PROYECTO FIN DE CARRERA

TÍTULO: Scalability study of Guifi.net and mesh networks

TITULACIÓN: Ingeniería de Telecomunicación

AUTOR: Víctor García Fernández

DIRECTOR: Roc Messeguer Pallarès

FECHA: 7 de marzo de 2011

Agradecimientos:

A mi familia, por los valores que me han transmitido y su apoyo incondicional, con el que he contado y en el que me he sustentado siempre.

A Elena, por ayudarme pedaleando en nuestro extraordinario tándem.

A Carlos, por su confianza y paciencia al brindarme la oportunidad de simultanear mi desarrollo profesional y académico.

A mis profesores, a los que me ilusionaron y estimularon desde el primer día, a los que me han acompañado y ayudado a lo largo de mi carrera académica y en especial a Roc Meseguer, que ha tomado parte en todo el recorrido.

A los compañeros con los que he aprendido, de los que he aprendido y de los que, convertidos con el tiempo en amigos, seguiré aprendiendo.

Título: Scalability study of Guifi.net and mesh networks

Autor: Víctor García Fernández

Director: Roc Meseguer Pallarès

Fecha: 7 de marzo de 2011

Resumen

A día de hoy, existen multitud de comunidades y asociaciones de usuarios particulares que crean y desarrollan iniciativas de conectividad ciudadana.

De forma casual o profesionalizada, en pequeños grupos u organizados en estructuras de mayor tamaño, estas agrupaciones utilizan múltiples de tecnologías de comunicación, predominando las abiertas e inalámbricas, como base tecnológica para impulsar sus proyectos.

Los expertos que materializan estas redes utilizan su tiempo libre, un recurso valioso, para el diseño e implementación de estas.
Algunas de las tareas más costosas en tiempo son las relacionadas con la confección, prueba y validación de las configuraciones de red, fases fundamentales en el crecimiento y expansión de estas redes.

En muchos casos, especialmente en lo referente a los protocolos de comunicaciones, es difícil prever cómo se comportará una la red, con una determinada configuración, sometida a las inclemencias del medio radio.

Toda variación no prevista en el diseño, puede cambiar su topología, afectar a la distribución de carga de la red y al rendimiento general, en un entorno en el que realizar modificaciones o ajustes in-situ requiere coordinación entre los miembros de la comunidad y un esfuerzo que comporta desplazamientos, acceso a azoteas y trabajo en exteriores.

Este proyecto reúne la documentación, el software y los procedimientos necesarios para emular redes Ethernet y probar su desempeño bajo condiciones propias de entornos radio.

Virtualizando los nodos y utilizando los mismos sistemas operativos con las mismas aplicaciones, servicios, protocolos y configuraciones que en los escenarios reales, se estudia el impacto provocado por las problemáticas características del medio radio tales como pérdida de datos y variaciones en la velocidad de transferencia o la topología.

Title: Scalability study of Guifi.net and mesh networks

Author: Víctor García Fernández

Director: Roc Meseguer Pallarès

Date: 7 de marzo de 2011

Overview

Nowadays, there are a great number of user communities and associations who create and develop citizen connectivity initiatives.

Whether in a professional or a casual fashion, organized in small groups or larger structures, these associations make use of multiple telecommunication technologies, mainly open and wireless networks, as a technological base to drive their projects.

The expert users who materialize these networks invest their own spare time, a high value resource, to design and implement them.
Some of the most time consuming tasks are the ones related to the elaboration, test and validation of the network configurations, critical phases for the growth and expansion of these networks.

In most cases, especially regarding communication protocols, is difficult to anticipate the behavior of a network setup with a certain configuration and subject to the wireless environment's harshness.

Every unforeseen variation can change the network's topology, affecting the traffic load distribution and network performance, in an environment in which modifications or on-site adjustments require community member's coordination and an important effort involving trips, roof accesses and outdoor labors.

