismo tiempo, principalmente disco memoria y capacidad de cómputo. Además ofreció seguridad y fiabilidad para que el trabajo realizado por un empleado no interfiriese en el trabajo de otros.

A partir de este acontecimiento, se comenzó a desarrollar otros ordenadores como el *IBM System/360 Model 67* , que virtualizaba todas las interfaces hardware a través de un monitor de máquinas virtuales, llamado posteriormente hipervisor. Fue denominado con este nombre debido a la capacidad que tenía de ejecutar sistemas operativos dentro de otros [3]. En la década de 1980, la virtualización fue perdiendo fuerza debido al comienzo del modelo de informática distribuida con las aplicaciones basadas en una arquitectura *cliente-servidor* y el auge de los servidores *x86*, por tanto, la virtualización sólo se utilizaba en las supercomputadoras de grandes centros de datos como en bancos, laboratorios de investigación militares o de universidades. Aún así, bajo la sombra, la virtualización fue evolucionando hasta que en 1988 surge el *IBM AS/400* [3], un ordenador para pequeñas y medianas empresas capaz de ejecutar sistemas multiusario.

En 1999, *VMware*, una empresa fundada por cinco investigadores de la *Universidad de California*, presentó su primer producto, *Workstation 1.0*. Este permitía a un usuario ejecutar múltiples sistemas operativos como máquinas virtuales en ordenadores con arquitectura *x86* [8]. A la entrada del nuevo milenio, la virtualiación empezó a ser más conocida, varias

empresas como *VMware*, *Red Hat*, *IBM*, *Citriz Systems* y *Microsoft* comenzaron a presentar más productos de virtualización, de tal manera que los usuarios podían elegir entre diferentes softwares para virtualizar. Hasta que en 2013, *Docker INC.* lanzó un nuevo tipo de virtualización basada en contenedores, ofreciendo la compartición de recursos del propio sistema operativo aislados del resto del sistema [3]. Actualmente, los contenedores se han convertido en una herramienta muy importante para el despliegue de aplicaciones.

2.1.2 Defínción. Tipos de virtualización

Generalmente, cuando pensamos en el término de virtualización, imaginamos una máquina virtual ejecutando un sistema operativo diferente al de nuestro ordenador. Esta concepción de virtualización no es errónea pero tampoco correcta, puesto que, esa idea que nos viene rápidamente a la cabeza es un tipo de virtualización y no el término general. Además, conceptos ligados a la virtualización como *simulación* y *emulación* se suelen confundir.

La *virtualización* es un concepto abstracto y muy amplio, por ello es mejor empezar definiendo el término *"virtual"*. Decir que cierta cosa, objeto, elemento, foco, etc es virtual significa que no es real, es decir, que intenta producir un efecto aparente a la realidad. Así que, la *virtualización* es una tecnología que permite dividir y compartir el hardware de un dispositivo para generar otros dispositivos o funcioanalidades.

El concepto de virtualización se puede desglosar en cuatro tipos para conseguir definiciones más concretas:

- **Virtualización de escritorio**

  La virtualización de escritorio es un método por el cual se puede manipular la intefaz gráfica de un equipo de forma remota [3]. Esto supone que un usuario se puede conectar desde cualquier dispositivo, con conexión a Internet, a su escritorio de forma remota.En este caso, el recurso que se abstrae es el escritorio remoto. Físicamente el usuario no está delante de él, sino que accede a él, por tanto, no es consciente del lugar físico en el que se encuentra el equipo remoto.

Figura 2.1: IBM 7044

- **Virtualización de aplicaciones**

  La virtualización de aplicaciones es un proceso que hace creer que las aplicaciones que están siendo ejecutadas interactúen directamente con el sistema operativo, aunque en realidad no lo hacen [9], sino que, el que actua directamente con el sistema opertativo es un sofware de virtualización [3]. Esto quiero decir que se aísla la componente lógica de la aplicación del componente sistema operativo [10].

  Dentro de la virtualización de aplicaciones se pueden diferenciar dos tipos:

  - **Virtualización de aplicaciones limitada**. En este tipo se incluyen las aplicaciones que se pueden ejecutar desde dispositivos de almacenamiento externos, es decir, aplicaciones portables. "Lo normal es que en este caso, en virtualización de aplicaciones limitada, no medie ninguna capa de virtualización o software con las mismas prestaciones y que la portabilidad se encuentre limitada al sistema operativo sobre el que correrá la aplicación" [3]. En este caso, el recurso que se abstrae es el sistema operativo donde se ejecuta la aplicación portable.
  - **Virtualización de aplicaciones completa**. Corresponde a un software intermedio con el fin de comunicar el sistema operativo con el hardware [3]. La *simulación* es un tipo de virtualización de aplicación completa, al igual que la *portabilidad multiplataforma*.
    - **Portabilidad multiplataforma**. Este recurso permite ejecutar aplicaciones compiladas para una determinada CPU y un determinado sistema operativo en diferentes CPUs y sistemas operativos usando una traducción binaria dinámica [3]. Algunos ejemplos son *Portable .NET*, *Java Virtual Machine*, *Citrix XenApp*.
    - **Simulación**. Consiste en la representación de un sistema físico mediante un módulo abstracto [11] , es decir que modela toda la realidad. También se puede entender como un simulacro de cómo opera el sistema, ya que el funcionamiento coincide con la realidad, pero el modelo no es real.

      Supongamos un simulador de red, como por ejemplo, *ns3*, ver figura 2.2. Se crean varios elementos de red, un router, un switch, varios ordenadores, enlaces a través de un entorno gráfico. Sobre este modelo se pueden generar flujos de tráfico definidos a medida del simulador, por tanto este tráfico no es real. Esta característica es lo que define ser una simulación.

- **Virtualización de recursos**

  La virtualización de recursos engloba la representación abstracta de un elemento individual del ordenador, como la abstracción de la memoria, la red, el disco, dispostivos de entrada y salida, etc [3]. Por ello la *virtualización de recursos* se divide en varios tipos según los elementos que se quieran virtualizar:

  - **Encapsulación**. Consiste en ocultar la complejidad de un recurso creando una representación simplificada [3].
  - **Memoria virtual**. Es un mecanismo que permite la ejecución de programas par-

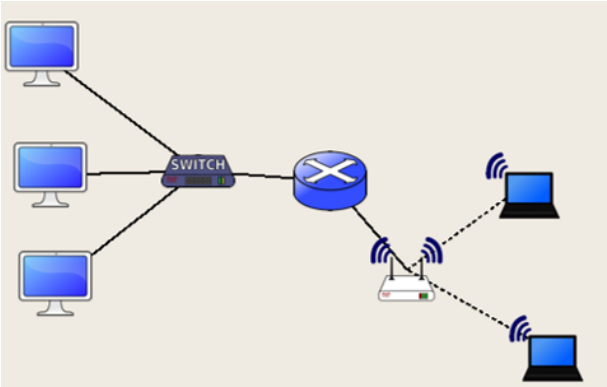


Figura 2.2: Escenario de red en ns-3

cialmente cargados en la memoria principal, ofreciendo al usuario un espacio de memoria ilimitado, es decir, que el recurso no tiene que estar totalmente cargado en la memoria principal, gracias al desacoplamiento entre el espacio de direcciones físicas y el espacio de direcciones virtual [3].

- **Virtualización de almacenamiento**. Este término se refiere al proceso de abstracción del almacenamiento físico en almacenamiento lógico [3].
- **Virtualización de red**. Permite proveer las funciones de la red y los recursos, tanto del hardware como del software, en un modelo basado solamente en software [12], al mismo tiempo que se siguen aprovechando las redes físicas subyacentes [3]. Los elementos de red operan de la misma forma que en una red física, puesto que ahora son abstracciones lógicas de módulos de software ejecutados dentro de un ordenador [3].
- **Virtualización de entrada y salida**. La capa de abstracción entiende el manejo de las solicitudes de entrada y salida entre los dispostivios virtuales y el hardware físico que es compartido. En este caso, los recursos abstraídos son las conexiones de entrada, salida y transporte [3].

- **Virtualización de plataforma**

  La*virtualización de plataforma* consiste en la abstracción de todo el hardware subyacentede de un sistema, de manera que, diversas instancias de sistemas operativos puedan ejecutarse de manera independiente [3]. ”Independiente” se refiere a que las máquinas virtuales comparten ciertos recursos creyendo que son solo suyos. Por eso se dice que las máquinas virtuales no se ven entre ellas, a pesar de compartirlos.

  Los tipos de *virtualización de plataforma* son los siguientes:

  - **Virtualización de sistema operativo**. Es la ejecución de un sistema operativo sobre otro sistema operativo a través de un hipervisor o capa de virtualización. Si la aplicación de virtualización implementa traducción binaria, se podrán ejecutar máquinas virtuales cuyo sistema operativo haya sido compilado para harware y juego de instrucciones diferente al de la máquina física [3].

- **Emulación**. Replica una arquitectura hardware al completo: procesador, juego de instrucciones, periféricos hardware, etc [3]. En comparación con la simulación, la emulación respresenta sistemas más completos, ya que, imita parte de la realidad y otra parte la modela. Por ejemplo, en un simulador de vuelo estaríamos en casa con un joystick y una aplicación de ordenador, mientras que, un emulador, estaríamos dentro de una cabina de un avión real con todos los sistemas reales recreando un vuelo. La emulación se aproxima bastante a la realidad, en caso de accidente en un vuelo real se produciría una tragedia, mientras que en el emulador se reiniciaría el vuelo.
- **Virtualización completa**. Se representa un entorno completo de hardware para cada máquina virtual. Por lo tanto, cada máquina virtual dispone de su propio conjunto de recursos de hardware virtual asignados por el hipervisor [13]. Esto significa que la máquina virtual no tiene acceso al harware físico del ordenador.
- **Paravirtualización**. La capa de abstracción no representa un entorno completo de hardware, sino que, el hipervisor ofrece una interfaz software para que los dispositivos virtuales accedan al hardware físico [13].
- **Virtualización a nivel de Kernel**. Es la ejecución de máquinas virtuales y otras instancias de sistemas operativos en el espacio de usuario del núcleo de Linux. Para ello, el núcleo de Linux es convertido en hipervisor [3].

2.2 Interfaces linux para virtualización de redes

En esta sección se van explicar los dispostivos e interfaces de red más conocidos y usados en Linux para la virtualización de redes.

- **Bridge**

  Un bridge es una dispositivo de la capa de Enlace que reenvía tráfico entrante, basado en las direcciones MAC, a través de sus interfaces. La decisión de reenvío está basada por las tablas de direcciones MAC que se van construyendo a medida que el bridge aprende [14].

  Puede soportar el protocolo STP para la eliminación de bucles de red a nivel de capa de Enlace, el filtrado de VLAN y Multicast Snooping [15].

  Uno de los usos de los bridges es su utilización para los programas de virtualización como VMware o VirtualBox, con el objetivo de compartir la interfaz de red física con una o más interfaces de redes virtuales. Estos programas aplican este concepto para conectar la interfaz de red virtual de una máquina virtual con la interfaz de red física, de esta manera, dicha máquina se puede conectar a Internet como se observa en la figura 2.3.

  Para crear un dispositivo bridge se usa el siguiente comando:

```
1  sudo ip link add <bridge_name> type bridge
```

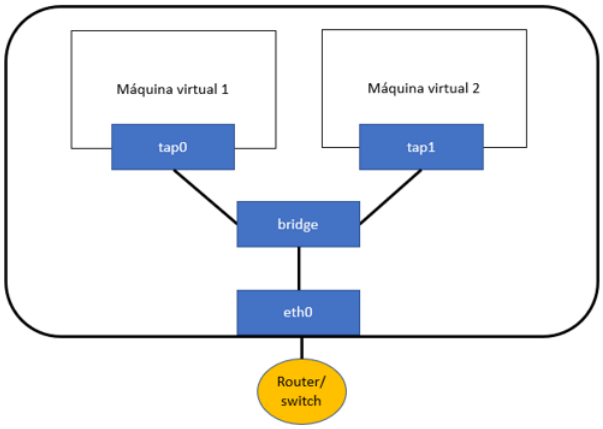


Figura 2.3: Conexión de dispostivos tap con un bridge

- **TUN/TAP**

  Tun/tap es una interfaz de red virtual que simula las conexiones físicas dentro del kernel del sistema operativo [16]. Puede interpretarse como un dispositivo punto a punto o como una tarjeta de red virtual que recibe y envía paquetes a través de un programa de espacio de usuario [17]. La interfaz tun procesa solo paquetes de la capa de Red mientras que la interfaz tap únicamente los de capa de Enlace [16].

  La figura 2.3 es un ejemplo de uso de las interfaces tap. Generalmente, en QEMU se utilizan interfaces tap como tarjetas de red virtual para conectar la máquina virtual a un bridge.

  Para crear una interfaz tun y tap se usan los siguientes comandos:

```
sudo ip tuntap add <tun_name> mode tun
sudo ip tuntap add <tap_name> mode tap
```

- **Bond**

  Bond es una interfaz capaz de agregar múltiples interfaces de red en una sola interfaz lógica como se observa en la figura 2.4. Su fin es garantizar que un sistema siempre esté accesible a través de una red informática. De este modo, se puede conseguir una alta disponibilidad y redundancia en los equipos informáticos donde realmente la conectividad sea un factor crítico [15].

  El comportamiento de una interfaz bond depende del modo de trabajo:

  – **Modo 0 (Balance-RR)**. Este modo es Round Robin, es decir, los paquetes de red entrantes son rotados entre cada una de las interfaces que forma la interfaz bond. Permite distribuir la carga y la tolerancia de fallos, sin embargo, no resulta un modo práctico, ya que, desde la perspectiva de la tabla ARP e IP las direcciones

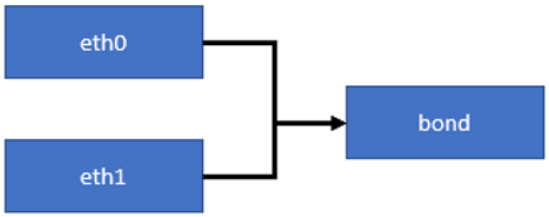


Figura 2.4: Interfaz Bond

MAC van saltando de una interfaz a otra [18].

- **Modo 1 (Active-Backup)**. En esta modalidad, una interfaz siempre está actica y la otra está deshabilitada, es decir, está en espera por si la interfaz activa falla. En el momento de fallo de la interfaz activa, la interfaz en espera se activa. De esta manera se asegura conexión en caso de que una interfaz falle [18].
- **Modo 2 (Balance-XOR)**. Se basa en el siguiente cálculo:

$$\frac{(Src_{MAC} \bigoplus Dst_{MAC})}{Num_{interfaces_{slaves}}} \tag{2.1}$$

La fórmula determina la interfaz por la que se enviarán los paquetes. Este modo brinda balanceo de carga y tolerancia de fallos.

- **Modo 3 (Broadcast)**. Transmite todas las tramas por todas las interfaces slave. Este modo solo expone tolerancia a fallos [18].
- **Modo 4 (802.3ad)**. Es el denominado de Dynamice Link Aggregation, también conocido como "port trunking". Permite alta disponibilidad y ofrece aumento de velocidad. El switch debe de soportar el "port trunking" y necesita compatibilidad con "ethtool" [18].
- **Modo 5 (Balance-TLB)**. Se determina la interfaz por donde se trasmite el paquete según la carga de las interfaces, de esta manera, se produce un balanceo de carga. El balanceo de la carga se determina sólo en la transmisión, no en la recepción. Para calcular la carga es necesario el soporte de "ethtool" [18].
- **Modo 6 (Balace-ALB)**. En este modo, se escoge la interfaz por donde se transmite el paquete según la carga tanto de transmisión como de recepción [18].

Para crear una interfaz bond se usa el siguiente comando:

```
1   sudo ip link add <bond_name> type bond miimon 100 mode <bond_mode>
```

- **Team**

La interfaz para realizar Teaming, ver figura 2.5, es similar a la interfaz Bond. Ambas agregan dos o más interfaces en una sola interfaz lógica. La diferencia entre ambas es que la interfaz Team provee un driver por cada interfaz agregada a la interfaz Team, además provee más funcionalidades como la monitorización de enlace para IPv6, balanceo de cargas para LACP [19]. Se puede interpretar esta interfaz como una mejora de la interfaz Bond.

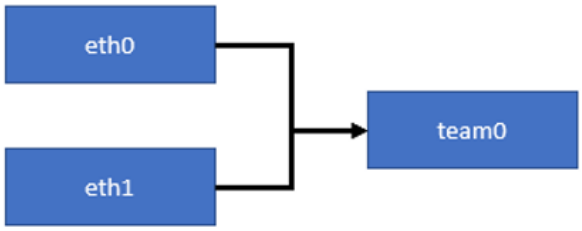


Figura 2.5: Interfaz Team

Para crear una interfaz team se usa el siguiente comando:

```
sudo teamd -o -n -U -d -t <team_name> -c '{"runner": {"name": "activebackup"},"↩
    ↪ link_watch": {"name": "ethtool"}}'
```

- **Virtual Local Area Network (VLAN)**

  Las redes de área local virtual permiten crear redes lógicamente independientes sobre la red de área local física usando switches que requieren compatiblidad VLAN [1].Como se puede observar en la figura 2.6, para generar las VLAN, los puertos de los switches se dividen en grupos y cada grupo es una VLAN diferente. Los dispostivos conectados a una VLAN sólo pueden comunicarse entre ellos.

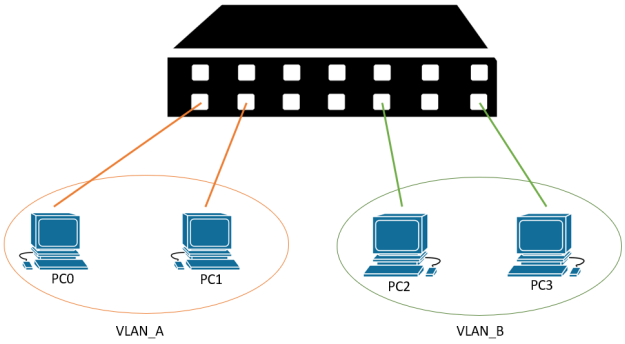


Figura 2.6: VLAN

Las redes VLAN suponen un aislamiento de tráfico y una mejora de la seguridad, ya

que solo los dispostivos pertenecientes al grupo pueden comunicarse. Además, mejora la movilidad de los dispostivos de una red a otra: si un usuario se mueve de un grupo a otro, es decir, de una subred a otra, habría que modificar el cableado para que el usuario se conectase al switch correspondiente, mientras que con esta tecnología, sólamente habría que reconfigurar el software del switch para que el puerto esté asociado a otro grupo. [1]

Si un PC de la *VLAN_A* de la figura 2.6 quiere comunicarse con otro PC de la *VLAN_B*, según lo que se ha definido antes no se podría, pero configurando un puerto del switch para que tenga acceso a las dos VLAN y se conecta a un router, ambas VLAN pueden comunicarse. En este ejemplo no tiene lógica, sin embargo, en una red más extensa, por ejemplo una empresa con muchos departamentos, en los que cada departamento supone una VLAN, unos departamentos se conectarán con unos y otros con otros, sí que es de utilidad para que haya comunicación entre determinados departamentos.

Continuando con el ejemplo, supongamos que *PC0* y *PC2* están conectados al mismo switch pero pertenecen a VLANs diferentes, al igual que sucede con *PC1* y *PC3*, como se puede observar en la figura 2.7. *PC0* necesita comunicarse con *PC1* ya que pertenecen al mismo departamento, por tanto a la misma VLAN pero, al no estar conectados al mismo switch no hay comunicación ya que los switches no están conectados físicamente.

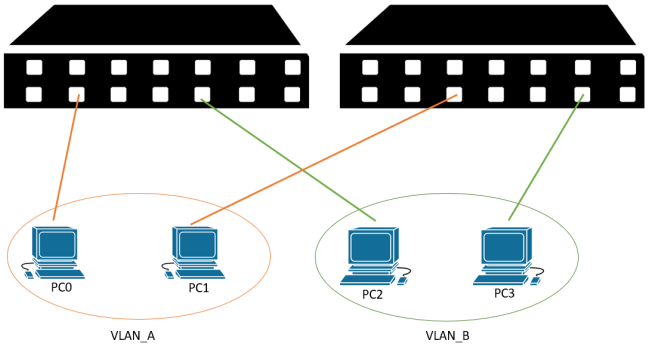


Figura 2.7: Dos switches y dos VLAN

En este punto se introduce un concepto denominado ***troncalización VLAN***. Un puerto del switch se configura como *puerto troncal* con el fin de enlazar switches [1]. En la figura 2.8 el puerto superior de la derecha es un puerto troncal que se conecta con el puerto superior izquierdo del otro switch consiguiendo la interconexión entre los dos switches. Cuando *PC0* envía una trama a *PC1*, la trama llega al otro switch a través del *enlace troncal*, pero el switch no sabe a qué VLAN pertenece dicha trama, por eso el IEEE definió una trama especial para la comunicación entre los switches VLAN, la trama ***802.1Q***.

La trama *802.1Q* está formada por la trama *Ethernet* estándar más una *etiqueta VLAN* de 4 bytes añadida a la cabecera. De esta manera, el switch emisor añade la *etiqueta VLAN* a la trama y el switch receptor la analiza y la elimina para enviar la trama

Ethernet al dispositivo de la VLAN correspondiente [1].

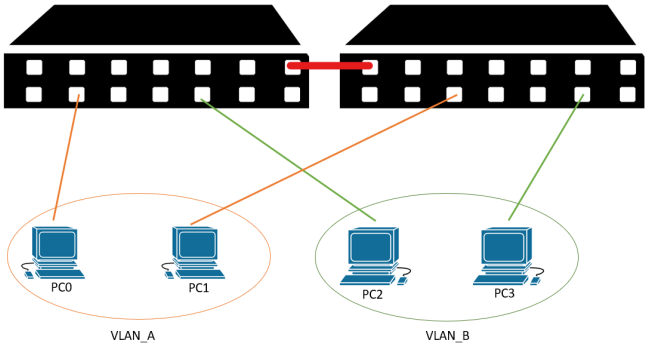


Figura 2.8: Conexión de dos switches VLAN con dos VLAN

Para crear interfaces VLAN mediante comandos se toma de referencia la figura 2.9. A partir de la interfaz *eht1* se crean las interfaces VLAN*eth1.1* y *eth1.2*:

```
sudo ip link add eth1 name eth1.1 type vlan id 101
sudo ip link add eth1 name eth1.2 type vlan id 102
```

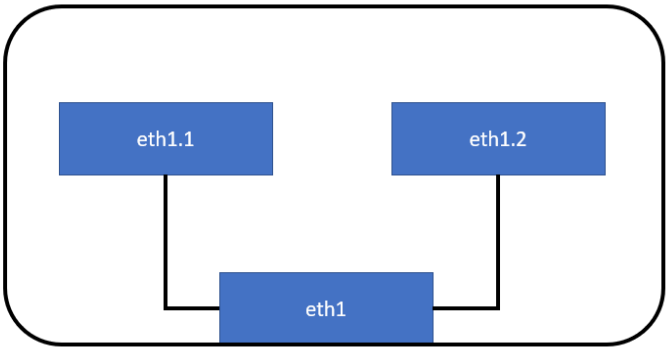


Figura 2.9: Interfaces VLAN

- **Virtual Extensible Local Area Network (VXLAN)**

  VXLAN es un protocolo de tunelización para transportar el tráfico de la capa de enlace sobre la capa de red, es decir, envía tramas Ethernet sobre redes IP a través de un tipo de encapsulación denominada *MAC in UDP*. La trama *Ethernet* que contiene la dirección MAC origen y destino original se le añade la cabecera VXLAN [20].

  Esta tecnología surge con el fin de evitar problemas que sucedían en los centros de datos donde se empleaba la virtualización de servidores. Estos centros de datos utilizaban miles de servidores, lo que daba lugar a problemas relacionados con el tamaño de las

direcciones MAC y con el número limitado de *4000* VLANs [21]. Para superar este límite, VXLAN utiliza como identificador de red una palabra de 24 bits, en vez de 12 bits como se tenía antes, lo que permite disponer de más de *16 millones* de redes virtuales [20].

Para crear una interfaz VXLAN se usa el siguiente comando:

```
sudo ip link add <vxlan_name> type vxlan id <number_id> group 239.1.1.1 dev <interfaz-ethernet> dstport 4789
```

- **MACVLAN**

  Mavcvlan es una interfaz de red soportada por el kernel de Linux. Esta interfaz permite generar subinterfaces de red con una dirección MAC, que es generada aleatoriamente, y dirección IP. Las subinterfaces se sostienen a través de la interfaz de red física pero no son capaces de comunicarse directamente con ella [15].

  Los dispositivos como máquinas virtuales o contenedores pueden enlazarse a una subinterfaz para conectarse directamente a la red física.

  Las subinterfaces operan en cuatro modos diferentes:

  – **Macvlan privada**. Las subinterfaces creadas no pueden comunicarse directamente, ver figura 2.10. En caso de que el paquete salga hacia la red y un switch lo reenvíe a la otra subinterfaz, esta lo eliminará [15].

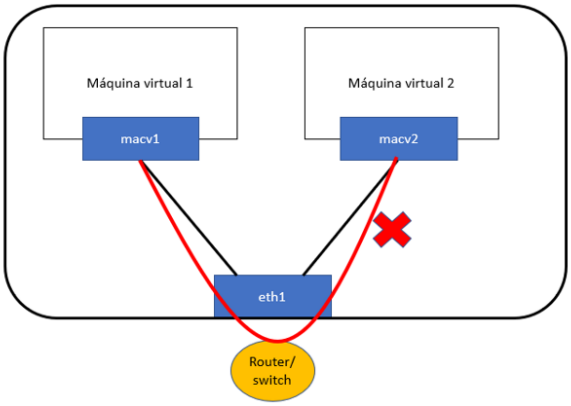


Figura 2.10: Interfaz Macvlan privada

  – **Macvlan VEPA**. Las subinterfaces no pueden comunicarse directamente, ver figura 2.11, al igual que en el modo privado, pero sí que permite una conexión indirecta a través de un switch que soporte IEEE 802.1Qbg [15].

  – **Macvlan Bridge**. Las subinterfaces se pueden comunicar directamente entre ellas sin salir hacia la red física, ver figura 2.12, es decir, se realiza una conexión puente entre las subinterfaces [15].

  – **Macvlan Passthru**. Permite la conexión directa entre la interfaz física y la subinterfaz [15], ver figura 2.13.

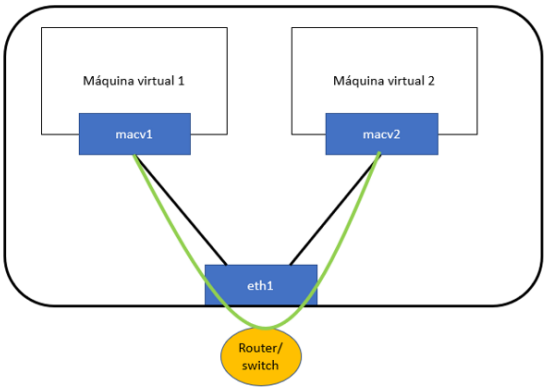


Figura 2.11: Interfaz Macvlan vepa

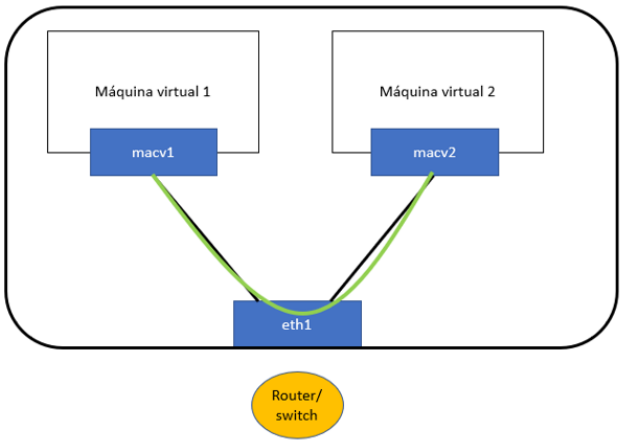


Figura 2.12: Interfaz Macvlan Bridge

Para crear una interfaz macvlan se usa el siguiente comando:

```
sudo ip link add <macvlan_name> link eth0 type macvlan mode <macvlan_mode>
```

- **IPVLAN**

  Las interfaces ipvlan son un nuevo dispositivo en el kernel de Linux que comparten la misma dirección MAC como se muestra en la figura 2.14. El driver del Kernel de Linux inspecciona la dirección IP de los paquetes para decidir por qué interfaz virtual serán procesados [15].