This project gathers the documentation, software and necessary procedures to emulate Ethernet networks and test its performance under specific radio environment characteristics.

Virtualizing the network nodes and using the same operating systems, the same applications, services, protocols and configurations setup on the real scenarios, it's possible to study the impact derived from radio environment such as data loss and network speed or topology variations.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

El objetivo de este proyecto se centra en seleccionar, evaluar y describir las herramientas y procedimientos necesarios para asistir al diseño de redes malladas inalámbricas y el desarrollo de protocolos y servicios de comunicaciones, permitiendo su prueba, estudio y evaluación bajo diversas condiciones propias del medio radio.

Entre los beneficiarios potenciales de la información recogida en esta memoria, se encuentran los usuarios y miembros de las comunidades de particulares que gestionan y mantienen las numerosas redes ciudadanas existentes hoy en día.

Se estima que el máximo exponente y referencia de este movimiento es la asociación Guifi.net, la red libre, abierta y neutral más grande de Catalunya.

1.1 ¿Qué es Guifi.net?

Probablemente, la referencia más apropiada a la que recurrir para tratar de definir Guifi.net sea la Wikipedia, pues comparte con esta algunos de los principios fundamentales.

> *Guifi.net es una red informática de telecomunicaciones libre, abierta y neutral,[…]*
> *Los nodos de la red son de particulares, empresas y administraciones que libremente se conectan a ésta para poder acceder a una auténtica red abierta de telecomunicaciones, y hacerla llegar allí donde haga falta la infraestructura y los contenidos que de otra manera no serían accesibles. [1]*

Esta definición, sintetizada en apenas un párrafo, da una idea inicial de las líneas generales por las que se guía el proyecto.

Guifi.net es una red abierta, cooperativa y de libre acceso, características que nos recuerdan a los fundamentos de la “Era 2.0”, que hace unos años invadió el mundo de las comunicaciones convirtiendo a los usuarios de espectadores a participantes en la creación y difusión de contenidos.

No es la primera vez que surge a la luz un proyecto similar. Desde la popularización de las tecnologías de red inalámbrica basadas en el IEEE 802.11, ha habido multitud de iniciativas parecidas que, con mayor o menor éxito, han intentado conectar a comunidades o municipios con Internet.

En unos casos la tecnología, en otros la organización o la legislación, han constituido barreras insuperables para algunos de estos proyectos, haciendo prosperar sólo a los modelos más estructurados, equilibrados y mejor dirigidos.

Entre estos supervivientes se encuentra Guifi.net, un proyecto que parte de una idea utópica y ha sido respaldado e impulsado por una fundación que ha sabido reunir y organizar los recursos tecnológicos y humanos necesarios para convertirla en realidad.

En sus seis años de vida, ha logrado rebasar la barrera de los diez mil nodos, cifras que la convierten en prácticamente una adulta en el mundo de las redes inalámbricas, lo que, unido a su presencia en los principales organismos nacionales e internacionales, le otorga una entidad de la que carecen la mayoría de estas iniciativas.

Sus actividades están notificadas ante la Comisión del Mercado de las Telecomunicaciones y se encuentran adscritas a su arbitrio, al igual que cualquier operador nacional de acceso a internet. Además, su fundación es miembro del RIPE como LIR, Local Internet Registry, como lo son todos los Sistemas Autónomos de los operadores conectados a Internet.

1.2 Estructura de la Memoria

A lo largo de los siguientes capítulos se abordan los diferentes elementos y tecnologías que conforman la solución, desde los protocolos utilizados, métodos de virtualización y emulación, construcción y configuración del software y modelación del entorno radio.

Finalmente, el último capítulo recoge ejemplos con los resultados y conclusiones de dos de los escenarios desarrollados durante el proyecto.

2 Protocolos de Encaminamiento

2.1 BATMAN, Better Approach to Mobile Adhoc Networks

Se trata de un protocolo de encaminamiento relativamente reciente y que continúa bajo desarrollo por la comunidad Freifunk[1] con el objeto de reemplazar a OLSR.