  Existen dos modos operativos, el modo *L2*, cuya interfaz física realiza la función de un switch, es decir, que actúa como macvlan modo bridge; y el modo L3, cuya interfaz física realiza la función de un router.

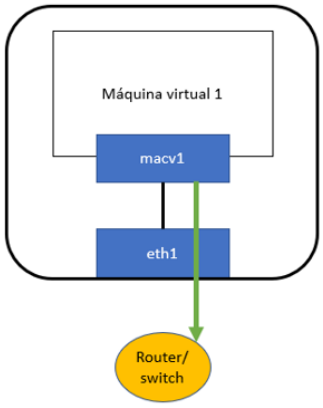


Figura 2.13: Interfaz Macvlan Passthru

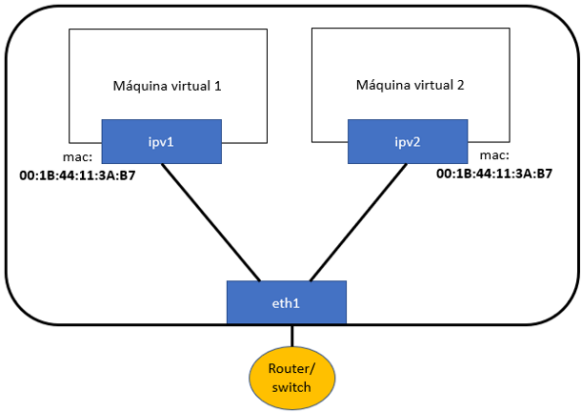


Figura 2.14: Interfaz IPVLAN

Para crear una interfaz ipvlan se usa el siguiente comando:

```
sudo ip link add name <ipvlan_name> link eth0 type ipvlan mode <ipvlan_mode>
```

- **VETH**

  Los dispositivos veth son interfaces de red virtual con el objetivo de conectar otros dispositivos de red. Actúan de puente entre dos espacios de nombres o entre un espacio de nombre y un dispositivo de red físico, pero también se pueden usar como dispositivos de red únicos [22].

  Un ejemplo de uso, es la conexión de red entre contenedores, como se muestra en la figura 2.15. Los contenedores usan espacios de nombres de red. Cada contenedor crea un

espacio de nombre de red y se utilizan los pares veth para interconectar los contenedores.

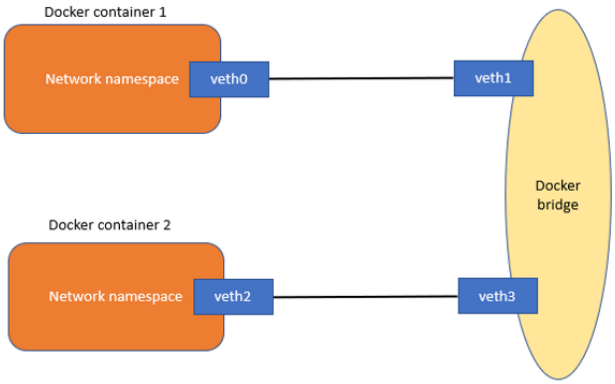


Figura 2.15: Conexión de contenedores mediante VETHS

Los dispositivos veth son creados en pares interconectados, es decir, cuando es creado uno, se crea otro en el extremo y se enlazan, como se pude ver en la figura 2.16.

Figura 2.16: Interfaz VETH

Para crear interfaces veth se usa el siguiente comando:

```
sudo ip link add dev veth0 type veth peer name veth1
```

- **MACVTAP/IPVTAP**

  Los dispositivos macvtap e ipvtap son un nuevo driver, con el objetivo de simplificar la virtualización del bridge [15], es decir, con estos dispositivos se puede obtener la misma comunicación que la combinación tun/tap y bridge como se puede ver en la figura 2.17

  El comando para crear una interfaz macvtap es el siguiente:

```
sudo ip link add link <interfaz-ethernet> name <macvtap_name> type macvtap
```

  El comando para crear una interfaz ipvtap es el siguiente:

```
sudo ip link add link <interfaz-ethernet> name <ipvtap_name> type ipvtap
```

- **MACSEC**

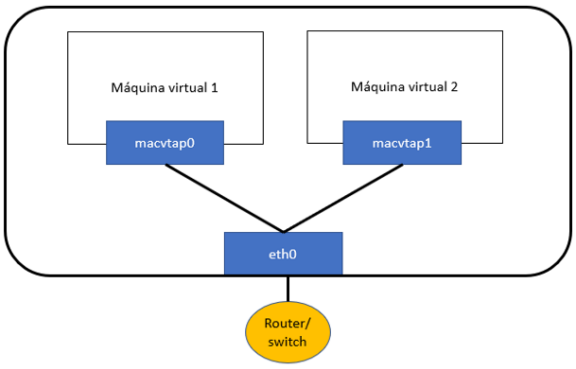


Figura 2.17: Interfaz MACVTAP

"MACsec, Media Access Control (MAC) Security, es un estándar definido en IEEE 802.1AE, que especifica un conjunto de protocolos que proporcionan medidas de seguridad para la protección de los datos en la capa de enlace (L2 OSI) de redes de área local (LAN) tipo Ethernet" [23]. MACsec no solamente protege tráfico IP sino también tráfico ARP, DHCP, LLDP y LACP [15].

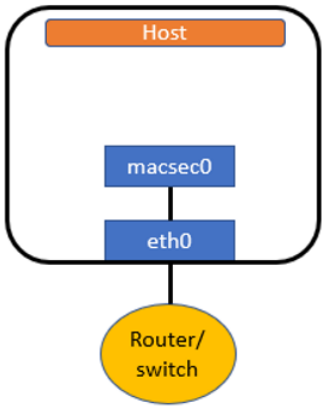


Figura 2.18: Interfaz MACSEC

El comando para crear una interfaz MACsec como se ve en la figura 2.18 es el siguiente:

```
sudo ip link add <vcan_name> link <interfaz-ethernet> type macsec
```

- **VCAN**

  El dispositivo virtual CAN es parecido a la interfaz loopback. Ofrece una interfaz CAN local virtual [24]. Es usado cuando se quiere testear el protocolo CAN en el host local.

  "La interfaz CAN o bus CAN es un protocolo basado en mensajes diseñado para permitir

que las unidades de control electrónico que se encuentran en los automóviles actuales, así como otros dispositivos, se comuniquen entre sí de manera confiable y basada en prioridades. CAN está respaldado por un amplio conjunto de estándares internacionales bajo ISO 11898" [25].

El comando para crear una interfaz vcan es el siguiente:

```
sudo ip link add dev <vcan_name> type vcan
```

- **VXCAN**

  VXCAN es un dispositivo similar al veth. Cuando se quiere enviar mensajes CAN de una MV o espacio de nombres de red a otra, se requiere interfaces VXCAN. También se crean en pares, uno para extremo [15].

  El comando para crear el par VXCAN es:

```
sudo ip link add <vxcan_name_A> type vxcan peer name <vxcan_name_B>
```

- **NLMON**

  NLMON abreviatura de Netlink Monitor es un dispositivo para analizar los mensajes netllink [15]. Es usado con la intención de realizar pruebas de diagnóstico cuando se detecta algún tipo de fallo en el proceso de comunicación entre el espacio de usuario y el kernel.

  Se considera un mensaje netlink a la información que se envía a través de los túneles netlink que conectan procesos en el espacio de usuario con el kernel interno de APIs para módulos de kernel [26].

  El comando para crear el dispositivo NLMON es:

```
sudo ip link add <nlmon_name> type nlmon
```

- **Dummy**

  La interfaz dummy es una interfaz de red completamente virtual que simula un dispositivo para enrutar paquetes, pero, realmente no envía ni recibe paquetes [15]. Su uso fundamental es para realizar testeos y diagnósticos de red.

  El comando para crear la interfaz dummy es:

```
sudo ip link add <dummy_name> type dummy
```

- **Intermediate Functional Block (IFB)**

  Es una interfaz que actúa como un concentrador Quality of Service (QoS) para diferentes orígenes de tráfico. Las tramas que van de una interfaz a otra han de pasar por esta interfaz con el fin de no perder el tráfico por desbordamiento [27].

  El principal uso de esta interfaz es para crear un cola de trafico entrante y moldear dicho tráfico según congestión.

Los comandos para crear una interfaz IFB para una cola Stochastic Fairness Queuing (SFQ) son los siguientes:

```
modprobe ifb
sudo ip link set <ifb_name> up
tc qdisc add dev <ifb_name> root sfq
```

2.3 Protocolos

Esta sección va a tratar sobre los protocolos usados en este TFG. En primer lugar se explica el protocolo Dynamic Host Configuration Protocol (DHCP) y en segundo lugar se expone el protocolo Spanning Tree Protocol (STP).

2.3.1 Dynamic Host Configuration Protocol DHCP

Una vez que la red ha recibido un bloque de direcciones IP, se ha de asignar a los dispositvos una dirección IP. Para ello, se va a explicar cómo un dispositivo, ya sea un *router* o *dispostivo final*, puede obtener una dirección IP.

Un administrador de red configura manualmente las direcciones IP a los dispostivos. En una red con diez dispostivos es una tarea asequible, pero en redes mucho más grandes, por ejemplo, con quinientos dispostivos, se convierte en una tarea tediosa, por eso, generalmente, la asignación y configuración de las direcciones IP se realiza de manera automática a través del protocolo DHCP.

DHCP es un protocolo para asignar de manera automática direcciones IP que deriva de BOOTP [RFC 2131] [28]. Funciona con una arquitectura *cliente-servidor*. El cliente solicita una dirección IP al servidor DHCP. En caso de que en la red no exista un servidor DHCP, será necesario un agente de retransmisión DHCP, por ejemplo, un router, que conozca la dirección de un servidor DHCP [1].

Un adminisitrador de red puede configurar DHCP de tres modos diferentes:

- **Modo manual**. El administrador de red configurará a mano las direcciones IP de cada dispositivo que se quiera conectar a la red, por lo que un dispositivo siempre tendrá configurada la misma dirección IP hasta que el administrador de red cambie manualmente su configuración. Es un mecanismo rígido y poco flexible, pero de utilidad para controlar los dispositivos conectados en la red [28].
- **Modo automático**. El servidor DHCP está configurado para asignar de manera automática una dirección IP la primera vez que el cliente DHCP solicita una dirección IP. Esta dirección será permanente hasta que el cliente DHCP la libere [28].
- **Modo dinámico**. El servidor DHCP se configura para que asigne una dirección IP temporal [28], es decir, por un periodo de tiempo configurable. De esta forma, cuando se agote el tiempo establecido, la dirección IP del dispositivo es revocada y deberá solicitar una nueva.

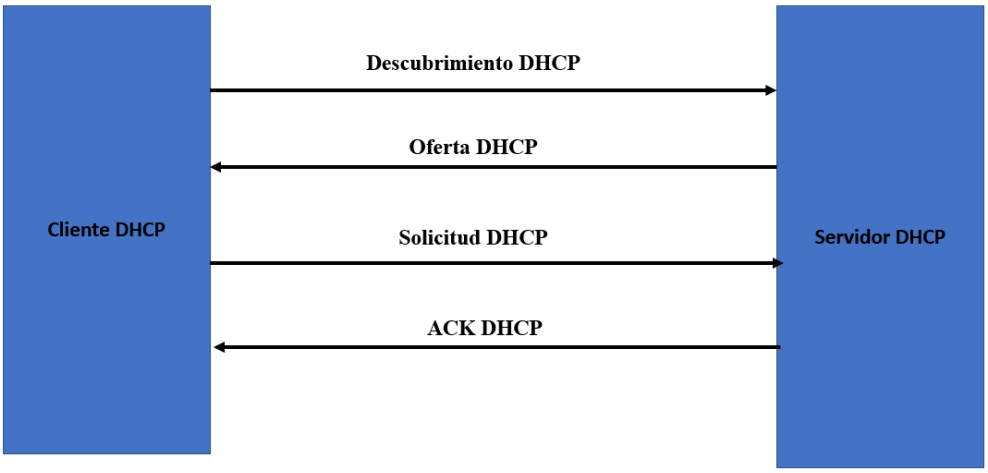


Figura 2.19: Fases DHCP

2.3.1.1 Proceso DHCP

El proceso para asignar una dirección IP se realiza en cuatro pasos, como se muestra en la figura 2.19:

- **Descubrimiento DHCP**. El primer paso que realiza un host recién llegado a la red es encontrar un servidor DHCP o dispositivo que actúe de servidor DHCP. Esto se realiza mediante un *mensaje de descubrimiento DHCP* dentro de un paquete UDP con puerto destino *67*, encapsulado en un datagrama [1]. El datagrama IP tiene asignada la dirección IP destino de difusión *255.255.255.255*, y como dirección IP origen *0.0.0.0*, ya que no tiene asiganda una direción IP.
- **Oferta(s) DHCP**. El datagrama IP encuentra un servidor DHCP quien responde al cliente DHCP con un *mensaje de oferta DHCP*. Este mensaje se difunde por la subred usando de nuevo la dirección IP de difusión *255.255.255.255* [1]. Un mensaje de oferta contiene una serie de parámetros para configurar la dirección IP: *ID de transacción*, *dirección IP propuesta*, *máscara de red*, *tiempo de arrendamiento de la dirección IP* [1]. En una subred, puede haber más de un servidor DHCP, por lo que a un cliente pueden llegarle varios *mensajes de oferta*.
- **Solicitud DHCP**. El cliente DHCP escogerá entre las ofertas llegadas y responderá al servidor de la oferta seleccionada con un *mensaje de solicitud*, que incluye los parámetros de configuración [1]. Implícitamente las ofertas recibidas de otros servidores son rechazadas.
- **ACK DHCP**. El servidor DHCP envía un *mensaje ACK DHCP* confirmando los parámetros de configuración [1].

Una vez finalizado este proceso, el cliente DHCP tendrá configurada una dirección IP que

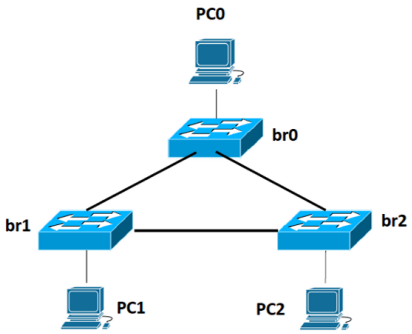


Figura 2.20: Topología bucle

podrá usar durante el tiempo de arrendamiento. Finalizado este tiempo, DHCP también tiene un mecanismo con el fin de renovar el tiempo de arrendamiento y así poder usar la dirección IP [1].

2.3.2 Spanning Tree Protocol STP

2.3.2.1 bucles de capa 2

Un bucle, también conocido como tormenta de broadcast, se produce cuando existe más de una ruta entre dos switches [29]. En la figura 2.20 br0, br1, br2 tienen dos rutas distintas para llegar al mismo destino, dando lugar a la formación de un bucle.

Supongamos que en la figura 2.21 *PC0* envía una trama broadcast, por ejemplo, una petición Address Resolution Protocol (ARP). La trama llega a *br0* y este la difunde hacia *br1* y *br2*. Cuando llega la trama a *br1*, la reenviará a través de sus interfaces, es decir, hacia *br2* y *PC1*. De la misma manera sucede en el otro sentido, *br2* difunde la trama que le ha llegado de *br0* hacia a *PC2* y *br1* y este último la reenvía hacia *PC1* y *br0*. Retomando la trama que llega a *br2*, éste la encamina hacia *PC2* y *br0*. En este punto, *PC1* ha recibido dos tramas de difusión y *PC2* otras dos tramas, además, los switches seguirían reenviando y recibiendo sus propias tramas broadcasst.

Al enviar una trama difusión, cada switch va a reenviar por todos los puertos de salida los paquetes que les llegue, de tal manera, que una misma trama difusión estará continuamente retrasmitiéndose, creando un bucle infinito.

Cuando hay más de un camino entre dos switches, la *tormenta de broadcast* provocará que las tablas ARP se hagan inestables. Además, también generará una sobrecarga en los procesadores de los switches. [29]

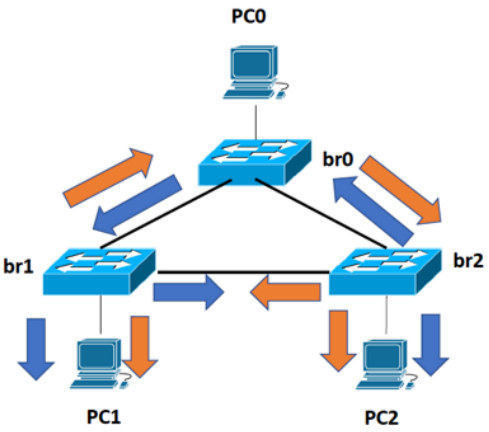


Figura 2.21: Proceso-bucle

2.3.2.2 Introducción STP

Una de las formas para la detección de los bucles es mediante el protocolo STP.

STP es un protocolo de red de capa 2, definido por el estándar IEEE 802.1D [30], cuya función principal es gestionar los enlaces entre los switches para la detección de bucles existentes en la topología de red. El protocolo observa constantemente la red, de forma que, cualquier error o agregación de un nuevo bridge o switch es detectado al instante. En el momento que lo detecta, el protocolo STP reconfigura los puertos de los switches o bridges para evitar enlaces redundantes en la ruta, y que aún tenga suficiente conectividad con el objetivo de que haya un sólo camino entre cada switch [30].

2.3.2.3 Estados de los puertos

Un puerto del switch recorre estos estados:

- **Blocking**. Es el estado de los puertos cuanno el protocolo STP ha sido habilitado con el fin de no retransmitir ninguna trama y de eliminar la capacidad de aprendizaje de direcciones MAC [31], pero si puede escuchar trmas BPDU con la finalidad de detectar cambios en la red [32].
- **Listening**. Es un estado transitorio en el que los puertos sólamente puede enviar y recibir tramas BPDU para la elección del *switch ráiz*, *puertos raíz* y *puertos designados* [32].
- **Learning**. En este estado, el switch puede aprender direcciones y crear tablas, pero aún no tiene la capacidad de reenviar tramas [32].
- **Forwarding**. El puerto tiene la capacidad funcional del estado ***learning*** para retransmitir tramas [31].

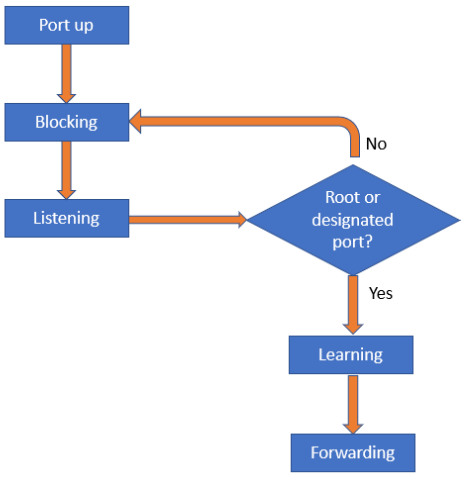


Figura 2.22: Estados de los puertos STP

En la figura 2.22 se observan los estados que ha de pasar un puerto con el objetivo de eliminar el bucle.

2.3.2.4 Funcionamiento STP

STP usa un algoritmo denominado Spannig Tree Algoirthm (STA) [29] para determinar qué puertos del switch pasan a estado bloqueado y así prevenir los bucles. Los switches se comunican a través de una serie de mensajes llamados Bridge Protocol Data Units (BPDU).

El funcionamiento STP se explica mediante los siguientes pasos:

1. **Elección del switch raíz**

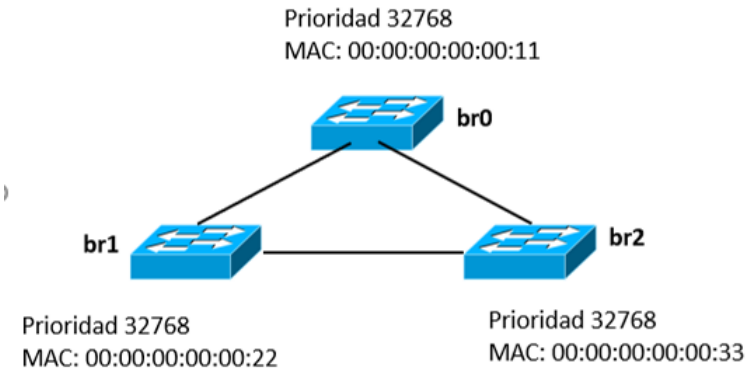


Figura 2.23: Elección del switch ráiz

El primer paso es la elección del *switch raíz* que es el de referencia para la topología STP [32].

El *switch raíz* se elige a través del campo *Bridge ID* de la trama BPDU de configuración. Cada switch envía tramas BPDU con un *Root ID* igual que su propio *Switch ID*. Cada switch recibe las tramas BPDU envidas desde los otros switches y las analizan para comprobar si el valor del campo *Root ID* es menor [29].De ser así, cada switch reemplaza el valor de *Root ID* por el menor valor recibido de dicho campo. El switch cuyo valor de *Root ID* más bajo será elegido *switch raíz* [32].

En caso de que varios switches tengan el mismo valor más bajo de *Root ID*, el switch raíz será el que tenga el valor más bajo de dirección MAC [32].

En el ejemplo de la figura 2.23, se aprecia que todos los switches tienen la misma prioridad, pero br0 contiene la dirección MAC más baja correspondiéndose con el *switch raíz*.

2. **Identificación del puerto raíz de cada switch**

   El segundo paso del proceso STP es la identificación de los puertos raíz de cada switch. El *puerto raíz* es el puerto del switch con menor coste de ruta desde dicho switch hasta el *switch raíz*[32], por lo tanto, cada switch, excepto, el *switch raíz*, tiene un solo *puerto raíz*.

   En la figura 2.24 se observa que todos los enlaces son de coste "*4*", es decir, son enlaces de *1Gbps*. Para determinar el *puerto raíz* de *br1*, se calcula el coste acumulativo desde dicho switch hasta el *switch raíz*. Se calculan dos caminos, uno de coste "*4*" y otro de coste "*8*" (coste "*4*"+ coste "*4*").

   Se selecciona el puerto cuyo camino al *switch raíz* es de menor coste: el de coste "*4*".

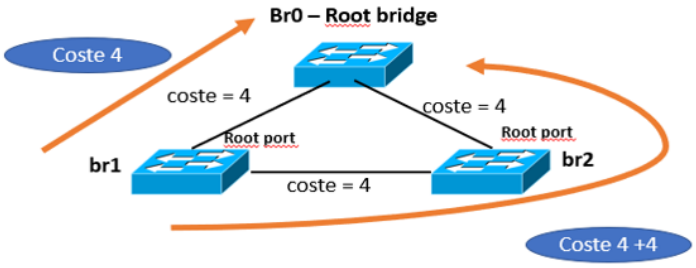


Figura 2.24: Identificación del puerto raíz

3. **Identificación de los puertos designados**

   El tercer paso del proceso STP es la identificación de los *puertos designados*. El *puerto designado* se determina mediante el enlace con el menor coste acumulativo para llegar al *switch raíz* [32]. El *puerto designado* nunca estará en estado *bloqueado* ya que se encarga del reenvío de las tramas por el enlace.

   En caso de que dos puertos del switch tengan el mismo coste acumulativo al *switch raíz*, se determina mediante el menor *bridge ID* [29].

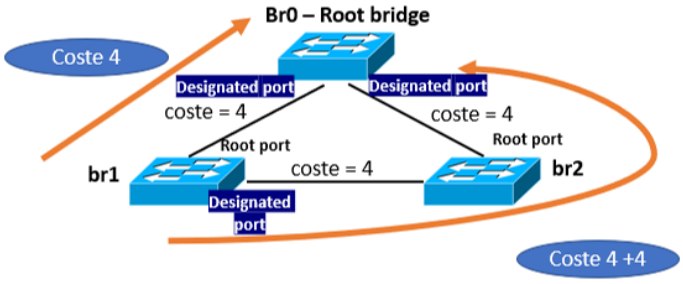


Figura 2.25: Identificación de los puertos designados

Si dos puertos son elegidos *puertos designados*, se formará un bucle, por tanto, uno de esos puertos se tendrá que eliminar o establecer el puerto en un estado de *bloqueo*.

En la figura 2.25, el *puerto designado* es el puerto que queda libre, pero aún no se ha eliminado el *bucle*, para ello hay que establecer, un puerto a estado *bloqueado*.

En resumen, para determinar el *puerto raíz* y *designado* de cada bridge, se utiliza la siguiente secuencia de criterios en orden:

a) Menor Root ID

b) Menor Root path cost

c) Menor Bridge ID

d) Menor Port ID

4. **Estado bloqueado de los puertos**

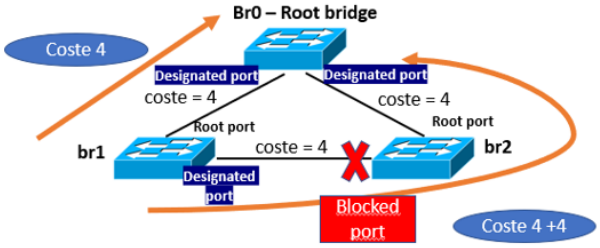


Figura 2.26: Estado bloqueado de los puertos

En un mismo enlace, no pueden existir dos puertos *designados* y cualquier puerto que no sea un *puerto raíz* o un *puerto designado* se establece en estado *bloqueado* [29]. Con este puerto *bloqueado* se rompe el *bucle* y se completa el proceso *Spannig Tree*.

En la figura 2.26, para establecer comunicación entre *br1* y *br2*, ha de pasar la trama a

través de *br0*, ya que el puerto de *br2* del enlace *br1-br2* está *bloqueado*, de tal manera que ya no se produce la tormenta de broadcast y la red puede funcionar de manera correcta.

2.3.2.5 BPDU

Una BPDU es una trama que contiene la suficiente información de los dispositivos para completar el cálculo del protocolo STP. Los switches envían dicha trama usando la dirección MAC del propio puerto como dirección de origen y una dirección MAC multicast de destino. La dirección MAC multicast es 01:80:C2:00:00:00 [32].

Existen dos tipos de BPDU:

- **BPDU de configuración**

  El formato de trama BPDU está formado por diversos campos:

  - **Protocol ID (2 bytes)**. El ID es el identifcador del protocolo que contiene el valor "*0x0000*" para identificar el IEEE 802.1D [33].
  - **Version (1 byte)**. Indica la versión del protocolo y contiene siempre el valor "*0x00*" [33].
  - **Message Type(1 byte)**. Indica el tipo de BPDU conteniendo el valor "*0x00*" que corresponde con la BPDU de configuración [33].
  - **Flags (1 byte)**. Campo de 8 bits que indica el objetivo de la trama BPDU. El bit de menor peso es el flag correspondiente a Topology Change (TC) y el bit de mayor peso es el flag correpondiente con Topology Change Acknowledge (TCA). Los demás bits están reservados [34].
  - **Root ID (8 bytes)**. Este campo indica el *switch raíz*. Los dos primeros bytes indican la prioridad y los restantes la dirección MAC. La primera vez que se enciende un switch, el campo *Root ID* contiene el mismo valor que el campo *Switch ID*. Sin embargo, como el proceso de elección continúa por la característica dinámica del protocolo, el menor *Switch ID* reemplaza el *Root ID* local para identificar el *Switch raíz* [33].
  - **Cost of path (4 bytes)**. El campo "Coste" indica el valor acumulativo de ruta desde el switch que envía la trama hasta el switch raíz. El campo se va actualizando a medida que la trama llega a cada switch del camino hasta alcanzar el switch raíz. [33]. El valor del enlace es inversamente proporcional a la capacidad del enlace [29].
  - **Bridge ID (8 bytes)**. Indica la prioridad y la dirección MAC del bridge que envía la trama. Este campo permite identificar dónde se origina la trama BPDU original, como también identificar los múltiples caminos desde el switch al *switch raíz*. La prioridad por defecto es "*32768*" [32].
  - **Port ID (2 bytes)**. Indica el número de puerto por el cual se envía los mensajes de configuración. Este campo permite que los bucles sean detectados y corregidos [33].

  - **Message Age (2 bytes)**, Indica el tiempo que transcurre desde que el *switch raíz* envió el mensaje de configuración [33].
  - **Max Age (2 bytes)**. Indica cuándo la configuración de mensaje debería eliminarse. Cuando el valor del campo *Message Age* alcanza el valor del campo *Max age*, el switch elimina su actual configuración y se inicia un nuevo proseo de selección para determinar un nuevo *switch raíz*. El valor por defecto son *20 segundos*, pero puede cambiarse. Como mínimo *6 segundos* y como máximo *40 segundos* [33].
  - **Hello time (2 bytes)**. Indica el tiempo de espera del switch raíz entre dos tramas de configuración BPDU. Por defecto son dos segundos, pero se puede modificar dicho tiempo. El valor ha de estar entre *1* y *10 segundos* [33].
  - **Forward delay (2 bytes)**. Indica el tiempo pasado entre los estados de escucha y aprendizaje. El valor por defecto son *15* segundos, pero puede ir desde los *4* hasta los *30* segundos [33].