La principal innovación que introduce B.A.T.M.A.N. es, como su propio nombre indica, un diseño especialmente enfocado a las redes móviles inalámbricas.
Estas redes se caracterizan por el dinamismo de su topología, que cambia con frecuencia debido a la adición y desaparición de nodos que acontecen con la variación en propagación de las señales radio.

Es precisamente la baja fiabilidad del medio aéreo lo que dificulta la convergencia y el óptimo funcionamiento de los protocolos de encaminamiento tradicionales.

Actualmente, OLSR es el protocolo más utilizado en este tipo de escenarios y, debido a que está basado en un algoritmo de estado-enlace, cada nodo necesita conocer la topología completa de la red para poder elaborar las tablas de encaminamiento.

Resulta evidente que, cuando hablamos de dispositivos inalámbricos de bajo consumo y baja potencia computacional (como pueden ser los routers mesh en un despliegue urbano, por ejemplo), este modelo resulta poco escalable ya que obliga a mantener información sobre la topología completa de una red en constante cambio y, por consiguiente, a encaminar los datos basándose en una tabla de encaminamiento altamente volátil y poco fiable.

B.A.T.M.A.N. pretende resolver este aspecto mediante la distribución del conocimiento sobre la topología de la red, de manera que cada nodo almacena y mantiene solamente la información relativa al siguiente salto (*next hop*) para cada nodo destino. De este modo, los nodos no almacenan las rutas completas extremo a extremo y los cambios en la topología sólo afectan localmente (a los nodos cercanos) en lugar de globalmente. Junto a esta información, también se almacenan datos referentes a las características de cada enlace en términos de fiabilidad, estabilidad, simetría, etc. Concretamente, la fiabilidad se caracteriza mediante el cálculo de la Calidad de transmisión (TQ, Transmission Quality) como el cociente entre la Calidad de Eco (EQ, Echo Quality), que combina la ida y la vuelta de un mismo paquete de información, y la Calidad de Recepción (RQ, ReceptionQuality).

De el mismo modo que RIP, B.A.T.M.A.N., se construye sobre UDP/IP y las actualizaciones se difunden mediante broadcast de paquetes “Hello” a los

[1] Freifunk.net (Radio Libre, en almemán) es una iniciativa abierta y no comercial para el desarrollo e impulso de las redes inalámbricas abiertas en Alemania. http://www.freifunk.net

nodos vecinos. Este tipo de mensajes se denominan OGM (Originator Messages) y tienen un tamaño total de tan sólo 52 Bytes, cabeceras incluidas.

2.2 BMX: B.A.T.M.A.N. Experimental

2.2.1 Introducción

El protocolo BMX guarda una estrecha relación con su predecesor B.A.T.M.A.N., de hecho, BMX nació como una rama paralela de desarrollo e implementación de nuevas funcionalidades y conceptos, destinada a mejorar el algoritmo inicial.

La primera implementación de BatMan-eXperimental, o BMX como se le conoce actualmente, nace de la revisión 0.3 de batmand, la aplicación de usuario de B.A.T.M.A.N. y las mejoras que ofrece, van enfocadas a dar una mayor profundidad de configuración desde la herramienta de línea de comandos bmxd.

A día de hoy el proyecto ha evolucionado hasta convertirse en una entidad propia y cuenta con dos versiones, la estable (bmx-0.3, la versión utilizada en este proyecto) y la de desarrollo (BMX6).

2.2.2 Configuración

Existen multitud de parámetros que permiten configurar y ajustar el algoritmo de BMX para que se comporte de una forma u otra en función de las características de la red en la que opera.

Por lo general, estos parámetros definen probabilidades, umbrales y ventanas de trabajo que determinan la permisividad de BMX frente a las pérdidas o irregularidades de la red a la hora de calificar y clasificar los enlaces y vecinos en las listas de ranking confeccionadas para definir la mejor ruta hacia un destino.