- **Topology Change Notification (TCN) BPDU**

  TCN BPDU es una trama para notificar los cambios de topologías de una red, no contiene datos sobre la red. Estas son envíadas por los switches que no son raíz cuando detectan un cambio en la red. Estos cambios ocurren cuando un puerto pasa a estado *Forwarding*, o también cuando un puerto en estado *Learning* o *Forwardiing* pasa a un estado *Blocking*. También se genera un cambio de topología cuando no llegan las BPDUs a los switches por la caída de un enlace o un nodo de la red.

  Los campos de TCN BPDU son tres:

  - **Protocol ID (2 bytes)**. El ID es el identifcador del protocolo. Contiene el valor "*0x0000*" para identificar el IEEE 802.1D [34]
  - **Version (1 byte)**. Indica la versión del protocolo, en el caso estudiado el valor se ha fijado a "*0x00*" [34]
  - **Message Type(1 byte)**. Indica el tipo de BPDU cuyo valor para TCN BPDU se corresponde con 0x80 [34].

2.4 La Web Y HTTP

Internet al principio fue de uso exclusivo para Universidades y organizaciones gubernamentales hasta que se desarrolló una aplicación que atrajo al usuario personal: la Web.

La Web está formada por tres elementos principales: HTML, como lenguaje para crear el contenido de las páginas Web; HyperText Transfer Protocol (HTTP), protocolo para la comunicación entre los ordenadores de la Web; y la Uniform Resource Locator (URL), que es la dirección donde se localizan recursos de la Web [35]. Así que, en esta sección se va a explicar los conceptos básicos de HTTP, los servidores web así como algunos lenguajes y herramientas para desarrollar una aplicación web.

2.4.1 Servidores Web

Los navegadores web como *Google Chrome*, *Microsoft Edge*, *Firefox*, etc, son los clientes que solicitan el contenido de las páginas web, mientras que el servidor web se encarga del procesamiento de las peticiones y del envío de respuestas en forma de página web (texto, imágenes, audio, vídeos, etc.).

Un servidor web es una computadora que guarda y transmite datos vía Internet. Cuando un usuario entra en una página de Internet, su navegador se comunica con el servidor enviando y recibiendo información con la finalidad de que se visualice en la pantalla.

Existen una cantidad considerable de tipos de servidores web. Algunos de los más utilizados son:

- **Nginx**. Es el servidor más utilizado a día de hoy [36]. También es de código abierto, auque ofrece una versión comercial. Al principio solamente funcionaba como servidor HTTP, pero hoy en día también sirve como proxy inverso, balanceador de carga HTTP y proxy de correo electrónico [37]. La estructura del software es asíncrona y controlada por eventos lo que permite el procesamiento de muchas solicitudes al mismo tiempo [37]. También es altamente escalable, es decir, que sus servicios aumentan con el tráfico de sus clientes.
- **Apache**. Es un servidor HTTP de código abierto y gratuito para sistemas operativos modernos incluidos Windows y UNIX. Ofrece módulos de seguridad, autenticación, caching, reescritura de URLs, etc [38]. Además, es compatible con una gran cantidad de aplicaciones, lenguajes de programación, frameworks, etc. Su punto débil se encuentra en el desfase de arquitectura respecto a otros servidores, pero aun así, sigue siendo de los más populares y usados.
- **LiteSpeed**. Es un servidor web de pago compatible com muchas tecnologías que usa *Apache*, esto ayuda a su adaptación, ya que migrar *Apache* a *LiteSpeed* es más sencillo [39]. Está diseñado para atender más peticiones y de manera más rápida que los demás servidores ya que está basado en eventos y no en procesos [39].
- **Microsoft-IIS**. Internet Information Services (IIS) es más conocido como servidor, de hecho, está en el top 5, sin embargo es un conjunto de servicios que transforman un sistema *Microsoft Windows* en un servidor capaz de ofrecer servicios web, FTP, de correo electrónico, etc. Su arquitectura permite añadir y quitar extensiones con facilidad y cuenta con características de seguridad y mecanismos de autenticación [40].

2.4.2 HTTP

HTTP define la sintaxis y la semántica que utilizan los elementos de la Web para comunicarse [41]. Es un protocolo orientado a transacciones y está basado en la arquitectura cliente-servidor.

El protocolo HTTP se implementa través de dos programas: un programa al que se le denomina cliente, y otro, al que se le denomina servidor [1]. Ambos programas informáticos se comunican a través de mensajes HTTP [1].

HTTP es un protocolo de capa de Aplicación que utiliza TCP como protocolo de capa de transporte. El navegador contacta y abre una sesión de comunicación TCP con el servidor a través de una serie de interfaces denominadas socket [1]. Se crean dos socket: uno en el lado del cliente, que conecta el navegador con TCP, y otro en el lado del servidor para conectarlo con TCP. Por consiguiente, el navegador enviará la solicitud al servidor a través de su socket y llegará al servidor donde lo recogerá por su interfaz socket y enviará una respuesta al cliente.

Además, HTTP es un protocolo *sin estado* y *no conectivo*, es decir, que no almacena la información de los clientes ni tampoco las transacciones [1]

Dependiendo del tipo de aplicación y de cómo se soliciten las solicitudes al servidor exsiten dos tipos de comunicación HTTP: *HTTP con conexiones no persistentes* y *HTTP con conexiones persistentes* [1].

2.4.3 HTTP con conexiones no persistentes

Una conexión no persistente significa que no dura en el tiempo. Para el protocolo HTTP quiere decir que cada vez que solicite un objeto al servidor hay que establecer una nueva sesión TCP, por tanto cada conexión TCP solamente soporta una transacción HTTP (una petición y una respuesta) [1] como se puede observar en la figura 2.27. Además, se puede configurar de dos modos: en *"serie"*, que solamente existe una conexión TCP en cada momento, y en *"paralelo"*, que permite varias conexiones TCP abiertas la vez.

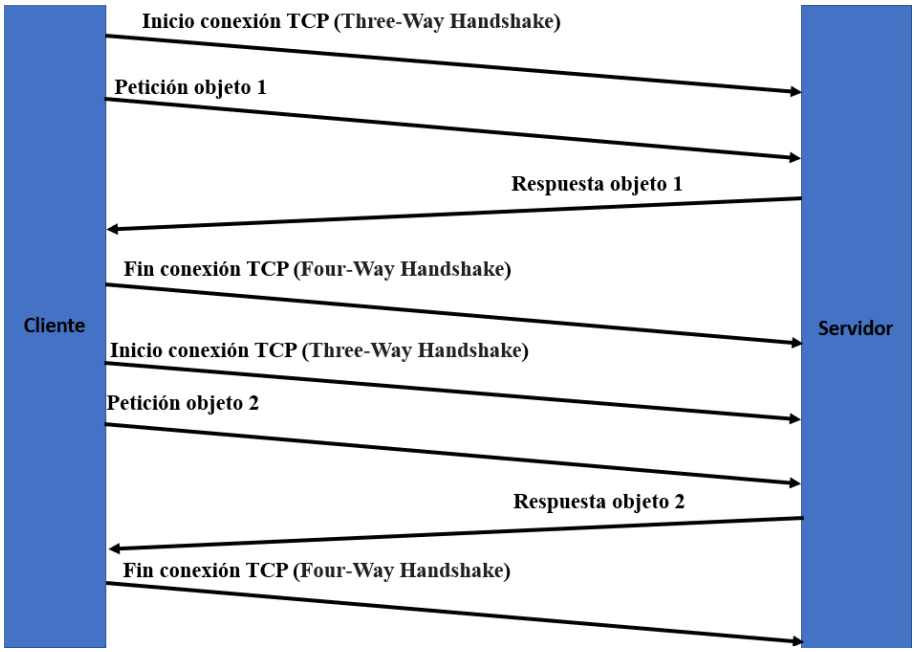


Figura 2.27: HTTP no persistente

Supongamos que accedemos a la siguiente página web inventada: *www.davidtfg.com*. El cliente solicita una conexión TCP al puerto *"80"*, puerto HTTP por el que está escuchando el

servidor. El servidor acepta la conexión del cliente y se crea el socket de conexión al servidor. Una vez establecida la conexión y creados los dos sockets, el proceso cliente ya está preparado para solicitar los objetos al servidor y envía un mensaje HTTP que contiene la ruta donde se localiza el archivo html: *index.html*. El servidor recibe dicha petición, encapusla el objeto en el interior de un mensaje de respuesta HTTP y se inicia el cierre de la conexión TCP, pero no se cierra hasta que el servidor se asegure de la entrega del mensaje al cliente. Cuando el cliente recibe el objeto solicitado, en este caso *index.html* y lo examina, se encuentra que tiene referencias a otros archivos HTML, vídeos, imágenes, etc, entonces, cierra la conexión TCP e inicia todo el proceso de nuevo para obtener todos los objetos contenidos en el archivo *index.html*

Las conexiones HTTP no persistentes presentan una serie de inconvenientes. Se han de asignar variables y buffer TCP cada vez que se establece una nueva conexión. Si por cada objeto se abre una nueva conexión, la cantidad de variables y bufferes es mayor, a consecuenia de esto, podría llegar a sobrecargar el servidor disminuyendo su rendimiento y no atendiendo todas las solicitudes.

2.4.4 HTTP con conexiones persistentes

Si en una conexión no persistente se establecía una conexión TCP diferente para cada objeto solicitado, en una conexión persistente sólo es necesario establecer una sola conexión para solicitar todos los objetos, ver figura 2.28. El servidor deja la conexión TCP abierta después de enviar la primera respuesta, así, las siguientes peticiones y respuestas que tienen lugar entre el mismo cliente y servidor pueden enviarse a través de la misma conexión [1]. Transcurrido un tiempo, que es configurable, sin que haya habido ninguna solicitud y respuesta, el servidor cierra la conexión TCP.

Existen dos modos de configurar las conexiones persistentes: *sin pipelining*, es decir, sin canalización, que sólamente se permite una transacción HTTP a la vez y *con pipelining*, es decir, con canalización, que permite varias transacciones HTTP a la vez.

2.4.5 Formato de los mensajes HTTP

Las especificaciones HTTP *[RFC 1945; RFC 2616; RFC 7540]* definen los dos tipos de mensajes HTTP: *mensajes de solictud* y *mensajes de respuesta* [1].

- **Mensaje de solicitud**

  Un mensaje de solicitud está escrito en código ASCII, por lo tanto, cualquier usuario que obtenga el mensaje, por ejemplo, mediante un sniffer de paquetes, puede saber cuál es el contenido de este. Comprende varias líneas: la primera, denominada *línea de solicitud* describe la petición; una serie de líneas opcionales llamadas *líneas de cabecera*, que describen el cuerpo del mensaje; una línea vacía y posteriormente el cuerpo del mensaje (opcional) que contiene los archivos de respuesta y los datos asociados a la petición [42].

  La *línea de solicitud* está compuesta por tres campos: el campo de método, la URL y

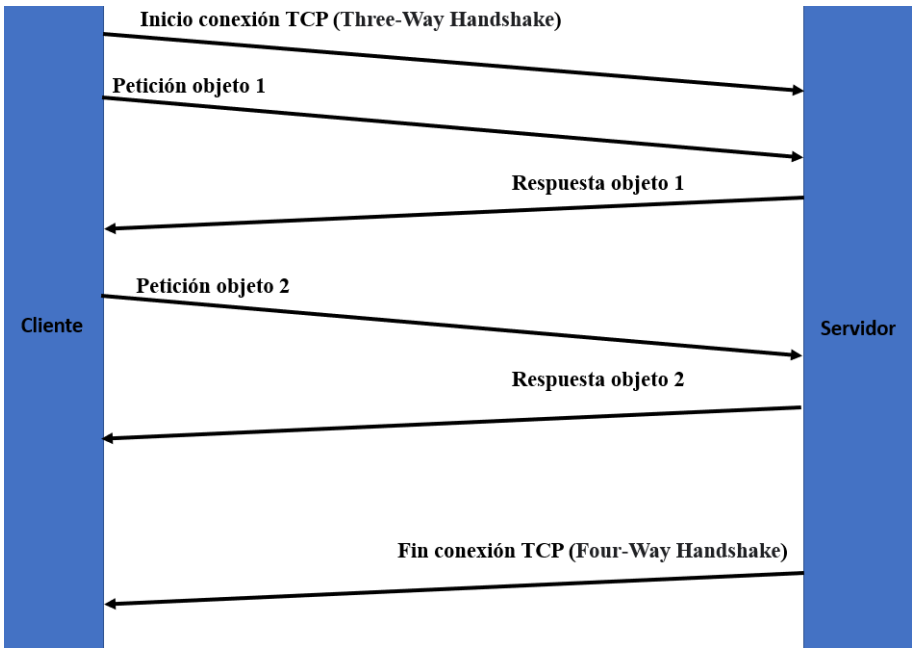


Figura 2.28: HTTP persistente

el campo de la versión HTTP [1].

Los métodos más significativos son:

- **GET**. Método para obtener los objetos del servidor.
- **POST**. Método para enviar información al servidor.
- **HEAD**. Método para obtener solamente el encabezado del objeto, es parecido a *GET*.
- **PUT**. Método para cargar un nuevo objeto.
- **DELETE**. Método para eliminar un objeto.
- **OPTIONS**. Método para consultar opciones.

Las *líneas de cabecera* siguen todas la misma estructura: una cadena de caracteres que indican el tipo de cabecera, seguido del carácter "*:*" y del valor de la cabecera. Existen muchos tipos de cabeceras como "*Host*", "*Connection*", "*User-agent*" y se pueden encontrar muchas más en la siguiente URL: "**https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers**" La cabecera completa, incluido el valor, ha de ser desarrollada en una única línea [43].

El cuerpo, que es la parte final del mensaje HTTP puede aparecer o no. En las peticiones para obtener datos como *GET*, *HEAD*, *DELETE*, o *OPTIONS* no necesitan cuerpo. Los cuerpos pueden estar divididos en dos tipos: cuerpos con un único dato, que se tratan de un sólo archivo definido por las dos cabeceras: *Content-Type* y *Content-Length*, y cuerpos con múltiples datos, que son aquellos que estan formados por diferentes objetos, como por ejemplo, formularios HTML [43].

Se puede observar un ejemplo de mensaje de solicitud en la figura 2.29.

Figura 2.29: Mensaje de respuesta HTTP capturado en Wireshark

- **Mensaje de respuesta**

  Un mensaje de respuesta también está formado por diversas líneas: la primera línea, denominada *línea de estado*, seis líneas de cabecera y después el cuerpo [1].

  La *línea de estado* contiene tres campos, el primero indica la versión de HTTP, el segundo se corresponde con un código de estado y el tercero, el texto del estado, que se trata de una corta descripción, a modo informativo, del significado del código de estado [43].

  El primer dígito del código de estado define la clase de respuesta:

  - **1xx**. No usado pero reservado para un uso futuro.
  - **2xx**. La petición se recibió exitosamente, entendida y aceptada.
  - **3xx**. El cliente tiene que realizar otra acción, como usar otro método, en vez de *GET* usar *HEAD* para completar la petición.
  - **4xx**. Error de cliente. La petición contiene una sintaxis errónea o no puede procesarse de manera correcta.
  - **5xx**. Error de servidor cuando la petición es válida.

  Las cabeceras del mensaje de respuesta tienen la misma estructura que la cabecera del mensaje de solicitud, algunos ejemplos de cabeceras son: *Connection*, *Date*, *Server*, *Last-Modified*, *Content-Length*, *Content-Type*, etc.

  En cuanto al cuerpo de los mensajes de respuesta, se dividen en tres grupos: cuerpos con un único dato de longitud conocida y definido en las cabeceras: *Contet-Type* y *Content-Lenght*; cuerpos con un único datos de longitud desconocida y codificado en partes indicadas en la cabecera: *Tranfer-Encoding* y por último, cuerpos con múltiples

datos que contienen varias cadenas de información distintas, suelen ser poco comunes.

Un ejemplo de respuesta HTTP se puede encontrar en la figura 2.30.

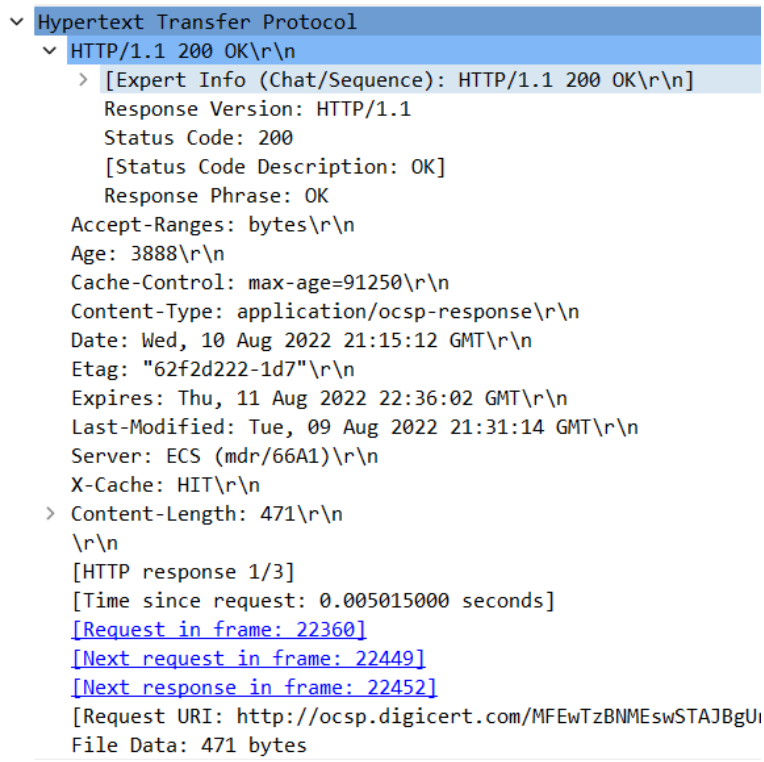


Figura 2.30: Mensaje de respuesta HTTP capturado en Wireshark

2.4.6 Desarrollo Web

El desarrollo de una aplicación web se compone de dos partes: el *FrontEnd* y el *BackEnd*.

- **FrontEnd**. Es el desarrollo de la interfaz visual, es decir, de la parte web que ven los usuarios. Existen diferentes lenguajes de programación para desarrollar el FrontEnd, los más habituales son:
  - **HTML5**.

    HTML5 es un lenguaje de marcado para la creación de páginas web y aplicaciones en la era de los dispositivos móviles, computación en la nube y redes sociales. Además este lenguaje se diferencia de los demás porque no está compuesto por instrucciones o comandos, sino por un conjunto de etiquetas que organizan y declaran el contenido del documento [44].

    Algunos elementos son simples, compuestos por una sola etiqueta, pero otros están compuestos por dos etiquetas, la primera de apertura y la segunda de cierre. Estas dos se distinguen por una barra inclinada que se escribe antes de la palabra clave

en la etiqueta de cierre. Por ejemplo, <html>... </html>.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Mi primer documento HTML</title>
    </head>
    <body>
        Hola Mundo
    </body>
</html>
```

Como se puede observar en el anterior script, un documento simple, ya de por sí, contiene una estructura relativamente compleja. Es la estructura que siempre ha de contener un documento HTML. La primera línea muestra la definición del documento seguida por la etiqueta de apertura <html>. Entre las etiquetas <html> se insertan otras etiquetas, como:

* **<head>**.En esta etiqueta se define el título de la página web, además se puede declarar la codificación de los caracteres usada por el documento, se puede dar información general acerca de la página web y también se pueden incorporar archivos externos necesarios para agregar estilos (CSS), código (JavaScript) o incluso imágenes [44].
* **<body>**. Esta etiqueta es otra sección muy importante que forma parte de la organización principal del documento. El cuerpo es la parte visible del documento.

– **JavasScript**.

JavaScript es un lenguaje de secuencias de comandos, muy similar a cualquier otro lenguaje de programación como C++, Python, Java, etc. Mientras que otros lenguajes se basan en el paradigma de la programación orientada a objetos, JavaScript es un lenguaje basado en prototipos [44]. Lo cierto es que, aunque parezca contradictorio, se relaciona más con objetos que cualquier otro.

JavaScript puede realizar innumerables tareas, desde proveer instrucciones simples hasta calcular complejos algoritmos, pero su característica más importante es su capacidad de almacenar y procesar información que se obtiene a partir de a variables.

– **CSS**.

Cascading Style Sheets (CSS) es un complemento desarrollado para superar las limitaciones y reducir la complejidad de HTML [44]. Inicialmente, se usaban atributos dentro de las etiquetas HTML para asignar algunos estilos básicos a cada elemento. Sin embargo, a medida que el lenguaje de programación iba evolucionando, el código era cada vez más complejo y HTML no pudo satisfacer las demandas de los diseñadores web. Como resultado, CSS fue implementado en HTML5 para separar la estructura de su presentación y otorgar estilos visuales a los elementos del documento, como tamaño, color, fondos, bordes, etc.

Las propiedades son la parte más importante de CSS. Existen docenas de propiedades y en cada versión se van añadiendo nuevas. Las más comunes son:

* **font**. Permite declarar varios estilos para el mismo texto, como grosor, tamaño,tipo de letra, etc.
* **color**. Declara el color de un elemento.
* **background**. Permite aplicar varios estilos al fondo de un elemento.
* **width**. Declara el ancho de un elemento.
* **height**.Declara el alto de un elemento
* **margin**.Declara el margen externo de un elemento.
* **padding**. Declara el margen interno de un elemento.
* **border**. Declara el ancho, color, estilo del borde de un elemento.
* **text-align**. Declara la alineación de un elemento. Centro, izquierda, derecha.

- **BackEnd**. Se desarrolla en el lado del servidor. Es la parte web que no es accesible para los usuarios porque se encarga de la utilización de los datos para que llegue al usuario [45].

  Python, Node.js, PHP, C# son lenguajes de alto de nivel para el desarrollo del BackEnd. Estos lenguajes tienen diferentes frameworks, que son herramientas con el propósito de facilitar el trabajo del BackEnd. Algunos de estos son: Django, Flask, ASP.NET, etc.

  – **Flask**.

    Flask es un *micro-framework* escrito en Python para facilitar la creación de aplicaciones web. *"Micro"* no significa que haga menos cosas que otros frameworks sino que ofrece un núcleo sólido con los sevicios básicos, mientras que extensiones de paquetes externos proveen el resto de servicios [46]. De esta manera sólo es necesario instalar los servicios que se vayan a utilizar.

    Flask tiene tres principales dependencias: Web Server Gateway Interface (WGSI), que describe cómo se comunica un servidor web con una aplicación web; plantillas, que facilitan el desarrollo de páginas webs, y la integración de una interfaz de línea de comandos [46].

    A continuación se va a exlicar la estructura básica de Flask a través de un pequeño ejemplo:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hola_mundo():
    return "<p>Hola mundo!</p>"

@app.route("/adios")
def adios_mundo():
    return "<p>Adiós mundo!</p>"
```

Todos los programas en los que se vaya a usar Flask siguen la siguiente estructura:

1. Importación de la clase Flask de la instancia WGSI.
2. Crear una instancia. Se crea una instancia llamada app aunque se puede poner otro nombre, lo importante es la obligación de pasar como primer argumento la palabra reservada ___*name*___. Es necesario para que Flask conozca dónde encontar las plantillas de la aplicación o ficheros estáticos [47].
3. Rutas y métodos. Los métodos son las funciones que se quieren implementar cuando se accede a la URL del método. Flask se encarga de hacer transparente al usuario la ejecución de la función a partir de la ruta. El elemento *route* de *app* es el encargado de decirle a Flask qué URL debe ejecutar su correspondiente función [47].

– **Django**

Django es un framework que permite crear aplicaciones web seguras de cualquier complejidad en unas pocas líneas y en unos tiempos muy razonables [48]. Fue diseñado para encargarse del manejo de los controladores, por eso se dice que trabaja en un ambiente *modelo*, *vista* y *controlador*, motivo por el cual es reutilizable y de desarrollo rápido.

2.5 Herramientas

En esta sección se explican las herramientas usadas en este proyecto.

2.5.1 BRITE

Para la generación de topologías de red aleatorias hemos utilizado la herramienta BRITE desarrollada por Alberto Medina, Anukool Lakhina, Ibrahim Matta, John Byers, Department of Computer Science, Boston University [49].

Es un generador de topologías flexible que no está limitado a una única forma de generar topologías, es más, soporta múltiples modelos de generación, ver figura 2.31:

- **Modelos a nivel de router**

  BRITE contiene la clase RouterModel derivada de la clase Model. La idea de esta clase es generar topologías a nivel de router, separadas de modelos para otros entornos como Autonomous System (AS), LANs, etc.

  Los modelos de topologías a nivel de router que proporciona BRITE son *Router Waxman* y *Router Barabasi-Albert*:

  – **Router Waxman**

  Hace referencia a un modelo de generación de topologías aleatorias que usa el

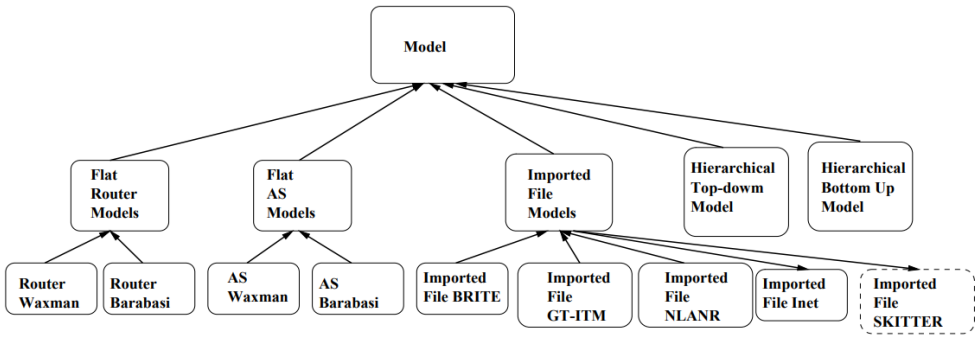


Figura 2.31: Modelos de generación de topologías [49]

modelo de probabilidad *Waxman* para la interconexión de los nodos de la topología [49], la cual es:

$$P(u, v) = \alpha e^{-d/(\beta L)} \tag{2.2}$$

Donde $0 < \alpha$, $\beta \leq 1$, *"d"* es la distancia Euclídea desde el nodo u al nodo v y *"L"* es la máxima distancia entre dos nodos cualesquiera. Un aumento de α produce un aumento en la densidad de enlaces, mientras que un aumento en β aumenta el ratio de enlaces más largos sobre los enlaces más pequeños. Ante la inviabilidad de modelar las topologías con cuatro parámetros (nºnodos, grado, α y β) por la sobrecarga de pruebas, se decide fijar los valores α y *beta*, siguiendo el modelado de Zegura [50], en donde propone utilizar $\alpha = 0.2$ y $\beta = 0.15$ para modelar redes de Internet.

– **Router Barabasi-Albert**

El modelo Barabasi-Albert es un algoritmo para generar redes aleatorias complejas libres de escala. Libres de escala quiere decir que el algoritmo sigue leyes exponenciales. Este modelo se caracteriza por dos conceptos: *Growth* (crecimiento), que el número de nodos en la red se incrementa con el tiempo, y *Preferencial Attachment* (Conexión preferencial), que a mayor conectividad de los nodos, mayor probabilidad tienen de recibir nuevos enlaces [49].

La red comienza con un conjunto de N_0 nodos conectados. Los nuevos nodos son añadidos a la red de uno en uno. Cada nuevo nodo añadido a la red se conecta a $N \leq N_0$ nodos existentes con una probabilidad que es proporcional al número de enlaces existentes. La probabilidad de conectar un nuevo nodo con el nodo i es:

$$P_i = \frac{k_i}{\sum_j k_j} \tag{2.3}$$

donde *ki* es el grado del nodo objetivo i y $\sum_j k_j$ es el sumatorio de los grados de salida de todos los nodos que se han unido previamente a la topología.

- **Modelos a nivel de AS**

  Los modelos de nivel AS son muy similares a las topologías a nivel de router, con la diferencia de que en los modelos AS, los nodos AS posicionados en el plano tienen la capacidad de contener topologías asociadas.