El Anexo I recoge los parámetros más relevantes de la configuración de BMX. Concretamente los más interesantes desde el punto de vista del control sobre la tolerancia del protocolo a los cambios en el medio y la clasificación de calidad de los enlaces.

A continuación se incluye la configuración de la publicación de información HNA, frecuentemente utilizada en troncales mesh para anunciar las redes de acceso.

Anuncios HNA

Los HNA, o Host and Network Association, es un método utilizado por OLSR y BMX para diseminar rutas hacia redes, de la misma forma que los mensajes OGM anuncian la interfaz de host desde la que se envían.

Por lo general, cuando un router que implementa el protocolo BMX tiene varias interfaces de red conectadas a otros routers, BMX envía mensajes OGM a través de estas, publicando sus direcciones IP.

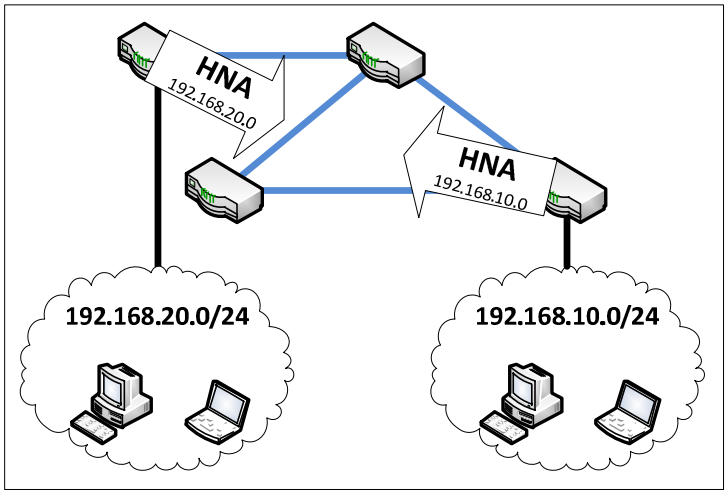


Figura 2-1. Mensajes HNA en una red de routers BMX.

Como puede observarse en la figura anterior, si el mismo nodo o router está conectado a otras redes que no implementan BMX (representadas con enlaces de color negro) BMX envía mensajes HNA junto con los OGM a través de las interfaces BMX (representadas con enlaces azules) con el fin de dar a conocer sus redes asociadas.

2.2.3 RTQ, medida de calidad bidireccional

BMX denomina RTQ al parámetro Round Trip Quality, que mide la calidad combinada de ambos sentidos, emisión y recepción y es equivalente a la Echo Quality de B.A.T.M.A.N.
Los valores del parámetro están comprendidos entre 0 y 100.

Por su característica, RTQ constituye el parámetro de calidad más relevante en BMX, ya que este protocolo sólo tiene en cuenta la señalización recibida a través de enlaces bidireccionales.

Puede encontrarse información ampliada en el Anexo A1.

Debido a este factor en particular y teniendo en cuenta que es necesario contemplar ambos sentidos de la comunicación para caracterizar un enlace entre dos nodos, las pruebas prácticas realizadas en este proyecto tomarán como referencia y medida comparativa el parámetro RTQ.

3 User-Mode Linux

3.1 Introducción

User-Mode Linux, de ahora en adelante UML, es una implementación de código abierto para la virtualización de sistemas que permite ejecutar múltiples instancias del Kernel de Linux (sistema huésped) dentro del espacio de usuario de otro Kernel de Linux (sistema anfitrión).

Recibe su nombre debido a que se ejecuta en el *userspace* o User-Mode de Linux, ya que es iniciado por el usuario como si de una aplicación se tratara.

3.2 Funcionamiento

Mediante este sistema, se pueden ejecutar varias máquinas virtuales gobernadas por la misma o diversas versiones del Kernel de Linux UML, todas ellas hospedadas dentro de un mismo sistema Linux, como si de aplicaciones de usuario se tratara.