- **Topologías Jerárquicas**

  BRITE soporta la generación de dos topologías jerárquicas:

  – **Topología jerárquica Top-down**

  Para esta topología BRITE genera primero una topología de nivel AS entre las posibles: *Waxman*, *Barabasi*, *importada*, etc. Posteriormente BRITE generará topologías a nivel de router usando los distintos modelos de generación posibles. BRITE interconectará estas topologías a nivel de router según la conectividad de la topología AS. BRITE proporciona cuatro mecanismos de interconexión basándose en el generador de topologias GT-ITM [51]. Si se considera *(i, j)* un enlace a nivel , entonces escoge un nodo u de la topología a nivel de router, asociada al nodo AS *i*, *RT(i)*, y un nodo v de la topología a nivel de router, asociada al nodo AS *j*, RT(j):

  * **Aleatorio**. EL nodo u es seleccionado de forma aleatoria de *RT(i)* y de *RT(j)*.
  * **Por menor grado**. Los nodos u y v son los nodos con menor grado en *RT(i)* y *RT(j)*.
  * **Por menor grado sin ser nodos hoja**. Los nodos u y v son los nodos con el menor grado en *RT(i)* y *RT(j)* sin ser nodos hijo.
  * **Con grado mayor que *"K"***. Los nodos u y v son nodos con grado mayor o igual a *"K"* en *RT(i)* y *RT(j)*.

  La topología final es una topología a nivel de router formada por las topologías individuales de cada nodo AS.

  – **Topolgía jerárquica Bottom-up**

  Para esta topología, BRITE primero genera una topología a nivel de router. Posteriormente asigna a cada nodo AS un número de routers según el tipo de asignación seleccionada por el usuario. Estos tipos de asignación son:

  * **Constante**. Asigna a cada AS un número igual de nodos .
  * **Uniforme**. Asigna un numero uniformemente distribuido en el rango [1, NumNodes].
  * **Exponencial**. Asigna un valor exponencialmente distribuido con valor *NumNodes/NumAS*.
  * **Heavy-tailed**. Asigna un valor de una distribución por densidades en el rango [1, NumNodes].

  Una vez definido el número de routers por nodo AS, BRITE selecciona los nodos que pertenecen a cada nodo AS mediante dos mecanismos:

  - **Selección aleatoria**. Selecciona un nodo aleatorio para asignarlo al nodo *AS i*, hasta que dicho nodo alcanza el número de routers que tiene asignados. Se repite para cada nodo AS.
  - **Paseo aleatorio**. Realiza un paseo aleatorio a través del grafo, en cada paso se escoge un nodo vecino aleatorio. Cada nodo visitado es asignado al *AS i* hasta que este se llena. Se repite para cada nodo .

- **Topologías importadas**

  BRITE también permite la generación de topologías a partir de ficheros importados de otros generadores de topologías. Por ejemplo, podríamos generar una topología top-down a partir de ficheros de otros generadores como GT-ITM. permite importar las siguientes topologías: BRITE, GT-ITM, NLANR y Inet.

2.5.2 QEMU

QEMU [52] es un software libre con el objetivo de emular y virtualizar mediante dos modos. El primero, provee una máquina virtual con su correspondiente CPU, memoria, disco y dispositivos emulados para ejecutar un sistema operativo. Este modo es totalmente emulado, sin embargo, también se puede correr dentro de un hipervisor. Dependiendo de la arquitectura del ordenador se podrá usar un hipervisor u otro. El segundo modo, QEMU puede lanzar procesos compilados para una u otra CPU. En este modo la CPU siempre es emulada, no tiene opción de usar un hipervisor [53].

La diferencia entre QEMU y otros hipervisores como VMware o VirtualBox es la característica de emulación. VMware o VirtualBox solamente pueden virtualizar, pero QEMU es capaz de emular, lo que permite ejecutar sistemas operativos o incluso programas para una determinada arquitectura diferente a la de la computadora.

Según la arquitectura de la computadora soporta un hipervisor u otro. Se resume en la tabla 2.1.

QEMU ofrece una gran variedad de configuraciones, ya que provee una gran cantidad de comandos, como *qemu-img disk* que permite crear, convertir y modificar imágenes.

QEMU al ser una herramienta tan amplia en cuestión a funcionalidades, me voy a centrar en explicar los dispositivos de red soportados por QEMU, ya que el primer objetivo de este proyecto es conectar unos nodos con otros a través de máquinas virtuales generadas en QEMU.

La red en QEMU se divide en dos partes: el dispositivo de red virtual conectado a la máquina virtual y el *BackEnd* de la red, que interactúa con el dispostivo de red virtual emulado.

Para crear un *backend* se utiliza el siguiente comando:

```
netdev TYPE,id=NAME,...
```

Hay diferentes tipos de *BackEnds* para seleccionar según el entorno que se quiera desplegar:

Arquitectura	Hipervisores
ARM	kvm (64 bit only), tcg, xen
MPS	Kvm, tcg
PPC	kvm,tcg
RISC-V	Tcg
S390X	kvm, tcg
SPARC	tcg
x86	hax, hvf (64 bit only), kvm, nvmm, tcg, whpx (64 bit only), xen

Tabla 2.1: Correspondecia Arquitectura-Hipervisores [53]

- **QEMU User Networking**. La red de usuario de QEMU está implementada por *"slrip"* que provee la pila *TCP/IP* en QEMU y la usa para implementar una red virtual de tipo NAT [54].

  Es la red predeterminada de QEMU, generalmente sencilla de usar y no requiere de privilegios de administrador, sin embargo, sostiene una serie de limitaciones como la inviablidad de procesar el tráfico . Esto supone que el uso del comando *ping* no es útil para comprobar la conexión entre máquinas virtuales. Otra de las limitaciones es que la máquina virtual no puede ser accedida desde la red externa [54]. Estas limitaciones generan una red pobre con poco rendimiento.

  El típico ejemplo de la red de usuario predeterminada que se establece en QEMU se puede observar en la figura 2.32

- **Tap**. Una red de tipo *Tap* necesita el uso del dispostivo de red *tap* en el host. Ofrece muy buen rendimiento y puede configurarse para crear cualquier tipo de topología de red. El problema es que requiere una configuración de la red del host que tiende a ser diferente dependiendo de su sistema operativo. Para su ejecución necesita privilegios de administrador [54].

  El argumento para generar conexión de red a través de un dispostivo *tap* es el siguiente:

```
netdev tap,id=mynet0
```

- **VDE**. La red de tipo *VDE* usa una estructura de red denominada como indica su propio nombre: Virtual Distributed Ethernet (VDE) [54]. La idea es crear interruptores virtuales, que son básicamente sockets, y conectar las máquinas virtuales con el host . Además, es mucho más potente, ya que puede conectar bridges virtuales juntos, ejecutarlos en diferentes hosts y supervisar el tráfico de la red [55].
- **Socket**. La red usando sockets permite crear una red de máquinas virtuales para que se vean entre ellas, es decir, para que se comuniquen [54].

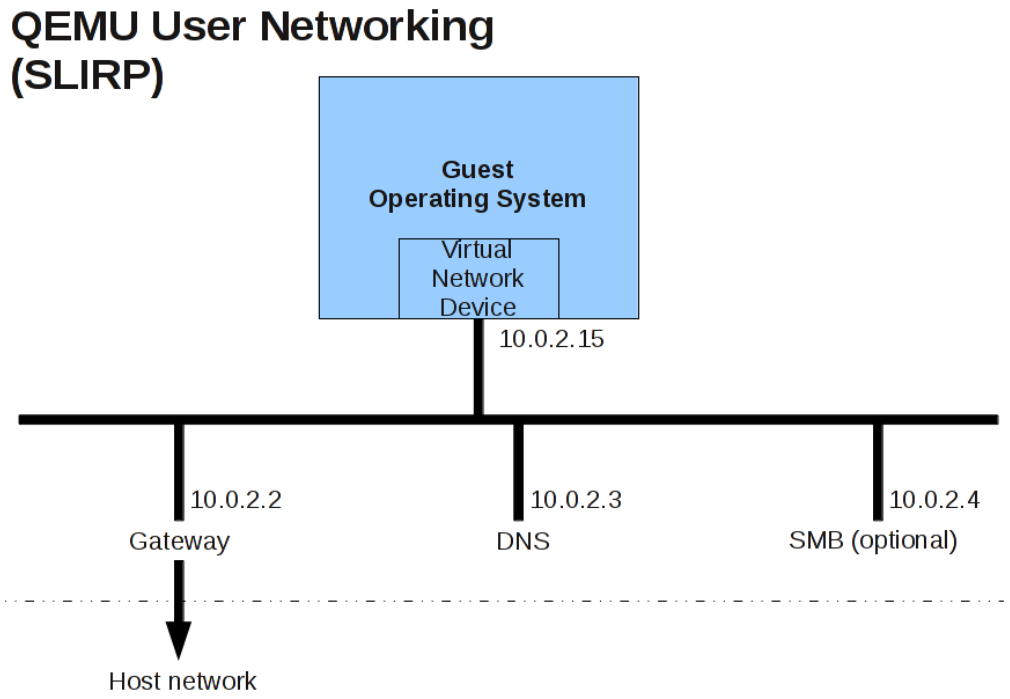


Figura 2.32: QEMU User Networking [54]

El argumento para ejecutar una red mediante sockets es el siguiente:

```
netdev socket,id=mynet0,listen=:1234
netdev socket,id=mynet0,connect=:1234
```

2.5.3 Dnsmasq

En distribuciones GNU/Linux existen diversas herramientas para instalar y configurar un servidor DHCP como ***ISC DHCP***, ***DHCP3 Server*** o ***Dnsmasq***.

ISC DHCP ofrece una solución completa par la implementación de servidores, agentes y clientes DHCP. Además soporta ambas versiones IP, IPv4 y IPv6 y es apropiado para un volumen alto de aplicaciones. *"Esta herramienta es más nueva y sustituye a DHCP3"* [56].

Dnsmasq acepta búsquedas DNS y las responde desde una pequeña caché local, o las reenvía hacia un servidor DNS recursivo. Además, Carga el contenido de /etc/hosts, de tal forma que los DNS locales, los cuales no aparecen en el DNS mundial, puedan ser resueltos [57].

El servidor DHCP de dnsmasq incluye soporte para DHCPv4, DHCPv6, BOOTP y PXE. Admite tanto direcciones estáticas como dinámicas [57]. Al configurar una direción IP, esta se asocia a la dirección MAC del dispositivo en el archivo */var/lib/misc/dnsmasq.leases*.

2.5.4 Graphviz

Graphviz es un software de código abierto para visualizar todo tipo de grafos. La visualización de un grafo es una manera de representar la información de manera estructural en forma de diagrama [58]. Es importante para aplicaciones de red, así el usuario es capaz de visualizar una topología de red y no tener que imaginársela a través de unos datos escritos en un archivo de texto. Esta herramienta no sólamente es capaz de visualizar grafos de red sino que, como se ha comentado previamente, pinta todo tipo de gráficos, así que, también puede ser útil para realizar diagramas de flujos, esquemas para el estudio de cierta materia, etc.

Para visualizar el grafo, la herramienta necesita una configuración de entrada, ya sea por medio de un archivo o en la misma línea de comandos. Normalmente se usa un fichero.

En la figura 2.33 se muestra un ejemplo de archivo de entrada de *Graphviz*. Se indica la forma del nodo y la interconexión de nodo. El archivo de salida es una imagen visualizando la información contenida en el fichero de entrada. La salida de este ejemplo se observa en la figura 2.34.

```
graph brite_topology {
        node [shape = circle];

        0 -- 1;
        1 -- 3;
        1 -- 4;
        0 -- 2;
        2 -- 5;
        2 -- 6;
}
```

Figura 2.33: Archivo de entrada de Graphviz

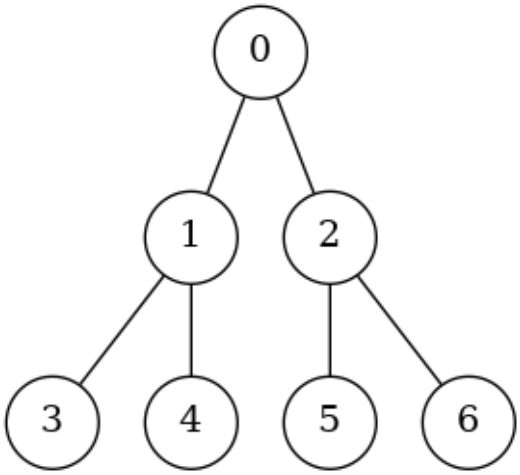


Figura 2.34: Topología graficada

3 Desarrollo e implementación

El objetivo de este capítulo es explicar al lector el desarrollo realizado durante el proyecto para llevar a cabo el objetivo general del mismo, el cual es el desarrollo de una herramienta para desplegar redes virtuales heterogéneas de manera automática

Dicha herramienta se puede dividir en dos partes muy diferenciadas: la primera parte, es el propio despliegue de red, es decir, la distribución de los componentes de la red y su cooperación para lograr la conectividad entre los nodos. La segunda parte, consiste en la realización de una página web con el fin de facilitar al usuario la selección de la cantidad de nodos, la arquitectura y el despliegue de la red.

A continuación se deja el enlace de github donde se encuentran todos los programas desarrollados de la herramienta: https://github.com/NETSERV-UAH/DEVE

3.1 Obtención de los ficheros de salida de BRITE

A lo largo de esta memoria, de manera repetitiva, se va a mencionar que la topología a desplegar surge de la información del archivo de salida que proporciona esta herramienta.

El grupo de investigación *Redes y Sistemas Inteligentes - Networks and Intelligent Systems* de la UAH desarrolló varias *scripts* de automatización para generar los archivos de salida de BRITE, en estos archivos se encuentra la información del grafo que describe la topología de la red que se desea desplegar. Dichas *scripts* se pueden encontrar en el siguiente repositorio de github: "https://github.com/NETSERV-UAH/BRITE".

En la figura 3.1 se muestra la estructura de un archivo de salida de BRITE. Dicho fichero se divide en tres partes: un encabezado; la información de los nodos y, por último, la información sobre los enlaces.

```
Topology: ( 4 Nodes, 4 Edges )
Model ( 1 ): 4 1000 100 1 1 1 0.2 0.15 1 10 1024

Nodes: (4)
0 0.00 0.00 2 2 -1 RT_NODE
1 41.00 176.00 2 2 -1 RT_NODE
2 364.00 91.00 2 2 -1 RT_NODE
3 92.00 487.00 2 2 -1 RT_NODE

Edges: (4):
0 1 0 180.71 0.60 10.00 -1 -1 E_RT U
1 2 1 334.00 1.11 10.00 -1 -1 E_RT U
2 3 2 480.42 1.60 10.00 -1 -1 E_RT U
3 0 3 495.61 1.65 10.00 -1 -1 E_RT U
```

Figura 3.1: Archivo de salida de BRITE

(a) Nodos.txt **(b)** Enlaces.txt

Figura 3.2: Archivo de salida de BRITE dividido en dos archivos

Finalmente y, para facilitar el uso de la información topológica, el archivo orignal se divide en dos nuevos ficheros: un fichero de nodos, denominado *Nodos.txt*, que proporciona la información de los nodos, y un fichero de enlaces, denominado *Enlaces.txt*, que proporciona la interconexión de los nodos. En la figura 3.2 se muestra el contenido de ambos ficheros generados.

3.2 Diseño de la red virtual

El diseño de red a desplegar viene definido por la información topológica obtenida a partir del fichero descritos anteriormente (fichero de nodos y fichero de enlaces). Cabe restaltar el concepto de *nodo* que este proyecto va a utilizar. Para este proyecto un *nodo* representado en BRITE es un punto en un plano que se conecta con otros puntos. Este puede ser cualquier dispositivo que se encuentre dentro de una red, como un ordenador, un router, un switch, etc. En este despliegue de red, el nodo se conforma por un dispositivo *bridge* y una MV tal como se muestra en la figura 3.3.

El despliegue de la red sigue tres fases en el siguiente orden secuencial: primero, el despliegue y configuración de los dispostivos e interfaces de la red (*bridge*, *tap*, *servidores DHCP*, *veth*); segundo, el arranque de las MV y, por último, el establecimiento de los enlaces entre los dispositivos *bridge* a través de las interfaces *veth*.

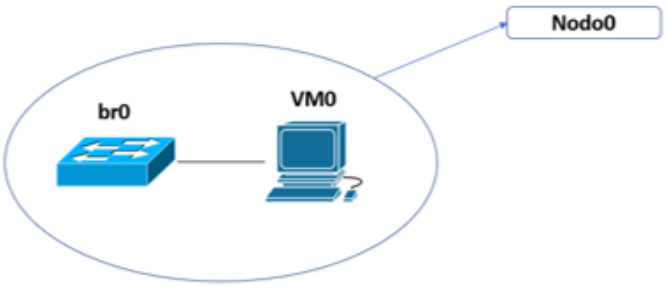


Figura 3.3: Equivalencia nodo

Para un mejor entendimiento por parte del lector, se va a explicar el despligue de la red mediante dos maneras diferentes, en primer lugar, un despliegue manual de las tres fases y, en segundo lugar, la automatización de la tres fases.

3.3 Despliegue manual

Esta sección consiste en el despliegue de una topología de red de manera manual. Se toma como ejemplo la red mostrada en la figura 3.4, que se trata de una topología *Barabasi* de grado dos con cuatro nodos: el *nodo0* se conecta con el *nodo1*, el *nodo2* con el *nodo0* y el *nodo3* con el *nodo2*.

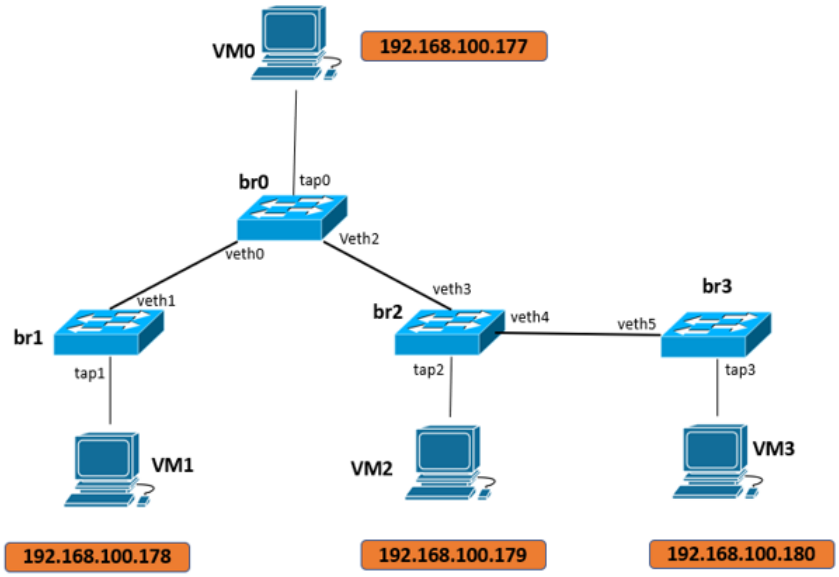


Figura 3.4: Ejemplo Topología Barabasi 4 nodos grado 2

3.3.1 Fase I. Despliegue de red

El despliegue de red, se lleva a cabo mediante la creación de los dispositivos *bridge*, la configuración de una dirección IP en cada *bridge*, la creación de las interfaces *tap*, el encendido de los *bridges* y *taps*, la configuración de servidor de DHCP para cada *bridge* y la creación de las *interfaces veth*.

Para desplegar la topología de la figura 3.4 se realizan los siguientes pasos:

- **Creación de los dispositivos *bridge***

  Se han de crear 4 dispositivos *bridge* tal como se muestra en la figura 3.4: *br0*, *br1*, *br2* y *br3*.

```
sudo ip link add br0 type bridge
sudo ip link add br1 type bridge
sudo ip link add br2 type bridge
sudo ip link add br3 type bridge
```

- **Asginación de una dirección IP a cada *bridge***

  Las direcciones IP asigandas a los *bridges* comienzan desde la dirección *192.168.100.2* y continúan por la dirección *192.168.100.3* y así sucesivamente. La asignación de las direcciones IP se otorgan en función de la ID del *bridge*: *br(ID)*, a menor ID, menor dirección IP.

  La dirección IP *192.168.100.2* se asignará a *br0*, la dirección *192.168.100.3* a *br1*, la dirección *192.168.100.4* a *br2* y la dirección*192.168.100.5* a *br3*.

```
sudo ip addr add 192.168.100.2/24 brd 192.168.100.255 dev br0
sudo ip addr add 192.168.100.3/24 brd 192.168.100.255 dev br1
sudo ip addr add 192.168.100.4/24 brd 192.168.100.255 dev br2
sudo ip addr add 192.168.100.5/24 brd 192.168.100.255 dev br3
```

- **Creación de las interfaces *tap***

  Se ha de crear una interfaz tap para cada MV, por tanto, se ha de añadir *tap0*, *tap1*, *tap2* y *tap3*.

```
sudo ip tuntap add name tap0 mode tap
sudo ip tuntap add name tap1 mode tap
sudo ip tuntap add name tap2 mode tap
sudo ip tuntap add name tap3 mode tap
```

- **Conexión de la interfaz *tap* con su *bridge* correspondiente**

  El enlace de los *bridges* con las interfaces *tap* siempre siguen el mismo patrón, el ID. Por ejemplo *(br0-tap0)*, *(br1-tap1)*. El enlace que no cumpla esta regla es erróneo, como por ejemplo, *(br0-tap1)*.

```
sudo ip link set tap0 master br0
sudo ip link set tap1 master br1
sudo ip link set tap2 master br2
sudo ip link set tap3 master br3
```

- **Activar dispostivos *bridge***

  Se han de activar los cuatro dispositivos *bridge* creados: *br0*, *br1*, *br2* y *br3*.

```
sudo ip link set br0 up
sudo ip link set br1 up
sudo ip link set br2 up
sudo ip link set br3 up
```

- **Activación de las interfaces *tap***

Se han de activar las cuatro interfaces *tap* creadas: *tap0*, *tap1*, *tap2*, *tap3* y *tap4*.

```
sudo ip link set tap0 up
sudo ip link set tap1 up
sudo ip link set tap2 up
sudo ip link set tap3 up
```

- **Configuración del servidor DHCP en cada *bridge***

  Cada *bridge* ha de ser configurado con un sevidor DHCP, ya que cada uno debe proporcionar una dirección IP a la MV enlazada directamente. Por este motivo, solo ha de configurarse una dirección IP en cada servidor DHCP.

  Las direcciones IP a configurar aparecen en la figura 3.4.

```
sudo dnsmasq --interface br0 -p 0 --bind-interfaces --dhcp-range↩
    ↪ =192.168.100.177,192.168.100.177

sudo dnsmasq --interface br1 -p 0 --bind-interfaces --dhcp-range↩
    ↪ =192.168.100.178,192.168.100.178

sudo dnsmasq --interface br2 -p 0 --bind-interfaces --dhcp-range↩
    ↪ =192.168.100.179,192.168.100.179

sudo dnsmasq --interface br3 -p 0 --bind-interfaces --dhcp-range↩
    ↪ =192.168.100.180,192.168.100.180
```

- **Creación de las interfaces *veth***

  Se han de generar tres enlaces, por lo que se han de crear seis interfaces *veths*: *(veth0-veth1)*, *(veth2-veth3)* y *(veth4-veth5)*.

```
sudo ip link add dev veth0 type veth peer name veth1
sudo ip link add dev veth2 type veth peer name veth3
sudo ip link add dev veth4 type veth peer name veth5
```

3.3.2 Fase II. Despliegue de las máquina virtuales

Los nodos van a desplegarse mediante máquinas virtuales utlizando la plataforma *QEMU*. Para su configuración, se han seleccionado tres arquitecturas distintas: *arm1176*, *i386* y *x86_64* y un sistema operativo para cada arquitectura.

Con la arquitectura *ARM* se quiere emular una *Raspberry Pi*, por tanto, el sistema operativo debe estar basado en *Raspbian*, no necesariamente la versión oficial, sino una versión compatible con *arm1176*. Para la arquitectura *Intel* de 32 bits se escoge el sistema operativo *Q4OS* y para Intel de 64 bits, *Alpine*.

3.3.2.1 Emulación de una *Raspberry Pi* con *QEMU*

En la última versión de *QEMU* es posible emular los dispositivos embebidos *Raspberry Pi0, Pi1, Pi2* y *Pi3*, pero no tienen dispositivos de red para poder comunicarse con otras máquinas. Como no es viable esta opción, se selecciona otra máquina proporcionada por QEMU: *"versatilepb"*.

Para poder emular una *Raspberry Pi* en dicha máquina se necesitan los siguientes archivos: *"kernel-qemu-5.4.51-buster"* y *"versatile-pb-buster-5.4.51.dtb"* que se encuentran en el siguiente repositorio de github: *"https://github.com/dhruvvyas90/qemu-rpi-kernel"*. Además de estos archivos, también se necesita disponer de la imagen del sistema operativo de la *Raspberry Pi*: *2020-02-13-raspbian-buster-lite.img*.

Una vez descargados los archivos, se desarrolla la siguiente *script* para emular una *Raspberry Pi* en *QEMU*:

Código 3.1: rpi.sh

```
#!/bin/bash
qemu-system-arm\
    -M versatilepb\
    -cpu arm1176\
    -m 256\
    -drive "file=image/2020-02-13-raspbian-buster-lite.img,if=none,index=0,media=disk↩
        ↪ ,format=raw,id=disk0"\
    -device "virtio-blk-pci,drive=disk0,disable-modern=on,disable-legacy=off"\
    -dtb dtb/versatile-pb-buster-5.4.51.dtb\
    -kernel kernel/kernel-qemu-5.4.51-buster\
    -append "root=/dev/vda2 panic=1 rootfstype=ext4 rw"\
    -net nic,macaddr=00:16:3e:00:00:00\
    -net tap,ifname=tap0,script=no,downscript=no\
    -no-reboot
```

Las líneas de la *script* que empiezan por *"-net"* son para la configuración del dispositivo de red en *QEMU* como se explicó en el apartado 2.5.2.

Para ejecutar el código de manera correcta se ha tenido que configurar previamente la interfaz *tap* mediante el siguiente comando:

```
sudo ip tuntap add name tap0 mode tap
```

3.3.2.2 Emulación de la arquitectura i386 y x86_64 con *QEMU*

Para la configuración de una máquina i386 y x86_64 se necesita configurar un disco de almacenamiento y disponer de la imagen del sistema operativo.

Las imagénes para cada arquitectura son las siguientes:

- **Arquitectura i386:** *"q4os-4.8-i386-instcd.r2.iso"*
- **Arquitectura x86_64:** *"alpine-standard-3.15.4-x86_64.iso"*

Para crear el disco de almacenamiento en *QEMU* se escribe:

```
qemu-img create -f qcow2 alpine-disk1.img 10G

qemu-img create -f qcow2 q4os-disk1.img 10G
```

Para establecer el formato de imagen existen varios tipos, pero los más recomendables son *RAW* y *QCOW2*. En este caso se optó por *QCOW2*, formato exclusivo para *QEMU*. Una vez creado el disco y descargada la imagen, se escriben los dos siguientes códigos:

Código 3.2: q4os.sh

```
#!/bin/bash

sudo qemu-system-i386 \
    -name q4os \
    -m 256 \
    -cdrom images/q4os-4.8-i386-instcd.r2.iso \
    -drive file=disks/q4os-disk \
    -boot c\
    -enable-kvm \
    -netdev tap,id=mynet0,ifname=tap0,script=no,downscript=no \
    -device e1000,netdev=mynet0,mac=52:55:00:d1:55:01
```

Código 3.3: alpine.sh

```
#!/bin/bash

sudo qemu-system-x86_64 \
    -m 256 \
    -cdrom ../images/alpine-standard-3.15.4-x86_64.iso \
    -drive file=../disks/alpine-disk \
    -boot c\
    -enable-kvm \
    -netdev tap,id=mynet0,ifname=tap0,script=no,downscript=no \
    -device e1000,netdev=mynet0,mac=52:55:00:d1:55:01
```

Para poder ejecutarlo de manera correcta se ha tenido que configurar previamente el disposivo *tap* mediante el siguiente comando:

```
sudo ip tuntap add name tap0 mode tap
```

El primer arranque de ambas máquinas corresponde con la instalación del sistema operativo, mientras que los siguientes se inician con el sistema operativo ya configurado para su uso.

Cuando se inician las máquinas virtuales, estas deben de obtener una dirección IP estática del *bridge* asociado, dado que cada *bridge* también realiza la función de servidor DHCP. Según la figura 3.4, el servidor DHCP de *VM0* debería ser *br0*, que sería lo intuitivo, no obstante, existe la posibilidad de obtener una dirección IP de un *bridge* que no esté conectado directamente, por ejemplo, que *VM0* obtenga la dirección a través de *br1*.

Es necesario que cada *bridge* proporcione la dirección IP a la MV que está conectada directamente con él. El primer motivo es para que el nodo con ID más bajo (nodo0, nodo1, etc.), obtenga la dirección IP más baja, y el usuario final pueda conocer la dirección IP de cada MV de manera sencilla. Siempre empieza con una determinada dirección IP fija y se incrementa de uno en uno. El segundo motivo consiste en la dificultad de establecer la relación *MV-BRIDGE* y *NODO* en caso de que la dirección IP no sea suministrada por el *bridge* correspondiente de la MV. Al acceder a una MV se puede obtener la dirección IP, sin embargo, sin un análisis de paquetes, no se puede determinar qué *bridge* le ha adjudicado la dirección IP, por lo que tampoco se puede saber, desde la posición del *bridge*, a qué MV le ha asignado la dirección IP. Esto, puede dar lugar a una confusión de relación *MV-BRIDGE* y *NODO*. Por ejemplo, si *br0* y *VM0* están asociados al *nodo0*, pero la *VM0* tiene como dirección IP la proporcionada por *br1*, cuando debería ser la proporcionada por *br0*, se declarará que *br0* es del *nodo0*, pero *VM0* es del *nodo1* y la relación fallaría, ya que, tanto *VM0* como *br0* deberían de pertenecer al *nodo0*.

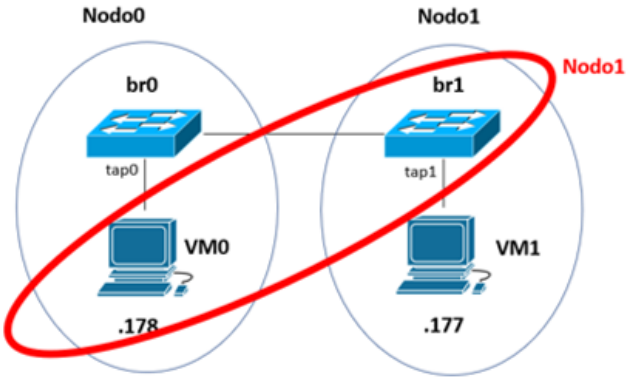


Figura 3.5: Asignación errónea MV-BRIDGE, NODO

En un supuesto caso de que se pudiese determinar a qué MV le asigna el *bridge* la dirección IP, se podría establecer la relación *MV-BRIDGE* y *NODO*, como aparece en la figura 3.5, lo único que habría que cambiar el nombre de las MV después de desplegar la topología, en la figura donde está escrito *VM0* debería de ser *VM1*, de tal manera que, ahora la asociación sería correcta: *VM1-br1* y *Nodo1*. Si en redes pequeñas ya es confuso cambiar el nombre a las MV para que la asociación sea la correcta, en redes grandes, el nivel de confusión es mucho mayor, y se puede llegar a cometer errores. Además, es poco intuitivo tener cruzada la conexión de los *bridges* con las máquinas virtuales para tener la relación *MV-BRDGE* y *NODO*.

3.3.3 Fase III. Interconexión de los nodos

La interconexión de los nodos consiste en generar la conexión de los enlaces para interconectan los *bridges* mediante las *interfaces veth* creadas en el apartado 3.3.1. Según muestra

la figura 3.4 se debe de enlazar *br0* con *br1*, *br0* con *br2* y *br2* con *br3*.

- Enlace *br0-br1*

  Para enlazar el *bridge br0* con *br1* se ha de enlazar la interfaz *veth0* con el *bridge* br0 y la interfaz *veth1* con br1.

```
sudo ip link set veth0 master br0
sudo ip link set veth1 master br1
```

- Enlace *br0-br2*

  Para enlazar el *bridge br0* con *br2* se ha de enlazar la interfaz veth2 con el *bridge* br0 y la interfaz *veth3* con br2.

```
sudo ip link set veth2 master br0
sudo ip link set veth3 master br2
```

- Enlace *br2-br3*

  Para enlazar el *bridge br2* con *br3* se ha de enlazar la interfaz veth4 con el *bridge* br2 y la interfaz *veth5* con br3.

```
sudo ip link set veth4 master br2
sudo ip link set veth5 master br3
```

Se recuerda que la interfaces veth se crean por pares, por esta razón, según cómo se crearon en el apartado 3.3.1 y cómo se enlazan con los *bridges* en este apartado, *br0* se conecta con *br1*, *br0* con *br2* y *br2* con *br3*.

Por último, se activan las interfaces veths:

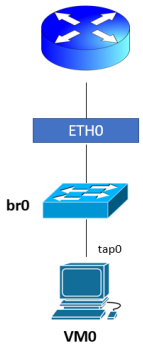
```
sudo ip link set up veth0
sudo ip link set up veth1
sudo ip link set up veth2
sudo ip link set up veth3
sudo ip link set up veth4
sudo ip link set up veth5
```

3.3.4 Conexión a Internet. Conexión de red en modo puente y en modo NAT

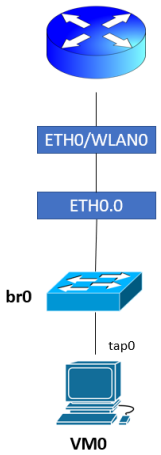
Para mejorar el despliegue de la red se quiere añadir conexión a Internet a las máquinas virtuales.

Las máquinas virtuales no cuentan con acceso directo a Internet, sin embargo, se puede lograr a través de la tarjeta de red física del host que está conectado al router vía wifi o ethernet.

Se establecen dos tipos de red para conseguir el acceso a Internet: el modo puente y el modo NAT. Con el modo puente se obtiene la dirección IP de la red LAN del router, en



(a) Conexión modo puente



(b) Conexión modo NAT

Figura 3.6: Conexión a Internet

cambio, con la configuración en modo NAT, se configura una red diferente y la tarjeta de red física del host ha de actuar como enrutador entre la red LAN y la red creada.

Para configurar una máquina virtual en modo puente, como se observa en la figura 3.6a se han de realizar los siguientes pasos:

- Añadir un dispositvo bridge

```
sudo ip link add br0 type bridge
```

- Añadir un dispositivo tap

```
sudo ip tuntap tap0 mode tap
```

- Enlazar interfaz tap al bridge

```
sudo ip link set tap0 master br0
```

- Enlazar interfaz física al bridge

```
sudo ip link set eth0 master br0
```

- Encender bridge y tap

```
sudo ip link set br0 up
sudo ip link set tap0 up
```

- Añadir dirección IP al bridge de la red LAN a través del router

```
sudo dhclient -v br0
```

La conexión de la red en modo puente se realiza exclusivamente a través de la interfaz ethernet, ya que la interfaz wifi no se puede enlazar con el *bridge*.

En la figura 3.6, el diseño del modo puente respecto al modo NAT es algo diferente. Este último no conecta el *bridge* directamente con la interfaz de red física, es decir, con *eth0*. Si se conectara, estaríamos ante una conexión en modo puente. Por este motivo, se crea una interfaz VLAN, dicho con otras palabras, una interfaz *ethernet virtual*.

En este modo, el router no proporcionará el direccionamiento IP a la máquina virtual, sino que, esta función la realizará el *bridge*. Por esto, el *bridge* tendrá dos funciones principales: proveer la dirección IP a la máquina virtual y hacer de enlace entre la interfaz ethernet virtual y la interfaz tap de la máquina, ya que esta no puede conectarse directamente a la interfaz ethernet virtual.

Por otro lado, la tarjeta de red física debe de actúar como enrutador NAT, por lo que es necesario activar el reenvío de paquetes y crear una tabla de traducción IP para que la máquina tenga acceso a la red externa.

Para configurar una máquina virtual en modo NAT, como se observa en la figura 3.6b, se han de realizar los siguientes pasos:

- Añadir un dispositivo bridge

```
sudo ip link add br0 type bridge
```

- Añadir dirección IP al bridge

```
sudo ip addr add 192.168.100.2/24 brd 192.168.100.255 dev br0
```

- Añadir un dispositivo tap

```
sudo ip tuntap tap0 mode tap
```

- Configurar bridge como servidor DHCP

```
sudo dnsmasq --interface br0 -p 0 --bind-interfaces --dhcp-range↩
    ↪ =192.168.100.6,192.168.100.6
```

- Añadir interfaz VLAN

```
sudo ip link add link wlp0s20f3 name eth0.0 type vlan id 100
```

- Añador dirección IP a la interfaz VLAN

```
sudo ip addr add 192.168.100.1/24 brd 192.168.100.255 dev eth0.0
```

- Enlazar interfaz tap al bridge

```
sudo ip link set tap0 master br0
```

- Enlazar interfaz VLAN al bridge

```
sudo ip link set eth0.0 master br0
```

- Encender interfaces

```
sudo ip link set br0 up
sudo ip link set tap0 up
sudo ip link set eth0.0 up
```

- Crear tabla NAT

```
sudo iptables -t nat -A POSTROUTING -o wlp0s20f3 -j MASQUERADE
```

- Añadir configuración de reenvío de paquetes en la interfaz física de red

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

En el caso de la configuración de red en modo puente, las máquinas virtuales obtienen todos los parámetros de red necesarios a través del router, en cambio, en la configuración en modo NAT no obtienen toda la configuración, ya que únicamente obtienen una dirección MAC y una dirección IP. Con estos dos parámetros, la maquina virtual puede llegar al router de la red, pero no navegar por Internet al no tener configurado el DNS. Por ello, se debe de configurar en cada máquina añadiendo el DNS de Google: 8.8.8.8 en el fichero "/etc/resolv.conf". Se puede realizar en el primer arranque de la MV cuando se está configurando el sistema operativo.

3.4 Despliegue automático

Desplegar una red manualmente es una tarea tediosa sobre todo si se quieren desplegar topologías de muchos nodos interconectados. Con el objetivo de facilitar el despliegue de la red se desarrollan una serie de programas para automatizar el despliegue de la red virtual.

3.4.1 Fase I. Automatización del despliegue de la red

Esta subssección se divide en tres partes para llevar a cabo todo el proceso para automatizar el despliegue y configuración de los dispositivos e interfaces de red. En primer lugar se desarrolla un despliegue básico, en segundo lugar se mejora el despliegue para evitar bucles, y por último, se añade al despliegue una configuración para conseguir que las máquinas virtuales tengan conexión a Internet.

3.4.1.1 Despliegue básico de red

La automatización básica de red implementa la creación y configuración de los dispostivos e interfaces de red. Para ellos se ha desarrollado la sigguiente script:

Código 3.4: net_v3.sh

```
#!/bin/bash

if [ "$#" -ne 2 ]; then
    echo "Incorrect number of arguments"
    echo "Type: $# <number of nodes> <number of edges> "
    exit 0
fi

N_NODES=$1
N_EDGES=$2

ID_IP_BR=2
ID_IP_MV=128
ID_VETH_A=0
ID_VETH_B=1

for ((i=0; i<${N_NODES}; i++ )); do

    #BRIDGE
    sudo ip link add dev br${i} type bridge
    sudo ip addr add 192.168.100.${ID_IP_BR}/24 brd 192.168.100.255 dev br${i}

    #TAP
    sudo ip tuntap add name tap${i} mode tap
    sudo ip link set tap${i} master br${i}

    #Links UP
    sudo ip link set dev br${i} up
    sudo ip link set dev tap${i} up

    #DHCP
    sudo dnsmasq --interface br${i} -p 0 --bind-interfaces --dhcp-range=192.168.100.${↩
        ↪ ID_IP_MV},192.168.100.${ID_IP_MV}

    ID_IP_BR=$((ID_IP_BR + 1))
    ID_IP_MV=$((ID_IP_MV + 1))
done
```

Primero se realiza una comprobación del número de nodos a introducir en la ejecución del programa. En segundo lugar, se despliega cada *bridge*, se asigna una dirección IP a cada *bridge*, se despliega cada interfaz *tap*, se genera el enlace entre el bridge y la interfaz *tap*, se encienden los bridges y las interfaces y por último se configura el servidor DHCP de cada *bridge*.

La red a desplegar se configura en la subred *192.168.100.0/24*. De las 256 direcciones IP posibles, se disponen para su uso 254 direcciones. La dirección IP *192.168.100.0* corresponde a la dirección de red y la dirección *192.168.100.255* corresponde a la dirección de Broadcast.

De las 254 direcciones disponibles, se reserva la dirección *192.168.100.1* para configurar una interfaz VLAN. Las direcciones restantes se usan para establecer las direcciones IP en los dispositivos bridge y en las máquinas virtuales.

La topología se compone del mismo número de máquinas virtuales que de bridges. Se reparten las 253 direcciones en partes iguales: 126 direcciones para los bridges, empezando desde la *.2* hasta la *.127* y otras 126 direcciones para las máquinas virtuales, desde la *.128* hasta la *.253*.

Además, la división de las direcciones indica el número máximo de nodos, sin contar con la limitación de memoria RAM del ordenador, aunque se ha establecido que el número máximo de máquinas virtuales a desplegar son diez.

3.4.1.2 Despliegue de red para evitar bucles

Cuando se diseña una topología de red, es necesario tener en cuenta los diferentes caminos para llegar de un nodo a otro. En capa 2, el problema aparece cuando un mismo bridge tiene dos rutas distintas para llegar al mismo punto, entonces al enviar mensajes de difusión se produce una tormenta de broadcast.

En el apartado 4.1.7 se realiza una prueba para comprobar el funcionamiento de la red cuando existen bucles de capa 2. El resultado es una tormenta de broadcast que se soluciona con la activación del procolo STP en los bridges, comprobado en el apartado 4.1.8.

Para activar el protocolo STP en los bridges se recurre al comado:

```
brctl stp <bridge device> on
```

De modo que para eliminar los bucles en las redes a desplegar, se añade el comando anterior a la automatización para cada bridge creado:

```
1  for ((i=0; i<${N_NODES}; i++ )); do
2      ...
3      sudo brctl stp br${i} on
4      ...
```

3.4.1.3 Despliegue de red para conexión a Internet

En esta automatización del despliegue de red se quiere incluir la conexión a Internet de las máquinas virtuales. Para ello se crea una red de tipo NAT desplegando una interfaz VLAN entre la tarjeta de red del ordenador y *br0* como se muestra en la figura 3.7.

La interfaz de red física del ordenador debe de encaminar los paquetes provenientes de *eth0.0* a *eth0* y *eth0* hacia el router físico. Para conseguir comunicación del exterior al interior y viceversa, es necesario crear una tabla NAT y habilitar el reenvío de paquetes de la tarjeta de red física como se explicó en el apartado 3.3.4. Para conseguirlo se añade el siguiente código:

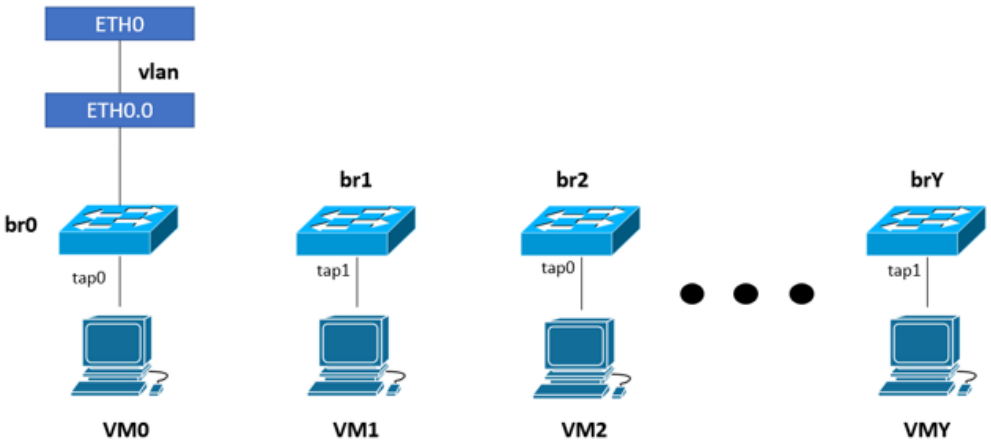


Figura 3.7: Conexión a Internet

```
#VLAN
sudo ip link add link wlp0s20f3 name eth0.0 type vlan id 100
sudo ip addr add 192.168.100.1/24 brd 192.168.100.255 dev eth0.0
sudo ip link set eth0.0 master br0
sudo ip link set dev eth0.0 up

#NAT
sudo iptables -t nat -A POSTROUTING -o wlp0s20f3 -j MASQUERADE
echo 1 |sudo tee /proc/sys/net/ipv4/ip_forward
```

3.4.2 Fase II. Automatización del despliegue de las máquinas virtuales mediante QEMU.

La segunda fase reside en el arranque de un determinado número de máquinas virtuales, en un principio solamente se van a desplegar *Raspberry Pis*.

ID del Nodo	Coordenada X	Coordenada Y

Tabla 3.1: Estructura de archivo de nodos

La imagen del sistema operativo de la *Raspberry Pi* también desempeña la función de disco de almacenamiento. QEMU no permite arrancar dos *Raspberry Pis* con el mismo disco en el mismo directorio, en cambio, si el disco es duplicado y se traslada a otra ubicación, QEMU ahora sí que admite el inicio de ambas máquinas.

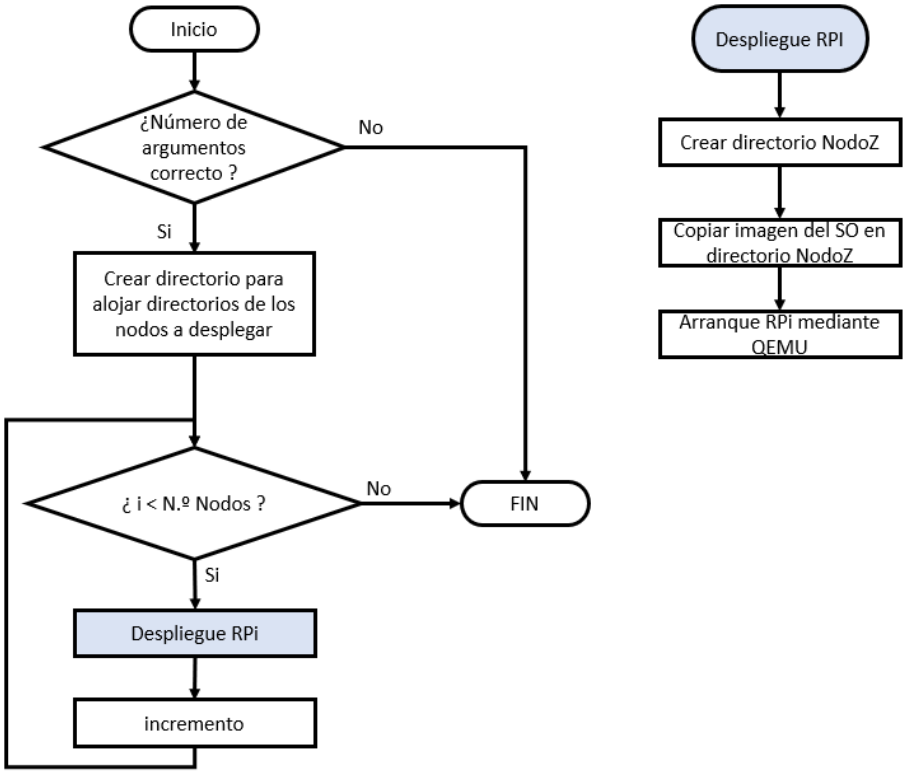


Figura 3.8: Diagrama automatización arranque máquinas virtuales

La automatización, mostrada en la figura 3.8, se basa en realizar tantas copias de la imagen del sistema operativo como nodos contenga la topología, y ubicarlas en diferentes directorios.

El argumento de entrada es el número de *Raspberry Pis* a desplegar, este número no lo va a introducir el usuario, sino que se va a leer del archivo de nodos cuya estructura es la de la tabla 3.1.

El número de nodos, que se introduce como argumento de entrada, se obtiene del recuento de líneas que contiene el archivo de nodos a través de la siguiente función:

```
1   def count_nodes():
2   nodes = 0
3
4   #Read Nodos.txt
5   node_file = open(PATH_TOPOS + FILE_NAME_NODES, 'r')
6   try:
7       for line in node_file:
8           nodes = nodes + 1
9   finally:
10      node_file.close()
11
12  return nodes
```

3.4.3 Fase III. Automatización de la interconexión de los nodos

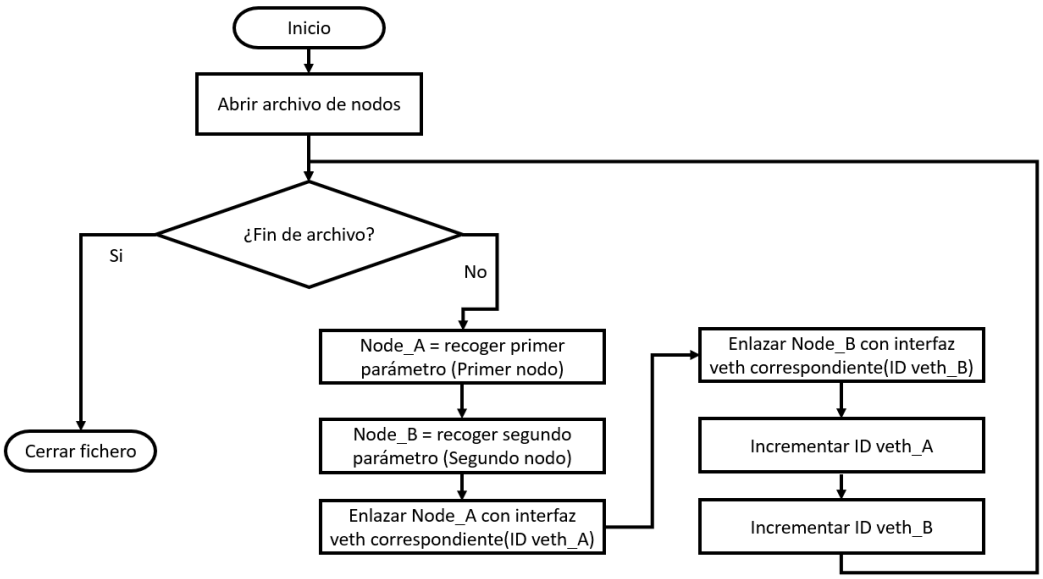


Figura 3.9: Diagrama conexión enlaces

Precedentemente, la comunicación entre los bridges se realiza de manera manual. El propio usuario elige los bridges a interconectar o los determina a través del archivo de enlaces, e introduce los comandos uno a uno para establecer la comunicación entre ellos.

Si cada vez que se despliega una red diferente, hay que teclear el comando para enlazar cada bridge, se convierte en una tarea tediosa y, sobre todo, cuando el número de enlaces es elevado. Para facilitar la tarea, se automatiza este proceso.

La lectura del archivo de enlaces y el establecimiento de ellos no lo realizará el usuario, sino la siguiente función escrita en python:

```
1  def count_edges():
2  edges = 0
3
4  #Read Enlaces.txt
5  edge_file = open(PATH_TOPOS + FILE_NAME_EDGES, 'r')
6  try:
7      for line in edge_file:
8          edges = edges + 1
9  finally:
10     edge_file.close()
11
12 return edges
```

El archivo de enlaces contiene tres campos separados por el carácter ';': *nodo origen*, *nodo destino* y *distancia entre los nodos.*

Hay que tener en cuenta que los enlaces son bidireccionales. A efectos prácticos, la nomenclatura *nodo origen* y *nodo destino* no es de importancia. Si en el campo origen indica *Nodo0*

y en el campo destino *Nodo1*, significa que hay que enlazar el *Nodo0* con el *Nodo1*. Además, sólo son necesarios los dos primeros campos para el establecimiento de los enlaces. Se puede prescindir del campo *distancia entre los nodos*.

La función, mostrada en la figura 3.9, lee línea a línea y separa el nodo origen y el nodo destino con el fin de establecer un extremo del par *veth* con un bridge, y el otro extremo, con el otro bridge. Dicha automatización se desarrolla en una función escrita en python:

```
1  def node_connection():
2  veth_A = 0
3  veth_B = 1
4
5  # READ FILE: Enlaces.txt
6  file_edge = open(PATH_TOPOS + FILE_NAME_EDGES, 'r')
7  try:
8      for line in file_edge:
9          #NODE CONNECTION ------> A <---> B
10         node_A = line.split(';')[0]
11         node_B = line.split(';')[1]
12
13         #VETH CONNECTIONS
14         command_A = 'sudo ip link set veth' + str(veth_A) + ' master br' + node_A + ' ; ' + 'sudo ip ↩
               ↪ link set up dev veth' + str(veth_A)
15         command_B = 'sudo ip link set veth' + str(veth_B) + ' master br' + node_B + ' ; ' + 'sudo ip ↩
               ↪ link set up dev veth' + str(veth_B)
16
17         os.system(command_A)
18         os.system(command_B)
19
20         veth_A = veth_A + 2
21         veth_B = veth_B + 2
22
23 finally:
24     file_edge.close()
```

3.4.4 Heterogeneidad

Hasta ahora solo se han desplegado *Raspberry Pis*, y el objetivo es que sea una red virtual heterogénea, es decir, que no solamente se quiere disponer de un solo tipo de arquitectura, sino que en la red confluyan dispositivos de diferentes arquitecturas.

La herramienta de automatización ha de disponer las características de las máquinas para llevar a cabo el despliegue deseado, por ello, se modifica la estructura del archivo de nodos. Se añaden dos campos nuevos: un campo para indicar el tipo de arquitectura, y otro campo para indicar el sistema operativo, como se muestra en la tabla 3.2.

ID del Nodo	Coordenada X	Coordenada Y	Arquitectura	Sistema operativo

Tabla 3.2: Nueva estructura de archivo de nodos

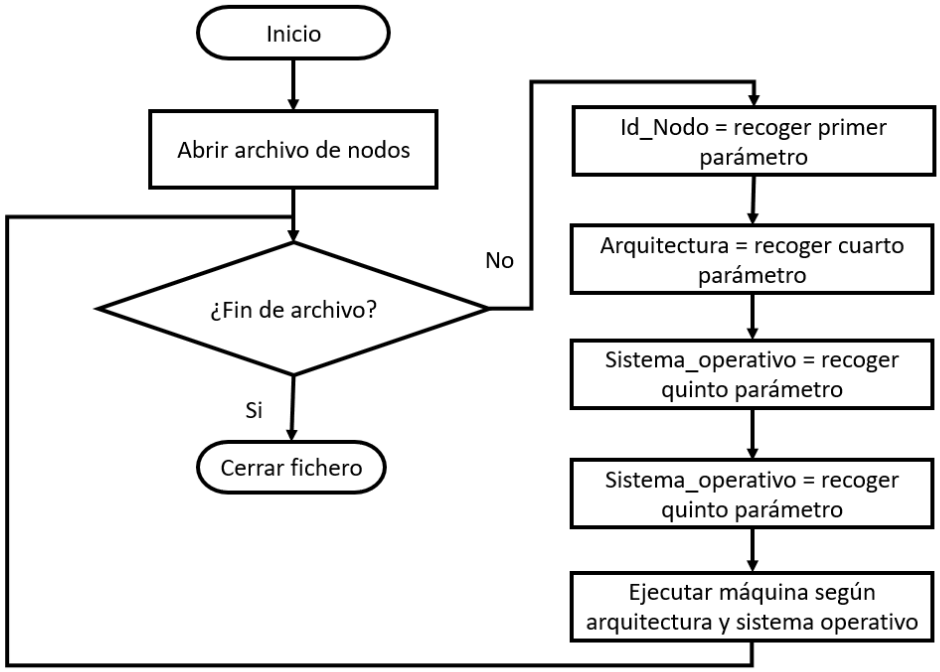


Figura 3.10: Diagrama lectura nodos

Con esta nueva estructura, la herramienta ya puede conocer tanto la arquitectura como el sistema operativo de la máquina que se quiere desplegar.

En la figura 3.10 se muestra el proceso que hay que seguir para leer y recoger las características de las máquinas virtuales. Consiste en la obtención de tres parámetros: el ID del nodo, la arquitectura y el sistema operativo. De esta manera se pueden desplegar máquinas a elección del usuario consiguiendo una red virtual compuesta por dispositivos de diferentes arquitecturas.

A raíz del diagrama observado en la figura 3.10 se desarrolla la siguiente función escrita en python:

```
1   def deploy_vm():
2
3   ARCH = ("arm" , "i386", "x86_64")
4   DISKS = ("q4os-disk", "alpine-disk")
5   OS_ARCH32 = ("q4os-4.8-i386-instcd.r2.iso") #Operating Systems for i364
6   OS_ARCH64 = ("alpine-standard-3.15.4-x86_64.iso") #Operating Systems for x86_64
7   DISK_ARCH32 = ("q4os-disk") #Hard Disks for intel 32 bits
8   DISK_ARCH64 = ("alpine-disk") #Hard Disks for intel 64 bits
9
10
11  # READ FILE: Nodos.txt
12  file_nodes = open(PATH_TOPOS + FILE_NAME_NODES, 'r')
13  try:
14      for line in file_nodes:
15          id_node = line.split(";")[0]
16          arch = line.split(';')[3]
```

```
17          o_system = line.split(';')[4].split('\n')[0]
18
19          if arch == ARCH[1]: #i386
20              if o_system == OS_ARCH32:
21                  #execute i386
22                  os.system(PATH_VM + "intel.sh " + id_node + " " + arch + " " + OS_ARCH32 + " " ↩
                        ↪ + DISK_ARCH32 + " " + random_mac())
23
24          elif arch == ARCH[2]: #i86_64
25              if o_system == OS_ARCH64:
26                  #execte x86_64
27                  os.system(PATH_VM + "intel.sh " + id_node + " " + arch + " " + OS_ARCH64 + " " ↩
                        ↪ + DISK_ARCH64 + " " + random_mac())
28
29          else: #ARM arch
30              os.system(PATH_VM + "rpi.sh " + id_node + " " + random_mac())
31
32      finally:
33          file_nodes.close()
```

3.4.5 Automatización Global

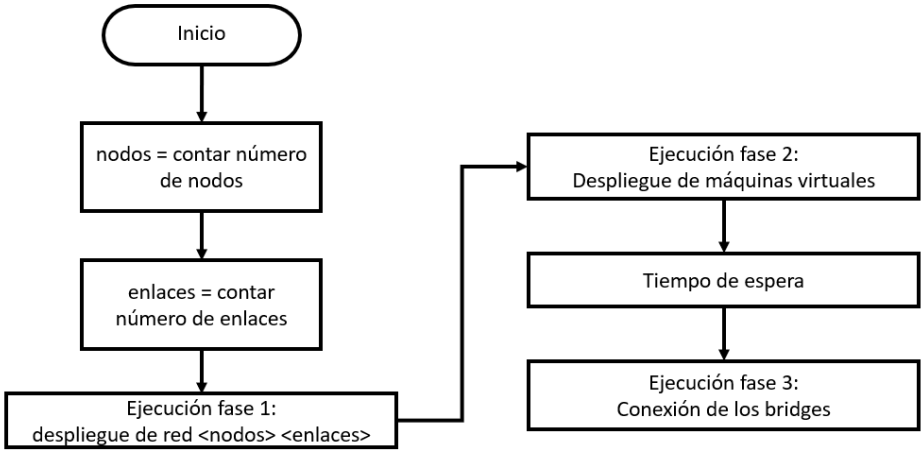


Figura 3.11: Diagrama automatización global

La automatización global consiste en la ejecución de todas las fases en un mismo proceso de manera secuencial y automática.

En la figura 3.11 se puede observar el proceso a seguir. Primero se leen la cantidad de nodos y enlaces de los archivos que caracterizan a la topología. Una vez leídos se ejecuta el despliegue de las interfaces y el arranque de las máquinas virtuales. Se espera un determinado tiempo, que ha sido establecido en 120 segundos, para que las máquinas virtuales obtengan la dirección IP del bridge correspondiente (En el apartado 3.5.3 se explica la obtención de este tiempo). Cuando todas las máquinas virtuales han obtenido su dirección IP, se ejecuta

la conexión de los bridges.

El proceso mostrado en la figura 3.11 se puede desarrollar en Python de manera sencilla mediante el siguiente código:

Código 3.5: herramienta.py

```
import os, time

from read_files.read_node_edges import *

PATH = "/home/david/TFG/heterogeneidad/"

def main():
    #Count nodes and edges
    nodes = count_nodes()
    edges = count_edges()

    print("Nodos = " + str(nodes))
    print("Edges = " + str(edges))

    #deploy net
    os.system(PATH + "net_v3.sh " + str(nodes) + " " + str(edges))

    #deploy RPis
    deploy_vm()

    #Wait to assign IP address to each RPi
    time.sleep(120)

    #Establishing edges
    node_connection()

if __name__ == "__main__":
    main()
```

3.4.6 Eliminación de la red vitual

La red virtual no ha de residir indefinidamente dado que consume recursos en el ordenador. Solamente ha de permanecer cuando se hagan pruebas, en el momento que no se tenga intención de utilizar la red se debería de eliminar.

En la figura 3.12 se observa el diagrama de fujos con el fin de elimimar la red. El número de nodos y el número de enlaces se han de pasar como argumentos a la script. De esta manera se eliminan todos los dispositivos e interfaces desplegados manteniéndose únicamente las interfaces físicas.

También se debe eliminar la tabla NAT y la interfaz VLAN (de donde cuelgan las interfaces de la red virtual). Además, el archivo donde se almacenan las direcciones IP, usadas por los servidores DHCP ha de vaciarse. Este archivo enlaza la dirección IP con la direción MAC de la MV. Si en el arranque de una MV no coincide la dirección MAC con la asociada a la dirección IP del archivo, la MV no obtendrá una dirección IP. Por este motivo se debe vaciar el archivo.

No solamente se han de eliminar los elementos anteriores, sino también hay que parar los procesos que intervienen en el despliegue de la red, como los procesos de QEMU, y procesos

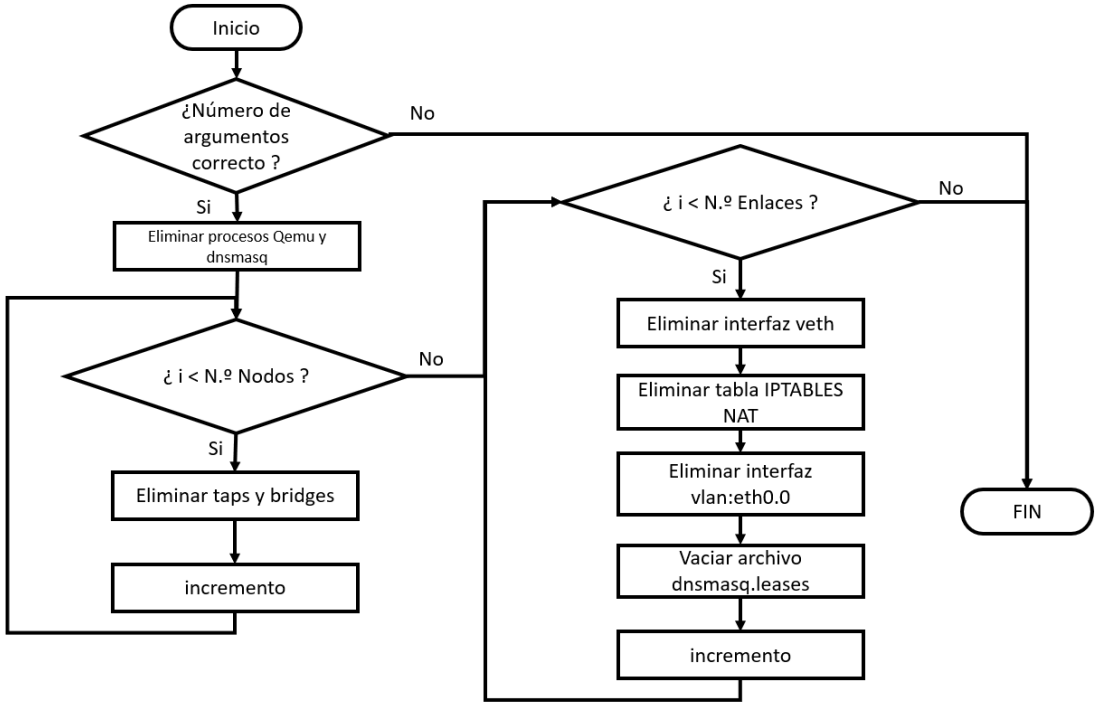


Figura 3.12: Diagrama del proceso de automatización para eliminar la red virtual

dnsmasq (proceso del servidor DHCP del bridge).

La forma más fácil de cerrar una máquina virtual es pulsando el botón de apagado, pero este método no es válido, ya que tiene que interaccionar el usuario para apagar las máquinas. Una forma más complicada pero más adecuada es eliminar el proceso de las máquinas de QEMU que se están ejecutando. Este mismo método se usa para eliminar los procesos "dnsmasq". El desarrollo para la eliminación de los procesos se puede observar en la figura 3.13.

Primero se observa qué procesos QEMU y "dnsmsaq" están en ejecución. Cada proceso contiene un número identificativo denominado PID, ver figura 3.14, que se va a guardar en un archivo para posteriormente leerlo a través de otro programa y eliminarlo mediante el comando:

```
os.kill(int(pid), SIGTERM)
```

La eliminación de la red es un procedimiento muy importante. Todas las herramientas ejecutadas consumen recursos de memoria y de CPU del ordenador. Si no se utiliza la red y no se eliminan los procesos, se están desperdiciando recursos llegando a causar problemas, tales como, el llenado de memoria RAM, imposibilitando la ejecución de otros programas o la ralentización de la CPU.

Uno de los problemas observados al no eliminar los procesos "dnsmasq" es la imposibilidad de ejecutar nuevos procesos "dnsmasq", como se observa en la figura 3.15.Esto es debido a la acumulación de los procesos "dnsmasq" no eliminados de otras redes desplegadas. Lo mismo

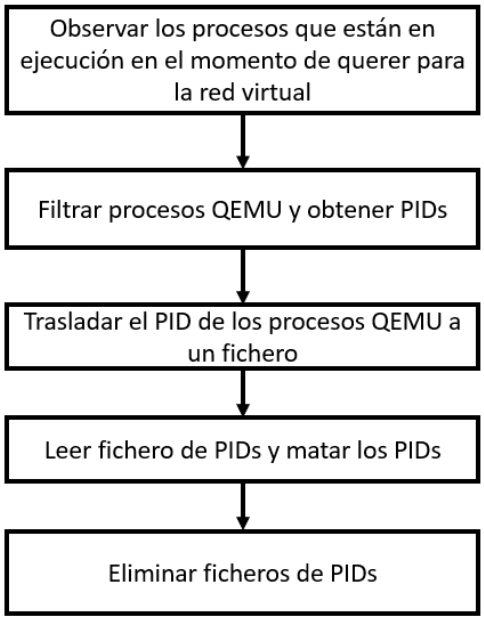


Figura 3.13: Diagrama eliminación de procesos

sucede cuando la memoria RAM se encuentra al límite de su capacidad y se quiere desplegar otra red virtual, se conseguiría desplegar las interfaces de red, pero no se ejecutaría QEMU.

Atendiendo a la figura 3.12 se desarrolla el siguiente programa para eliminar la red virtual:

Código 3.6: clean_net.sh

```
#!/bin/bash

if [ "$#" -ne 2 ]; then
    echo "Incorrect number of arguments"
    echo "Type: $# <number of nodes> <numer of edges"
    exit 0
fi

N_NODES=$1
N_EDGES=$2

ID_VETH=0

#Close RPis
ps -e | grep qemu | awk '{print $1}' > /home/david/TFG/heterogeneidad/pid_process.txt
ps -e | grep dnsmasq | awk '{print $1}' >> /home/david/TFG/heterogeneidad/pid_process.txt
python3 /home/david/TFG/heterogeneidad/read_files/close_qemu.py
rm /home/david/TFG/heterogeneidad/pid_process.txt

```

```
david@david-Lenovo:~/TFG/heterogeneidad$ ps -e | grep qemu
   5262 pts/0    00:00:45 qemu-system-arm
   5272 pts/0    00:00:45 qemu-system-arm
   5282 pts/0    00:00:43 qemu-system-arm
   5289 pts/0    00:00:43 qemu-system-arm
david@david-Lenovo:~/TFG/heterogeneidad$ ps -e | grep qemu | awk '{print $1}'
5262
5272
5282
5289
```

Figura 3.14: Procesos QEMU

```
dnsmasq: failed to create inotify: Demasiados archivos abiertos
dnsmasq: failed to create inotify: Demasiados archivos abiertos
dnsmasq: failed to create inotify: Demasiados archivos abiertos
dnsmasq: failed to create inotify: Demasiados archivos abiertos
```

Figura 3.15: Error dnsmasq

```
20
21 #DELETE BRIDGES, TAPS, AND DIRECTORIES
22 for ((id=0; id<${N_NODES}; id++ )); do
23     #Interfaces down
24     sudo ip link set dev tap${id} down
25     sudo ip link set dev br${id} down
26
27     #Delete interfaces
28     sudo ip link del tap${id}
29     sudo ip link del br${id}
30
31     #delete nodes directories
32     rm -rf /home/david/TFG/heterogeneidad/nodes/
33
34 done
35
36 #DELETE VETHS
37 for ((id=0; id<${N_EDGES}; id++ )); do
38     sudo ip link set dev veth${ID_VETH} down
39     sudo ip link del veth${ID_VETH}
40     ID_VETH=$((ID_VETH + 2))
41 done
42
43 #Delete IPTABLES NAT
44 sudo iptables -t nat -F
45
46 #Delete eth0.0 interface
47 sudo ip link del eth0.0
48
49 #Clean dhcp leases
50 cat /dev/null | sudo tee /var/lib/misc/dnsmasq.leases
```

3.5 Limitaciones de la herramienta

El despliegue de la red está limitado por los recursos del ordenador: memoria RAM, disco de almacenamiento, procesador, etc. Así que esnta sección se van a explicar diversas limitaciones que se han encentrado al desarrollar el despliegue de la red.

3.5.1 Elección de los sistemas operativos para las máquinas virtuales

La elección de los sistemas operativos se ha definido por la limitación de la memoria RAM. Han de ser lo más ligero posibles, ya que solo necesitan las funcionalidades básicas como poder configurar la red, editar archivos o realizar un ping para comprobar la conexión con otras máquinas.

Con esta limitación se escogen sistemas operativos que se ejecuten con poca memoria RAM. Para la emulación de la Raspberry Pi se eligió un sistema operativo basado en *Raspbian*, que no fue la versión original, sino una versión compatible con *arm1176*. Para la arquitectura *Intel* de 32 bits se escoge el sistema operativo *Q4OS* y para *Inte*l de 64 bits, *Alpine*. Todos ellos tienen una característica en común: la capacidad mínima requerida para arrancar dichas máquinas es de 256 MB.

3.5.2 Número máximo de nodos a desplegar

En el apartado 4.1.9 se realiza una prueba para establecer el límite de MVs a desplegar según la capacidad de la memoria RAM del ordenador. El resultado es 19, no obstante, el ordenador está al límite de su capacidad.

Para que la memoria RAM se sitúe como máximo en la mitad de su capacidad, el límite se establece en 10 MVs, por tanto, se van a desplegar como máximo 10 nodos.

3.5.3 Tiempo de espera entre el despliegue de las MVs y la interconexión de los bridges

Entre la ejecución del programa para automatizar el despliegue de las máquinas virtuales y la ejecución del programa para enlazar los bridges hay que esperar un determinado tiempo. Esto es debido a que cada MV necesita obtener la dirección IP de su bridge correspondiente, es decir, del bridge con conexión directa.

Si primero se interconectan los bridges y posteriormente se despliegan las máquinas virtuales, la MV podría obtener una dirección IP de cualquier dispositivo bridge desplegado y la herramienta requiere que la MV obtenga la dirección IP del bridge enlazado directamente.

La determinación del tiempo se calcula como la diferencia de tiempo entre el momento de arranque de la MV hasta el instante que dicha MV recibe una dirección IP.

El tiempo es calculado a través de las marcas de tiempo registradas en el fichero */var/log/syslog*. Cuando arranca una MV, en el log del sistema queda registrado el evento de inicio,

y cuando la MV recibe la dirección IP también queda registrado el evento. En la figura 3.16 la máquina virtual se inicia a las *17:23:30* y obtiene la dirección IP a las *17:23:57*, así que el tiempo de asignación de la dirección IP en una MV es:

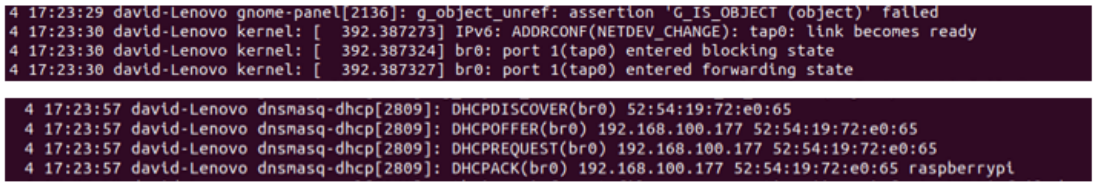


Figura 3.16: Tiempo de asignación de dirección IP

$$Tiempo = t = t_2 - t_1 = 17:23:57 - 17:23:30 = 27 segundos \tag{3.1}$$

Cuando se arranca una sola MV, el tiempo que se debe esperar para establecer los enlaces entre los bridges, es de 27 segundos. En el momento que se despliegan más máquinas, se producen una mayor cantidad de procesos, de modo que se gastan más recursos de memoria, de almacenamiento, de CPU, etc y ralentiza el ordenador. Debido a esto, a mayor número de nodos, mayor es el tiempo.

Como se ha establecido un límite de diez nodos a desplegar, se establece que el tiempo desde que se ejecuta el programa de despliegue de máquinas virtuales hasta que la decima máquina obtiene la dirección IP es de 96 segundos. se deja un margen por si el ordenador ejecuta otros procesos, estableciéndose el tiempo de espera en 120 segundos. Este tiempo

3.6 Interfaz web

Tras el desarrollo de las automatizaciones se da paso a la segunda y última parte del proyecto. Consiste en la elaboración de una aplicación web con arquitectura cliente-servidor mediante un microframework de Python, denominado *Flask*.

La aplicación se construye en base a tres páginas web, una página para enviar los archivos al servidor, otra para seleccionar la arquitectura de la MV y por último, una página para mostrar una imagen de la red desplegada e información de las máquinas virtuales y eliminar la red vitual.

3.6.1 Primera pagína web. Envío de ficheros al servidor

El diseño de la pagína web es el mostrado en la figura 3.17. Es una página sencilla que contiene dos etiquetas, una para introducir el archivo de nodos y la otra para el de enlaces. Además, también contiene un botón, cuya finalidad es enviar los ficheros al servidor cuando sea pulsado.

En la figura 3.18 se muestra la lógica del servidor con el fin de visualizar la página web

Figura 3.17: Página web de inicio

e interactuar con ella. Cuando se accede a la url: *http:localhost:8080*, el servidor recibe una petición *GET* de la página web y devuelve como respuesta un documento HTML con los objetos contenidos. El usuario ve la página web e interacciona con ella seleccionando los ficheros. En el campo de nodos hay que seleccionar el archivo cuyo nombre ha de ser *Nodos.txt* o *Nodos.csv*, y para el campo de enlaces, hay que elegir el fichero con nombre *Enlaces.txt* o *Enlaces.csv*. En el momento que el usuario pulsa el botón, se envía una petición *POST* al servidor y éste la procesa. Si en ella no hay ficheros o si el nombre es incorrecto, devuelve una página de error, al igual que si la topología se excede de los diez nodos.

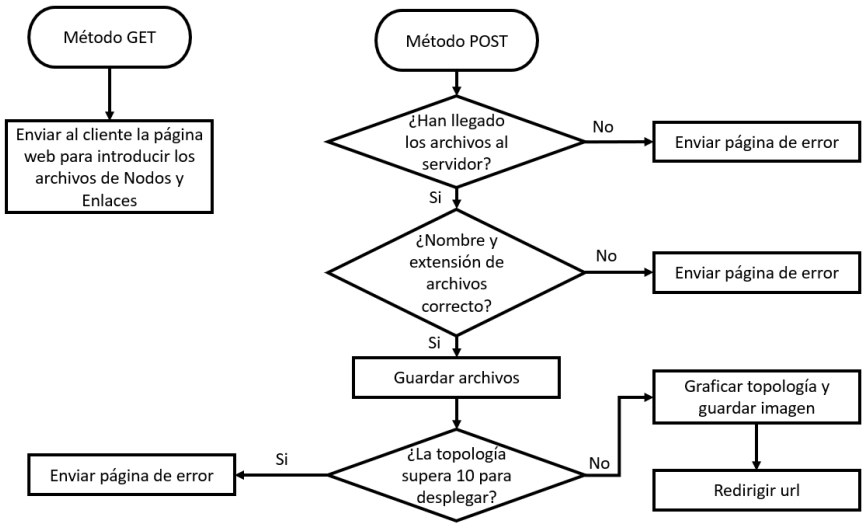


Figura 3.18: Servidor. Diagrama parte I

Una vez comprobado los ficheros, se guardan en un directorio donde está alojado el servidor y se grafica la topología para obtener una imagen visual de la descripción de los archivos a través de la herramienta *Graphviz* de Python. Se lee el archivo de enlaces y se escribe en

otro fichero la estructura requerida por *Graphviz*. *Graphviz* lee el nuevo archivo, grafica la topología y se guarda para un posterior uso. Finalmente, el cliente es redirigido a otra página a fin de elegir la arquitectura de las máquinas virtuales a desplegar.

3.6.2 Segunda página web. Selección de la arquitectura

El diseño de la segunda página web se centra en una tabla con dos columnas, una para el ID de la MV y otra para seleccionar la arquitectura. El número de filas queda determinado por el número de nodos.

La última sentencia que se observa en la figura 3.18, el servidor redirecciona la url de la web a la ruta:*https:localhost:8080/param*. El usuario obtiene la página web, mostrada en la figura 3.19, y selecciona para cada MV la arquitectura deseada. Una vez rellenado todos los campos se envía la información al servidor por medio de una petición *POST*. El servidor recoge los datos y los indexa en el campo *"arquitectura"* del fichero *Nodos.txt*. También se rellena el campo *"sistema operativo"* del mismo fichero, pero este campo no lo selecciona el usuario, ya que solo hay un sistema operativo para cada arquitectura. Debido a esto, el sistema operativo queda determinado por la arquitectura.

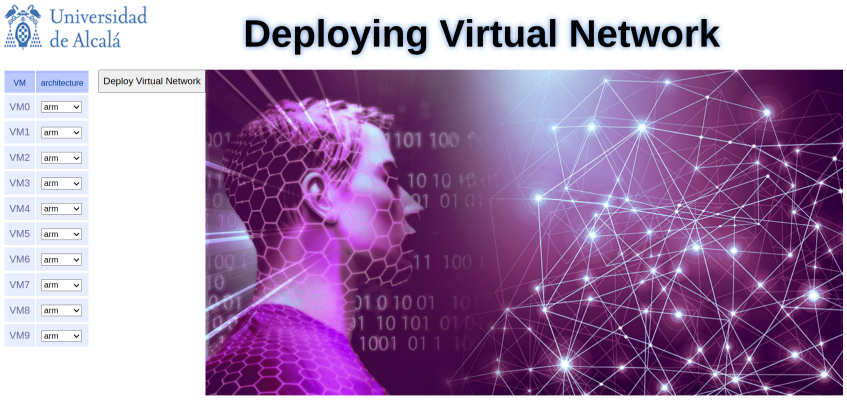


Figura 3.19: Página web para seleccionar arquitectura

Finalmente, el servidor ejecuta el programa visto en el apartado 3.4.5. Una vez finalizado el tiempo de despliegue de la red virtual, el servidor redirecciona la url a *http:localhost:8080/info-and-clean*.

La lógica del servidor se observa en la figura 3.20

3.6.3 Tercera página web. Mostrar información

Fundamentalmente, esta página web se centra en mostrar la información de la topología de red desplegada y eliminar la topología de red.

El diseño de la página web, como se observa en la figura 3.21 se focaliza en una tabla,

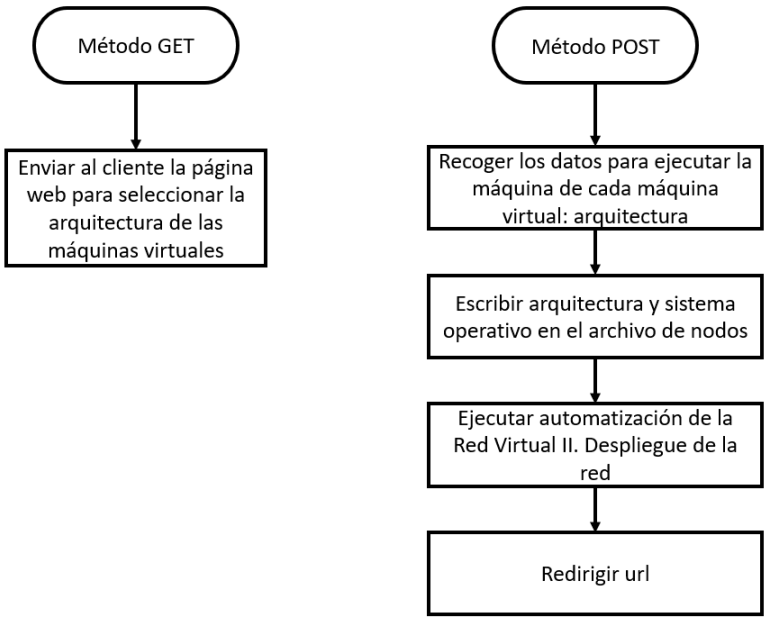


Figura 3.20: Servidor. Diagrama parte II

en una imagen y en un botón. La tabla contiene en seis columnas: ID de MV, arquitectura, sistema operativo, comando ssh, usuario y contraseña. Con esta información, el usuario puede conectarse a las máquinas virtuales de forma remota. La imagen es el grafo guardado anteriormente, de esta manera, el usuario puede visualizar la interconexión de los nodos. Y por último, el botón *"Stop Virtual Network"*, que envía la orden al servidor para ejecutar el programa de eliminación de la red virtual

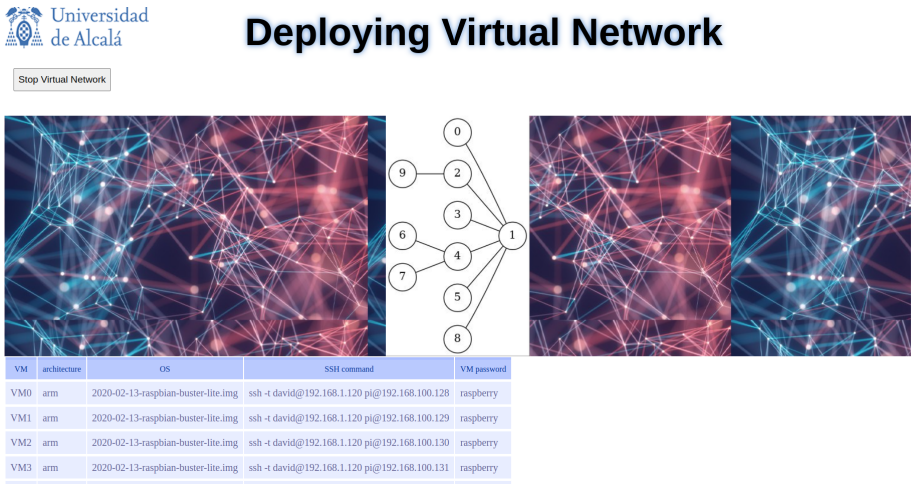


Figura 3.21: Página web para mostrar información de la red virtual desplegada

En la figura 3.22 se observa el proceso que sigue el servidor cuando la url es redirigida

a *http:localhost:8080/info-and-clean*. EL servidor envía la página web al cliente. Cuando el usuario pulse el botón *"Stop Virtual Network"*, se envía una petición *POST* al servidor. Éste ejecuta el programa para eliminar la red virtual y redirige la url a la página de inicio para poder desplegar otra red virtual.

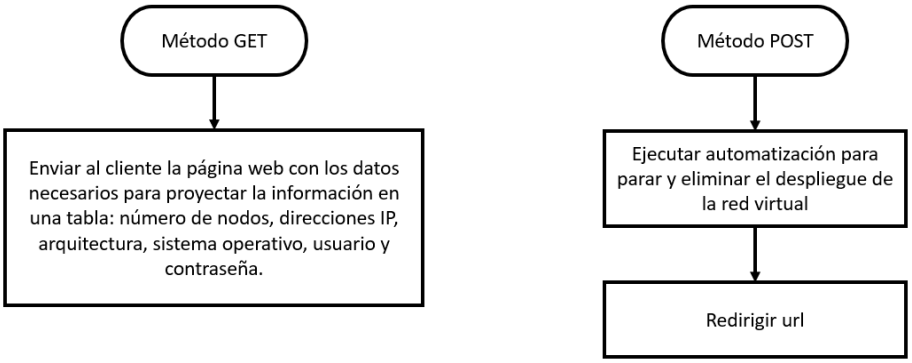


Figura 3.22: Servidor. Diagrama parte III

4 Pruebas y análisis de resultados

En este capítulo se expondrán las pruebas realizadas para comprobar el funcionamiento de la herramienta desarrollada. Por un lado, se mostrarán una serie de puebas unitarias y por otro lado, las pruebas de conjunto.

4.1 Pruebas unitarias

4.1.1 Prueba de despliegue de dispositivos e interfaces de la red

En esta prueba se va a comprobar el despliegue de los dispostivos e interfaces de la red del apartado 3.3.1. El resutaldo de la prueba se muestra en las figuras 4.1 y 4.2

En la figura 4.1 se observa que se han desplegado y encendido *br0*, *br1*, *br2*, *br3*, *tap0*, *tap1*, *tap2* y *tap3*. Cada bridge tiene una dirección IP y cada dispositivo tap está enlazado con su bridge correspondiente tal como se explicó en el apartado 3.3.1.

En la figura 4.2 se observa que se han desplegado 6 interfaces veth: *veth0* enlazada con *veth1*, *veth2* con *veth3* y *veth4* con *veth5*.

4.1.2 Prueba de despliegue de máquinas virtuales

Previamente, en la subsección 3.3.2 se explica cómo se emulan las arquitecturas *arm1176*, *i386* y *x86_64*, por lo que en este apartado se va a comprobar el despliegue de cada arquitectura.

```
10: br0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether ae:f8:b8:fa:22:38 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.2/24 brd 192.168.100.255 scope global br0
       valid_lft forever preferred_lft forever
11: br1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 72:76:1d:fb:9e:99 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.3/24 brd 192.168.100.255 scope global br1
       valid_lft forever preferred_lft forever
12: br2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether a2:0f:41:d1:f9:c4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.4/24 brd 192.168.100.255 scope global br2
       valid_lft forever preferred_lft forever
13: br3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether b6:d6:0c:e0:88:3a brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.5/24 brd 192.168.100.255 scope global br3
       valid_lft forever preferred_lft forever
14: tap0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel master br0 state DOWN group default qlen 1000
    link/ether ae:f8:b8:fa:22:38 brd ff:ff:ff:ff:ff:ff
15: tap1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel master br1 state DOWN group default qlen 1000
    link/ether 72:76:1d:fb:9e:99 brd ff:ff:ff:ff:ff:ff
16: tap2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel master br2 state DOWN group default qlen 1000
    link/ether a2:0f:41:d1:f9:c4 brd ff:ff:ff:ff:ff:ff
17: tap3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel master br3 state DOWN group default qlen 1000
    link/ether b6:d6:0c:e0:88:3a brd ff:ff:ff:ff:ff:ff
```

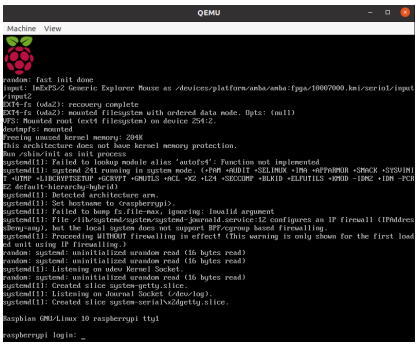
Figura 4.1: Prueba despliegue de dispostivos e interfacess

```
18: veth1@veth0: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether c6:43:8d:71:38:21 brd ff:ff:ff:ff:ff:ff
19: veth0@veth1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 56:df:41:f9:85:d4 brd ff:ff:ff:ff:ff:ff
20: veth3@veth2: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether aa:d9:e2:eb:86:de brd ff:ff:ff:ff:ff:ff
21: veth2@veth3: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 2a:1b:74:e3:0f:eb brd ff:ff:ff:ff:ff:ff
22: veth5@veth4: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 5a:15:5a:4c:aa:5e brd ff:ff:ff:ff:ff:ff
23: veth4@veth5: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 7a:95:d4:c0:cc:12 brd ff:ff:ff:ff:ff:ff
```

Figura 4.2: Interfaces veth desplegadas

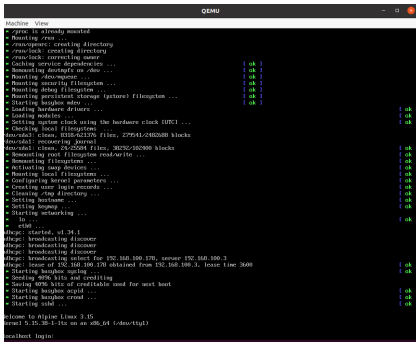
Se recuerda que para la arquitectura *i386* se configura el sistema operativo *Q4OS* y para *x86_64* se configura *Alpine*. Para la emulación de la Raspberry Pi se configura un kernel precompliado basado en *Raspbian*.

En la figura 4.3 se observa el despliegue de cada una de las tres arquitecturas.



(a) Emulación Raspberry Pi

(b) Emulacion *i386* configurado con *Q4OS*



(c) Emulacion *x86_64* configurado con *Alpine*

Figura 4.3: Despliegue de máquinas virtuales

4.1.3 Prueba interconexión de los bridges

En esta prueba se va a comprobar la interconexión de los bridges según se configuraron en el apartado 3.3.3.

```
18: veth1@veth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br1 state UP group default qlen 1000
    link/ether c6:43:8d:71:38:21 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::c443:8dff:fe71:3821/64 scope link
       valid_lft forever preferred_lft forever
19: veth0@veth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP group default qlen 1000
    link/ether 56:df:41:f9:85:d4 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::54df:41ff:fef9:85d4/64 scope link
       valid_lft forever preferred_lft forever
20: veth3@veth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br2 state UP group default qlen 1000
    link/ether aa:d9:e2:eb:86:de brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a8d9:e2ff:feeb:86de/64 scope link
       valid_lft forever preferred_lft forever
21: veth2@veth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP group default qlen 1000
    link/ether 2a:1b:74:e3:0f:eb brd ff:ff:ff:ff:ff:ff
    inet6 fe80::281b:74ff:fee3:feb/64 scope link
       valid_lft forever preferred_lft forever
22: veth5@veth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br3 state UP group default qlen 1000
    link/ether 5a:15:5a:4c:aa:5e brd ff:ff:ff:ff:ff:ff
    inet6 fe80::5815:5aff:fe4c:aa5e/64 scope link
       valid_lft forever preferred_lft forever
23: veth4@veth5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br2 state UP group default qlen 1000
    link/ether 7a:95:d4:c0:cc:12 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::7895:d4ff:fec0:cc12/64 scope link
       valid_lft forever preferred_lft forever
david@david-Lenovo:~/TFG/heterogeneidad/intel/pruebas$
```

Figura 4.4: Interconexión de los bridges

En la figura 4.4 se muestra que *br0* está enlazado con *veth0* y *veth2*; *br1* con *veth1*; *br2* con *veth3* y *veth4*; y *br3* con *veth5*. De esta manera se consigue disponer de los siguientes enlaces:*br0-br1*, *br0-br2* y *br2-br3*.

4.1.4 Prueba de despliegue manual para comprobar el funcionamiento de las interfaces veths

Para probar el funcionamiento de las interfaces veth enlazados con los bridges se diseña una topología basada en dos bridges enlazados entre sí, y cada bridge conectado a una MV.

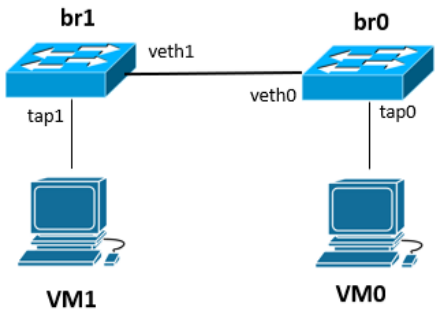


Figura 4.5: Topología para prueba de conexión entre bridges

Una vez desplegada la red de la figura 4.5, se comprueba el funcionamiento o no de la conexión entre *VM0* y *VM1* a través de los bridges. Para comprobar dicho funcionamiento se realiza un ping desde la máquina con dirección IP *192.168.100.10* a la máquina con dirección IP *192.168.100.11*.

En la figura 4.6 se comprueba que la máquina con dirección *192.168.100.10* está recibiendo respuesta de la máquina *192.168.100.11*. De esta manera se puede certificar la conexión entre

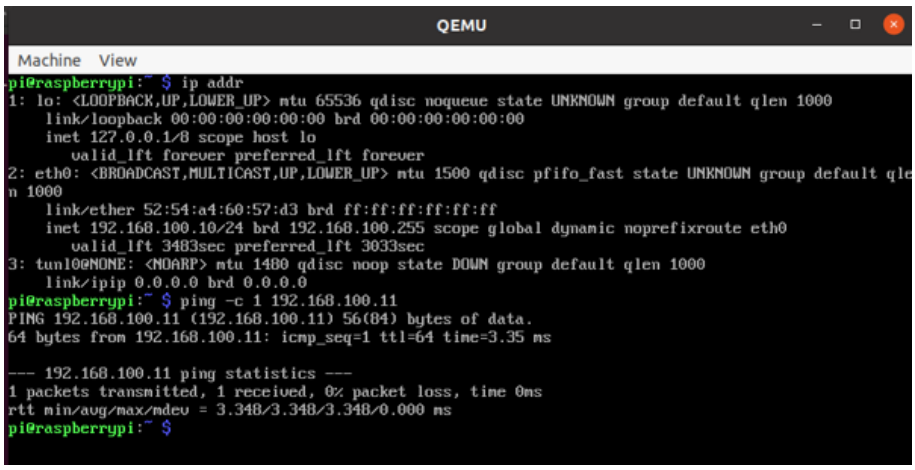


Figura 4.6: Ping desde VM0 a VM1

ambos sistemas a través de los bridges, puesto que las máquinas no se conectan directamente, sino que están conectadas indirectamente mediante los dispositivos bridge.

Otra prueba para confirmar el funcionamiento de estas interfaaces es deshabilitando el enlace entre *br0* y *br1*. Se desactivan las interfaces veth a través de estos comandos:

```
sudo ip link set veth0 down
sudo ip link set veth1 down
```

Al desactivar el enlace entre *br0* y *br1* y volver a realizar un ping entre las máquinas, como se puede observar en la figura 4.7, se verifica que no existe conexión entre ellas. En virtud a ello, se vuelve a comprobar que las interfaces veth (*veth0-veth1*) crean un enlace entre *br0* y *br1* consiguiendo la comunicación entre las máquinas.

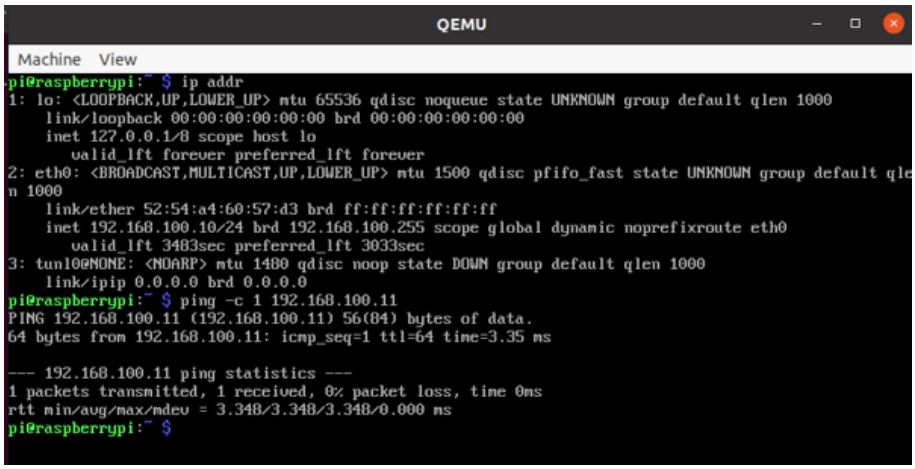


Figura 4.7: Ping desde VM0 a VM1 deshabilitando enlace

4.1.5 Comprobación de la asociación correcta MV-BRIDGE y NODO al arrancar las máquinas virtuales

En esta prueba se va a comprobar que la dirección IP de cada MV sea la deseada, cuyo propósito es conseguir la asociación *MV-BRIDGE* y *NODO*. Se despliega la red que se muestra en la figura 3.4 del apartado 3.3.4.

Arrancadas las máquinas virtuales (*VM0*, *VM1*, *VM2*, *VM3*), se capturan los mensajes DHCP en las interfaces *tap0*, *tap1*, *tap2*, *tap3*. El análisis de las capturas 4.8, 4.9, 4.10 y 4.11, se detalla en la tabla 4.1.

Interfaz	IP origen	Bridge	IP destino	MV
tap0	192.168.100.2	br0	192.168.100.177	VM0
tap1	192.168.100.3	br1	192.168.100.178	VM1
tap2	192.168.100.4	br2	192.168.100.179	VM2
tap3	192.168.100.5	br3	192.168.100.180	VM3

Tabla 4.1: Direcciones IP de la máquinas virtuales

Se verifica que cada MV obtiene la dirección IP del bridge correspondiente de la relación *MV-BRIDGE* y *NODO*:

- VM0-br0 -> NODO0
- VM1-br1 -> NODO1
- VM2-br2 -> NODO2
- VM3-br3 -> NODO3

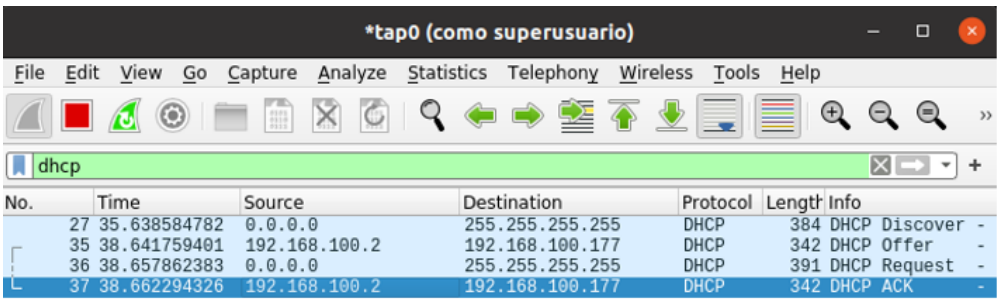


Figura 4.8: Dirección IP de la MV0

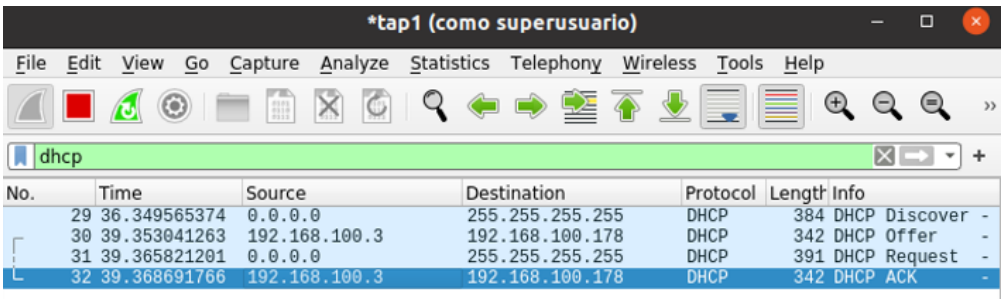


Figura 4.9: Dirección IP de la MV1

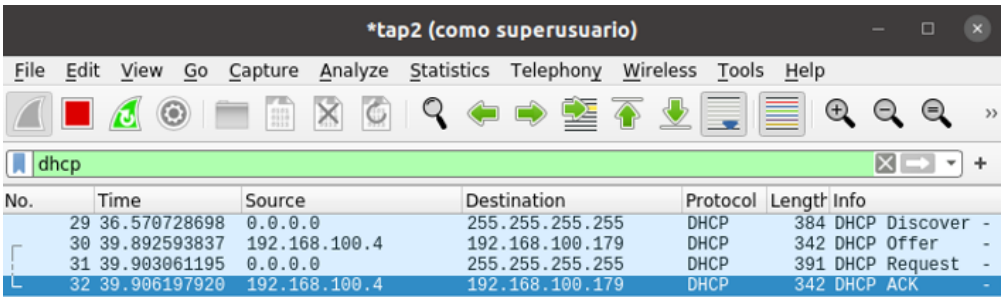


Figura 4.10: Dirección IP de la MV2

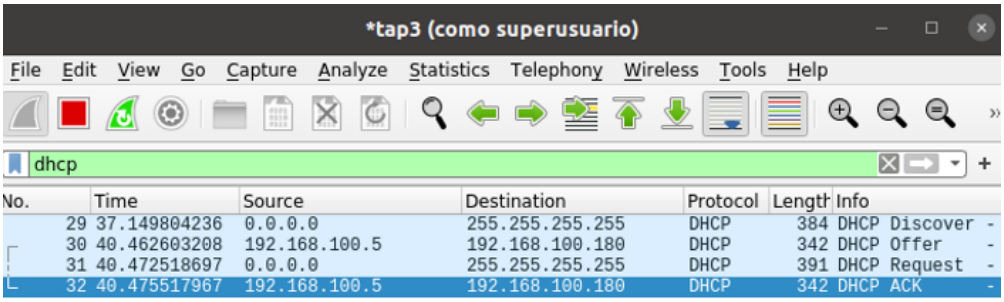


Figura 4.11: Dirección IP de la MV3

4.1.6 Prueba de conexión a Internet de las máquinas virtuales en modo puente y modo NAT

La prueba de conexión a Internet en modo puente se realiza con la MV que emula una Raspberry Pi. Está conectada directamente con el bridge y este con la interfaz física del host. A su vez, el host está conectado al router a través de su tarjeta de red física.

En la figura 4.12 se puede observar que la MV tiene una interfaz *eth0* configurada con dirección IP *192.168.1.29* perteneciente a la red LAN *192.168.1.0/24*, es decir, que la dirección IP ha sido proporcionada por el router físico. Al realizar un ping a *www.google.es* se comprueba que la máquina virtual tiene acceso a Internet.

Para la red en modo NAT se escoge la MV con arquitectura *i386*, que ha de obtener la dirección IP del servidor DHCP (configurado en el dispositivo bridge). En la figura 4.13 se

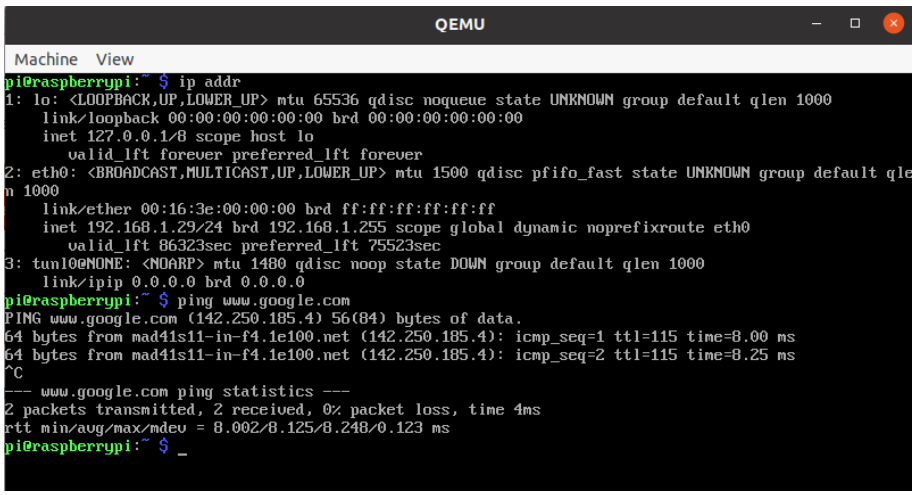


Figura 4.12: Prueba de conexión a Internet en modo puente

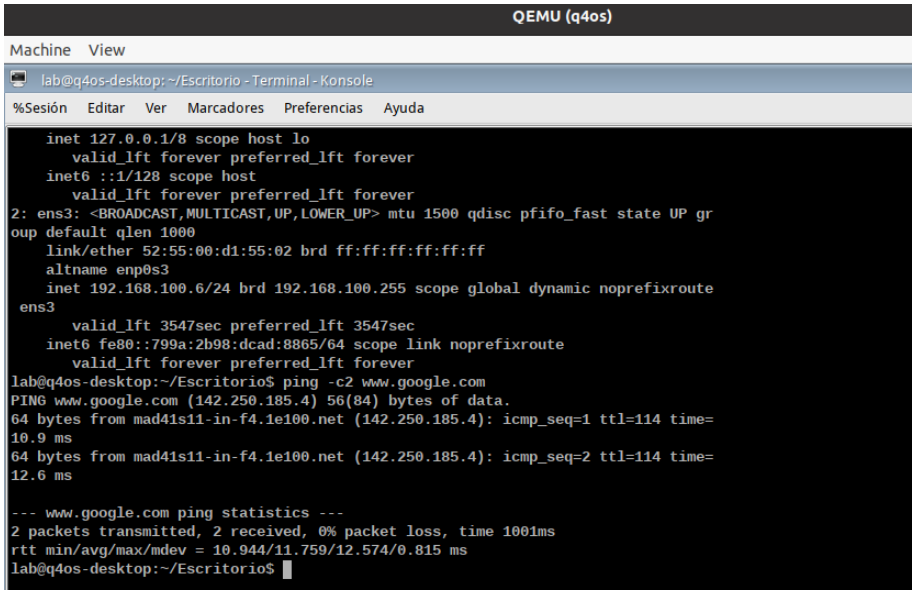


Figura 4.13: Prueba de conexión a Internet en modo nat

observa que la máquina obtiene de manera correcta la dirección IP *192.168.100.6*.

Con el fin de comprobar la conexión a Internet, se realiza un ping a *www.google.es* (servidor de la red externa). Como se observa en la figura 4.13, se obtiene respuesta por parte del servidor verificando que la máquina puede acceder a Internet.

4.1.7 Prueba de bucles en capa 2

Levantado el diseño de la figura 4.14, (*br0*, *br1* y *br2* forman un bucle), las máquinas virtuales no consiguen levantarse de manera satisfactoria. Se observa:

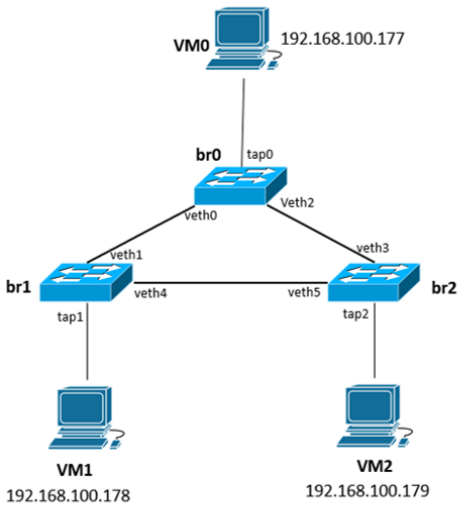


Figura 4.14: Topología formando un bucle

- PC ralentizado. Ventiladores funcionando al máximo.
- Inicio de máquinas virtuales bastante más lento.
- Las máquinas virtuales no arrancan de manera correcta. Aparece mensaje de error: "No se puede acceder a la consola, la cuenta root está bloqueada. Pulsa Enter para continuar". Tras pulsar Enter, la MV no reacciona.

Internamente se genera una tormenta de broadcast. Los bridges retransmiten las tramas de difusión por todas sus interfaces. Cada bridge recibe la misma trama desde diferentes bridges, produciendo un aumento de carga en el procesador de estos dispositivos. En consecuencia, se degrada la red e imposibilita el correcto funcionamiento de las máquinas virtuales.

4.1.8 Prueba STP

Se despliega de nuevo la misma topología de la figura 4.14, pero en este caso, cada bridge tiene habilitado el protocolo STP como se observa en la figura 4.15.

- *Br0* tiene habilitado STP. Las interfaces que tiene conectadas son *eth0.0* (conexión con la tarjeta de red física), *tap0* (conexión con *VM0*), *veth0* (conexión con *br1*), *veth2* (conexión con *br2*).
- *Br1* tiene habilitado STP. Las interfaces conectadas son *tap1* (conexión con *VM1*), *veth1* (conexión con *br0*), *veth4* (conexión con *br2*).
- *Br2* tiene habilitado STP. Las interfaces conectadas son *tap2* (conexión con *VM2*), *veth3* (conexión con *br0*), *veth5* (conexión con *br1*).

En el momento de activar las interfaces veth, los puertos de los bridges se encuentran en

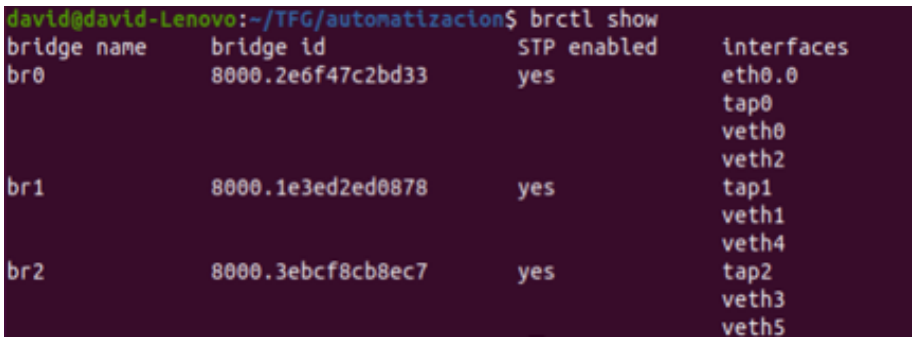


Figura 4.15: STP habilitado

modo *blocking* pasando a modo *listening*. En el modo *listening*, cada bridge podrá enviar y recibir *BPDUs*, tanto de configuración como TCNs, con el objetivo de determinar el *switch raíz* y el puerto *raíz* y *designado* de cada bridge. En este estado, si un determinado puerto no es *raíz* o *designado*, el puerto es bloqueado. Los puertos que no efectúan el cambio de estado de *listening* a *learning* se fijan en estado *bloqueado*.

Hay que tener en cuenta que cuando se inician los bridges y las interfaces, no se establecen todos al mismo tiempo. Al introducir un cambio en la red, esta recalcula y establece la topología STP. Es un proceso continuo hasta que se despliegan todos los componentes de la topología y la red se estabiliza. En caso de no haber ninguna modificación de la red, la topología STP se mantiene estable.

Tras el intercambio de tramas entre los bridges se escogen los puertos *raíz* y *designados* y el switch *raíz*. La red ha seleccionado al bridge *br1* como switch raíz, y sus puertos *veth1* y *veth4* como puertos *designados*. Los puertos de los demás bridges son *raíz*, *designados* o *bloqueados*. El puerto *veth0* y *veth5* son puertos *raíz*. Los puertos *veth1*, *veth3*, y *veth4* son puertos designados *veth2* es el puerto *bloqueado*, lo que significa que por dicho enlace no se transmiten paquetes.

Los bridges *br0* y *br2* no poseen comunicación directa, como consecuencia tampoco las máquinas virtuales conectadas a ellos: *VM0* y *VM2*. El intercambio de paquetes entre dichas máquinas virtuales se realiza a través de *br1*.

Se realiza un ping desde cada MV a las otras comprobando que existe comunicación entre todas.

El ping realizado con origen *192.168.100.177 (VM0)* con destino *192.168.100.178 (VM1)* debe transmitir el paquete a través de la interfaz *veth0*. En la figura 4.17 se comprueba que el paquete enviado desde *VM0* a *VM1* se transmite a través del puerto *veth0*.

Para comprobar el bloqueo de *veth2* y por tanto, la inutilización del enlace *br0-br2* se envía un ping desde *VM0* a *VM2*. El paquete se envía por el puerto *veth0*, llega a *br1* y lo retrasmite por *veth4* hasta llegar a *br2* que se lo envía a *VM2*.

De esta manera queda comprobado el correcto funcionamiento del protocolo STP en el desarrollo de la herramienta de automatización.

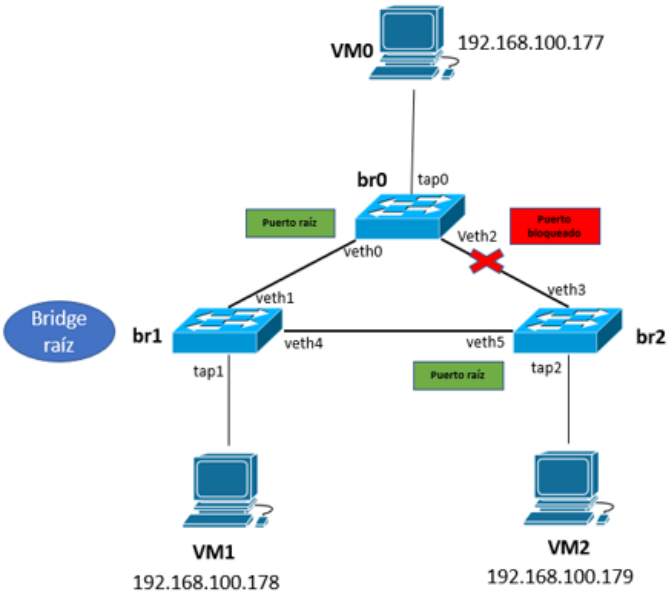


Figura 4.16: Topología STP

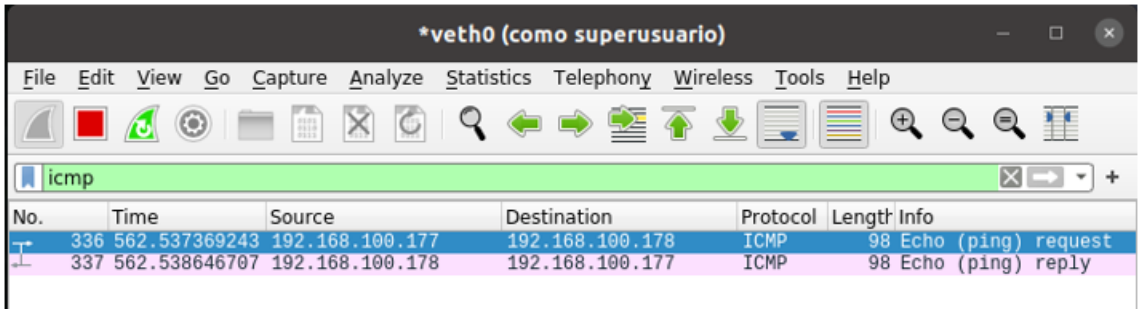


Figura 4.17: Captura de Wireshark. Ping desde VM0 a VM1

4.1.9 Determinación del número máximo de máquinas virtuales

El ordenador donde se despliega la red es un Lenovo Yoga 9i con procesador Intel de undécima generación y 16 GB de memoria RAM.

Las máquinas virtuales consumen memoria RAM del ordenador. Si este ordenador tiene 16 GB RAM de los cuales son usables 15,6 GB (aproximación a la baja de 15 GB) y cada MV necesita 256 MB de RAM como mucho se podrían desplegar 60 máquinas. Este cálculo es teórico e ideal. En el ordenador se están ejecutando otros procesos que también consumen memoria como los elementos de la red virtual (interfaces, dispositivos de red, protocolos, etc.) y el servidor web. La memoria restante es la memoria que se utiliza para levantar las máquinas virtuales.

Para determinar el número máximo de máquinas a desplegar se han realizado una serie de despliegues, en el que cada uno de ellos se ha ejecutado un número distinto de máquinas virtuales. Se ha comenzado desplegando 4 máquinas hasta llegar al fallo por la limitación de los recursos. El fallo se produce cuando se despliega la red con 19 máquinas.

En la tabla 4.2 se obtiene la relación de la memoria RAM usada respecto a la cantidad de nodos desplegados.

Nodos	RAM usada
4	4.8
5	5.7
6	6.7
7	7.6
8	8.5
9	9.1
10	9.8
11	10.0
12	11.0
13	12.0
14	12.0
15	13.0
16	13.0
17	14.0
18	14.0

Tabla 4.2: RAM usada por nodos desplegados

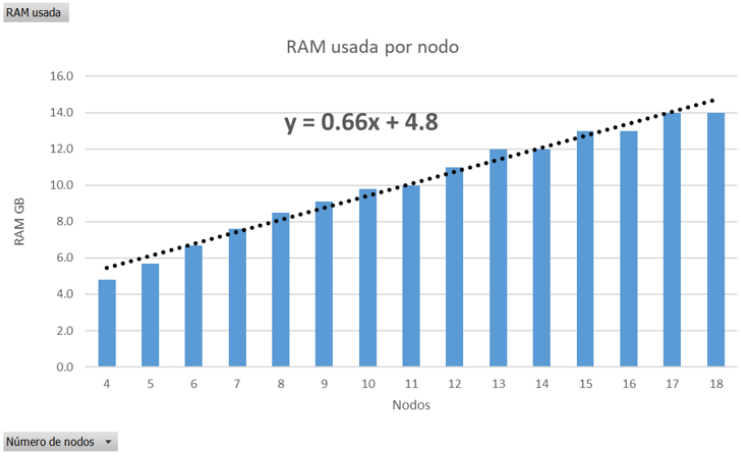


Figura 4.18: Grafico de la RAM usada por el número de nodos desplegados

En la figura 4.18 la tendencia de memoria de RAM usada según el número de nodos es:

$$RAM(GB) = 0.66 \cdot nodos + 4.8 \tag{4.1}$$

Con esta fórmula se puede conocer de manera aproximada la cantidad de memoria RAM que se va a necesitar para desplegar un determinado número de nodos.

4.2 Pruebas de Conjunto

4.2.1 Prueba de la aplicación: Herramienta para el despliegue automático de redes virtuales

Un vez finalizada la interfaz web se procede a comprobar la herramienta completa, es decir, la interfaz web junto con el despliegue automatático y la eliminación de la red.

El servidor únicamente acepta peticiones de direcciones IP de la red local, así que, la prueba se va a desarrollar en otro ordenador de la red donde no se esté ejecutando el servidor.

La URL está compusta por la dirección IP y el puerto de escucha del servidor. Cuando se realizó esta prueba la URL fue: http://192.168.1.120:8080. En la figura 3.17 del apartado 3.6.1 se observa la pagína web de inicio.

Se decide desplegar una topología barabasi de 10 nodos y grado 2. Así que se introducen los archivos de nodos y enlaces en los campos correspondientes. Al pulsar el botón para enviar los archivos al servidor, se redirecciona a la pagína web para elegir la arquitectura de las máquinas virtuales. Una vez que el servidor ha desplegado la red virtual, se muestra la última página web.

Mediante el comando ssh que proporciona la tabla de la página web que se observa en la figura 4.19, se procede a conectarse de manera remota a una MV, por ejemplo, *VM9*. La conexión establecida se muestra en la figura 4.21. Se realiza lo mismo para *VM7*.

Desde *VM7* se va a comprobar el funcionamiento de la red desplegada. Al realiar un ping a *www.google.com*, como se observa en la figura 4.22, se comprueba la conexión a Internet. Como se explicó en el apartado 3.4.1.3, las máquinas virtuales tienen acceso a Internet a través del bridge *br0*, ya que, el *nodo0* es el único nodo que está conectado directamente con la tarjeta de red fśica del ordenador. De esta manera, se puede comprobar que, los paquetes con dirección hacia la red externa, tienen que pasar por los nodos intermedios siguiendo la descripción topológica que se observa en la figura 4.20, hasta llegar al *nodo0*.

Del mismo modo, al realizar un ping a cualquier máquina, que no tenga conexión directa, los paqutes han de pasar por los nodos intermedios. En la figura 4.22, se observa la comprobación de la conexión a Internet, así como la comunicación entre *VM7* y *VM9*

Una vez verificada las diferentes conexiones se inicia la comprobación para eliminar los componentes de la red virtual. Al pulsar el botón *Stop Virtual Network* el servidor ejecuta el programa para borrar todos los dispositivos y procesos que intervienen en la red virtual. En la figura 4.23 se observa que no existe ninguna interfaz de la red virtual.

VM	architecture	OS	SSH command	VM password
VM0	arm	2020-02-13-raspbian-buster-lite.img	ssh -t david@192.168.1.120 pi@192.168.100.128	raspberry
VM1	i386	q4os-4.8-i386-instcd.r2.iso	ssh -t david@192.168.1.120 lab@192.168.100.129	lab2022
VM2	x86_64	alpine-standard-3.15.4-x86_64.iso	ssh -t david@192.168.1.120 root@192.168.100.130	lab2022
VM3	arm	2020-02-13-raspbian-buster-lite.img	ssh -t david@192.168.1.120 pi@192.168.100.131	raspberry
VM4	i386	q4os-4.8-i386-instcd.r2.iso	ssh -t david@192.168.1.120 lab@192.168.100.132	lab2022
VM5	x86_64	alpine-standard-3.15.4-x86_64.iso	ssh -t david@192.168.1.120 root@192.168.100.133	lab2022
VM6	arm	2020-02-13-raspbian-buster-lite.img	ssh -t david@192.168.1.120 pi@192.168.100.134	raspberry
VM7	i386	q4os-4.8-i386-instcd.r2.iso	ssh -t david@192.168.1.120 lab@192.168.100.135	lab2022
VM8	x86_64	alpine-standard-3.15.4-x86_64.iso	ssh -t david@192.168.1.120 root@192.168.100.136	lab2022
VM9	arm	2020-02-13-raspbian-buster-lite.img	ssh -t david@192.168.1.120 pi@192.168.100.137	raspberry

Figura 4.19: Información de la red virtual desplegada

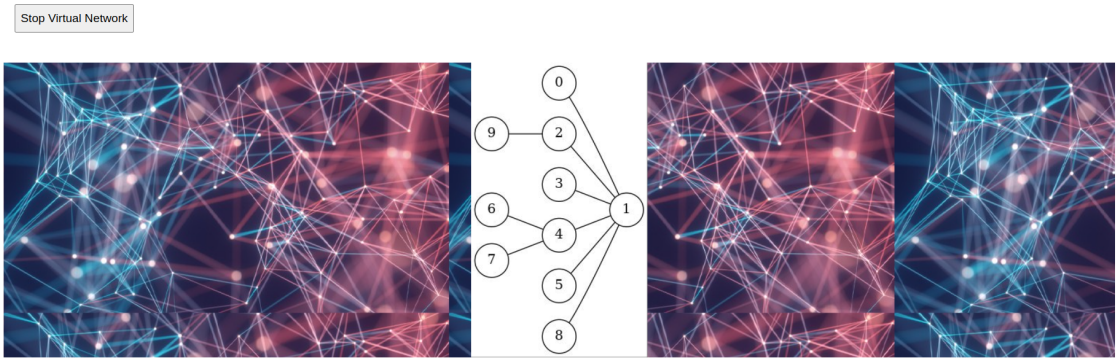


Figura 4.20: Visualización gráfica de la descripción topológica

Con esta prueba se comprueba el correcto funcionamiento de la herramienta para el despliegue automático de redes virtuales

```
pi@raspberrypi: ~
PS C:\Users\nerea> ssh -t david@192.168.1.120 ssh pi@192.168.100.137
david@192.168.1.120's password:
pi@192.168.100.137's password:
Linux raspberrypi 5.4.51 #1 Sat Aug 8 23:28:32 +03 2020 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Aug 25 13:17:06 2022 from 192.168.100.2

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

pi@raspberrypi:~ $
```

Figura 4.21: Conexión ssh a *VM9*

```
lab@q4os-desktop: ~
lab@q4os-desktop:~$ ping -c2 www.google.com
PING www.google.com (172.217.168.164) 56(84) bytes of data.
64 bytes from mad07s10-in-f4.1e100.net (172.217.168.164): icmp_seq=1 ttl=114 time=72.2 ms
64 bytes from mad07s10-in-f4.1e100.net (172.217.168.164): icmp_seq=2 ttl=114 time=11.7 ms

--- www.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 11.729/41.963/72.197/30.234 ms
lab@q4os-desktop:~$ ping -c2 192.168.100.137
PING 192.168.100.137 (192.168.100.137) 56(84) bytes of data.
64 bytes from 192.168.100.137: icmp_seq=1 ttl=64 time=5.41 ms
64 bytes from 192.168.100.137: icmp_seq=2 ttl=64 time=1.65 ms

--- 192.168.100.137 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.651/3.530/5.409/1.879 ms
lab@q4os-desktop:~$
```

Figura 4.22: Comprobación de conexiones desde *VM7*

```
david@david-Lenovo:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 38:fc:98:74:48:c0 brd ff:ff:ff:ff:ff:ff
    inet 172.20.10.11/28 brd 172.20.10.15 scope global dynamic noprefixroute wlp0s20f3
       valid_lft 86394sec preferred_lft 86394sec
    inet6 fe80::7211:cf14:a1e:ca80/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
david@david-Lenovo:~$
```

Figura 4.23: Interfaces eliminadas

5 Conclusiones y trabajo futuro

A modo de cierre de este TFG, primero se van a exponer las conclusiones mediante la revisión de los objetivos principales, y seguidamente, la línea futura de la herramienta desarrollada.

5.1 Conclusiones

Las conclusiones se van a tratar en base a la revisión de los dos principales objetivos.

Para comenzar, el primer objetivo principal definía el conjunto general de funciones capaces de realizar el despliegue de red virtual, el cual se descompuso en varios subobjetivos.

El primer subobjetivo consistió en el despliegue de máquinas virtuales con diferentes arquitecturas para conseguir una red virtual heterogénea. A través de varios scripts desarrollados en el apartado 3.3.2 se logran desplegar máquinas con arquitectura *i386* y *x86_64*, pertenenientes a *Intel*, y *arm1176* a *ARM*. En el apartado de pruebas 4.1.2 se verifica el correcto despligue de las máquinas virtuales. El segundo subobjetivo consistió en la interconexión de los nodos según la información contenida en el archivo generado por BRITE. La interconexión de los nodos se consigue mediante el despliegue de los enlaces para interconectar los bridges tal como se explicó en el apartado 3.3.3. En la prueba 4.1.4 se comprueba el buen funcionamiento de las interfaces veth para interconectar los bridges.

El tercer subobjetivo es el desarrollo de la automatización de la red virtual. La automatización se consigue mediante del dessarrollo de un conjunto de programas explicados a lo largo de la sección 3.4.6. En la prueba 4.2.1 se comprueba que los programas desarrollados funcionan correctamente.

Una vez revisados todos los subobejtivos definidos para logar el primer objetivo principal y, visto que todos ellos se han logrado, se puede dar por conseguido, también, el objetivo principal.

Una vez revisado el primer objetivo, se va a revisar el segundo y último objetivo principal que consistía en el desarrollo de una interfaz web para la herramienta de automatización. También se descompone en varios subobjetivos.

El primer subobjetivo se trató del desarrollo una página web para enviar los ficheros al servidor y el segundo subobjetivo el desarrollo del servidor web. Finalmente se crearon tres páginas web que interactúan con el servidor como se explicó en la sección 3.6.3. En la prueba 4.2.1 se puede comprobar que el servidor recoge los archivos de la primera página web, posteriormente, recoge las arquitecturas seleccionadas por el usuario final de la segunda página web y ejecuta el despliegue automático de la red. Por último, en la tercera página web, se

muestra información acerca de la red virtual desplegada.

En su conjunto, se puede determinar que tanto el primer objetivo como el segundo objetivo principal establecido, han sido logrados de manera satisfactoria. Por este motivo, se determina que la herramienta de automatización para el despliegue de redes virtuales es apta su uso.

5.2 Trabajo futuro

La herramienta desarrollada se utilizará en el laboratorio de investigación del departamento de automática de la Universidad de Alcalá de Henares. El corazón de la herramienta, es decir la lógica para la interconexión de los nodos está completamente diseñada, sin embargo, quedan ciertos detalles que mejorar en la herramienta:

- **Elección de las caracaterísticas para la configuración de las máquinas virtuales.** En la página web se puede seleccionar únicamente la arquitectura del dispositivo, puesto que, solo se implementó un sistema operativo para cada arquitectura. Se podría dar la opción de escoger más característcicas para crear máquinas virtuales más flexibles, por ejemplo, la capacidad del disco de almacenamiento y de la memoria RAM, el sistema operativo. Previamente habría que diseñar una máquina para cada tipo de característica teniendo en cuenta las capacidades del ordenador donde se esté ejecutando. Para llevarlo a cabo habría que realizar muchas comprobaciones en el momento de desplegar la red virtual. Por ejemplo, si se seleccionan diez nodos, cada uno con una memoria RAM de 2GB, el despliegue no se debería de ejecutar informando al usuario y explicando el motivo.
- **Comprobaciones de los archivos *Nodos* y *Enlaces* enviados al servidor.** Cuando el servidor recibe estos ficheros, solo se realiza la comprobación del nombre de los ficheros, de la extensión y del número de nodos a desplegar, sin embargo no se comprueba el contenido de ellos. Para mejorar dicho aspecto, habría que desarrollar una lógica para comparar la estructura de los ficheros enviados, con la estructura de un archivo que se procesa de manera correcta. Algunos ejemplos pueden ser el orden de los campos, la coherencia del valor de los campos. No pude ser que en el campo ID de nodo, haya valores alfabéticos y no numéricos.
- **Seguridad**. La seguridad es un tema muy importante en la actualidad, por lo tanto, la herramienta ha de ser segura en materia al desarrollo del código: seguridad en el diseño, seguridad en el despliegue. El servidor se ejecuta con permisos de administrador. Si el servidor sufre un ataque cibernético y entran en el sistema, los atacantes ya tienen el control del *"superusuario"*.

Estamos ante una herramienta aún en proceso de consolidación. Una vez que se mejore la aplicación con los puntos desarrollados anteriormente, puede llegar a ser una aplicación comercializable, aunque únicamente sea a nivel de investigación en Universidades.

Bibliografía

[1] J. F. Kurose, *Redes de computadoras : un enfoque descendente*, 7th ed. Madrid: Pearson Addison Wesley, 2017.

[2] B. García Ortega, "Industria 4.0. la cuarta revolución industrial," 2021.

[3] J. Gómez and E. Villar, "Introducción a la virtualización," Sep. 2018, accepted: 2018-09-25T17:01:26Z. [Online]. Available: http://148.202.167.116:8080/xmlui/handle/123456789/2273

[4] adm-rhiscom, "(Español) Simulación, Emulación o Virtualización en el proceso de Desarrollo para Canales de Venta - Rhiscom," Oct. 2020, section: Blog RHISCOM. [Online]. Available: https://rhiscom.com/en/simulacion-emulacion/

[5] G. Zimbrón, "Diferencia entre Simulación y Emulación," Aug. 2017. [Online]. Available: https://zimbronapps.com/educacion/sistemas-computacionales/simulacion/diferencia-simulacion-emulacion/

[6] "Historia de la Virtualización | HN Datacenter en Chile," Jan. 2021. [Online]. Available: https://www.hn.cl/blog/hablemos-de-virtualizacion/

[7] "Desarrollo de software de Altas Prestaciones - Una Vision historica de la Virtualizacion." [Online]. Available: https://sites.google.com/a/espe.edu.ec/desarrollo-de-software-de-altas-prestaciones/computo-de-altas-prestaciones/introduccion-a-las-maquinas-virtuales/emulacion/una-vision-historica-de-la-virtualizacion

[8] "Línea de tiempo de los 20 años de VMware." [Online]. Available: https://www.vmware.com/latam/timeline.html

[9] "What is Application Virtualization? | VMware Glossary." [Online]. Available: https://www.vmware.com/topics/glossary/content/application-virtualization

[10] "Virtualización de aplicaciones," Jul. 2012. [Online]. Available: https://inlab.fib.upc.edu/es/virtualizacion-de-aplicaciones

[11] adm-rhiscom, "(Español) Simulación, Emulación o Virtualización en el proceso de Desarrollo para Canales de Venta - Rhiscom," Oct. 2020, section: Blog RHISCOM. [Online]. Available: https://rhiscom.com/en/simulacion-emulacion/

[12] Y. F. Romero and K. G. Pombo, "Virtualización," *Telemática*, vol. 10, no. 3, pp. 61–73, 2011.

[13] "Virtualización: el alma de la nube." [Online]. Available: https://www.ionos.es/digitalguide/servidores/configuracion/virtualizacion/

[14] "Bridge vs Macvlan," Mar. 2016. [Online]. Available: https://hicu.be/bridge-vs-macvlan

[15] H. Liu, "Introduction to Linux interfaces for virtual networking," Oct. 2018. [Online]. Available: https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking

[16] "What is TUN/TAP? | Definition & use cases of TUN TAP." [Online]. Available: https://proprivacy.com/guides/tun-tap

[17] "Universal TUN/TAP device driver — The Linux Kernel documentation." [Online]. Available: https://www.kernel.org/doc/html/v5.8/networking/tuntap.html

[18] R. P. Fernández, "Cómo configurar un bonding de interfaces en GNU/Linux Debian." [Online]. Available: https://www.raulprietofernandez.net/blog/redes/como-configurar-un-bonding-de-interfaces-en-gnu-linux-debian

[19] "Difference between bonding and teaming in Linux ? | New Generation Enterprise Linux," Dec. 2018. [Online]. Available: https://ngelinux.com/difference-between-bonding-and-teaming-in-linux/

[20] "VXLAN | Blogs La Salle | Campus Barcelona." [Online]. Available: https://blogs.salleurl.edu/es/vxlan

[21] A. L. rosa, "¿Conoces las Redes VXLAN? Una red entre máquinas virtuales," Apr. 2019. [Online]. Available: https://pandorafms.com/blog/es/redes-vxlan/

[22] "veth(4) - Linux manual page." [Online]. Available: https://man7.org/linux/man-pages/man4/veth.4.html

[23] "CCN-PYTEC abr2021 Seguridad MACSEC.pdf."

[24] "SocketCAN - Controller Area Network — The Linux Kernel documentation." [Online]. Available: https://docs.kernel.org/networking/can.html

[25] "What Is Can Bus (Controller Area Network) | Dewesoft." [Online]. Available: https://dewesoft.com/daq/what-is-can-bus

[26] "netlink(7) - Linux manual page." [Online]. Available: https://man7.org/linux/man-pages/man7/netlink.7.html

[27] "Intermediate Functional Block." [Online]. Available: http://linux-ip.net/gl/tc-filters/tc-filters-node3.html

[28] F. Boronat Seguí, *Direccionamiento e interconexión de redes basada en TCP/IP. : (IPv4/IPv6, DHCP, NAT, Encaminamiento RIP u OSPF)*, ser. Académica, 2013.

[29] J. Handal, "Detección de bucles." [Online]. Available: https://nsrc.org/workshops/2013/walc/campus/raw-attachment/wiki/Agenda/STP.pdf

[30] "Spanning tree protocol," Apr. 2022, page Version ID: 1084114957. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Spanning_Tree_Protocol&oldid=1084114957

[31] O. Gerometta, "Mis Libros de Networking: Estado de los puertos STP," Sep. 2013. [Online]. Available: http://librosnetworking.blogspot.com/2013/09/estado-de-los-puertos-stp.html

[32] A. Balchunas, "Spanning tree protocol v1.31."

[33] "4.1.2.5 802.1D BPDU Frame Format." [Online]. Available: https://www.ii.pwr.edu.pl/~kano/course/module4/4.1.2.5/4.1.2.5.html

[34] "STP protocol frames." [Online]. Available: https://techhub.hpe.com/eginfolib/networking/docs/switches/5980/5200-3921_l2-lan_cg/content/499036672.htm

[35] S. Luján-Mora, *Programación de aplicaciones web: historia, principios básicos y clientes web*. Editorial Club Universitario, 2002.

[36] "Usage Statistics and Market Share of Web Servers, September 2022." [Online]. Available: https://w3techs.com/technologies/overview/web_server

[37] "NGINX." [Online]. Available: https://www.nginx.com/

[38] E. M. Talón, *Apache*. Ministerio de Educación, 2012.

[39] "LiteSpeed Web Server - Todo lo que Necesitas Saber." [Online]. Available: https://www.hostinet.com/hosting-web/litespeed-web-server-todo-lo-que-necesitas-saber/

[40] K. Schaefer, J. Cochran, S. Forsyth, D. Glendenning, and B. Perkins, *Professional Microsoft IIS 8*. John Wiley & Sons, 2012.

[41] "Capa de aplicación." [Online]. Available: http://dis.um.es/~lopezquesada/documentos/IES_1314/LMSGI/curso/xhtml/xhtml11/html/pag1.html

[42] H. Nielsen, R. T. Fielding, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.0," Internet Engineering Task Force, Request for Comments RFC 1945, May 1996, num Pages: 60. [Online]. Available: https://datatracker.ietf.org/doc/rfc1945

[43] "Mensajes HTTP - HTTP | MDN." [Online]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Messages

[44] J. D. Gauchat, *El gran libro de HTML5, CSS3 y Javascript*. Marcombo, 2012.

[45] S. G. Pérez Ibarra, J. R. Quispe, F. F. Mullicundo, and D. A. Lamas, "Herramientas y tecnologías para el desarrollo web desde el frontend al backend," in *XXIII Workshop de Investigadores en Ciencias de la Computación (WICC 2021, Chilecito, La Rioja)*, 2021.

[46] M. Grinberg, *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[47] "Lección 1: Guía básica para crear una aplicación Flask desde cero," Dec. 2018. [Online]. Available: https://j2logo.com/leccion-1-la-primera-aplicacion-flask/

[48] J. Forcier, P. Bissex, and W. J. Chun, *Python web development with Django*. Addison-Wesley Professional, 2008.

[49] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2001, pp. 346–353.

[50] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of IEEE INFOCOM'96. Conference on Computer Communications*, vol. 2. IEEE, 1996, pp. 594–602.

[51] K. L. Calvert, M. B. Doar, and E. W. Zegura, "Modeling internet topology," *IEEE Communications magazine*, vol. 35, no. 6, pp. 160–163, 1997.

[52] "QEMU." [Online]. Available: https://wiki.qemu.org/Main_Page

[53] "About QEMU — QEMU documentation." [Online]. Available: https://www.qemu.org/docs/master/about/index.html

[54] "Documentation/Networking - QEMU." [Online]. Available: https://wiki.qemu.org/Documentation/Networking

[55] "QEMU (Español) - ArchWiki." [Online]. Available: https://wiki.archlinux.org/title/QEMU_(Espa%C3%B1ol)#Trabajo_en_red_con_VDE2

[56] "What is the difference between dhcp3-server and isc-dhcp-server packages?" May 2012. [Online]. Available: https://askubuntu.com/q/140123

[57] "dnsmasq(8) — Arch manual pages." [Online]. Available: https://man.archlinux.org/man/dnsmasq.8.es

[58] "Graphviz." [Online]. Available: https://graphviz.org/

[59] "Intermediate Functional Block." [Online]. Available: http://linux-ip.net/gl/tc-filters/tc-filters-node3.html

[60] E. A. Marchionni, *Administrador de servidores*. USERSHOP, 2011, vol. 210.

Lista de Acrónimos y Abreviaturas

AP	Access Point.
ARP	Address Resolution Protocol.
ARP	Address Resolution Protocol.
AS	Autonomous System.
BOOTP	Bootstrap Protocol.
BPDU	Bridge Protocol Data Units.
BRITE	Boston University Representative Internet Topology Generator.
CPU	Central Processing Unit.
CSS	Cascading Style Sheets.
DHCP	Dynamic Host Configuration Protocol.
DNS	Domain Name Server.
FTP	File Transfer Protocol.
GUI	Graphical User Interface.
GWS	Google Web Server.
HTML	HyperText Markup Language.
HTTP	HyperText Transfer Protocol.
IA	Inteligenia Artificial.
ID	Identificador.
IEEE	Institute of Electrical and Electronics Engineers.
IETF	Internet Engineering Task Force.
IFB	Intermediate Functional Block.
IIS	Internet Information Services.
IoT	Internet de las Cosas.
IP	Internet Protocol.
IPv4	Internet Protocol Version 4.
IPv6	Internet Protocol Version 6.
ISO	International Organization for Standardization.
LACP	Link Access Control Protocol.
LAN	Local Area Network.
MAC	Media Access Control.
MV	Máquina virtual.
NAT	Network address Translation.
NIC	Network Interface Card.
OSI	Open Systems Interconnection.
PXE	Preboot Execution Environment.

QoS	Quality of Service.
RAM	Random Access Memory.
SFQ	Stochastic Fairness Queuing.
STA	Spannig Tree Algoirthm.
STP	Spanning Tree Protocol.
TC	Topology Change.
TCA	Topology Change Acknowledge.
TCN	Topology Change Notification.
TCP	Transmission Control Protocol.
TFG	Trabajo Final de Grado.
UAH	Universidad de Alcalá.
UDP	User Datagram Protocol.
URL	Uniform Resource Locator.
VDE	Virtual Distributed Ethernet.
Veth	Virtual Ethernet Device.
VLAN	Virtual Local Area Network.
VXLAN	Virtual Extensible Local Area Network.
WGSI	Web Server Gateway Interface.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá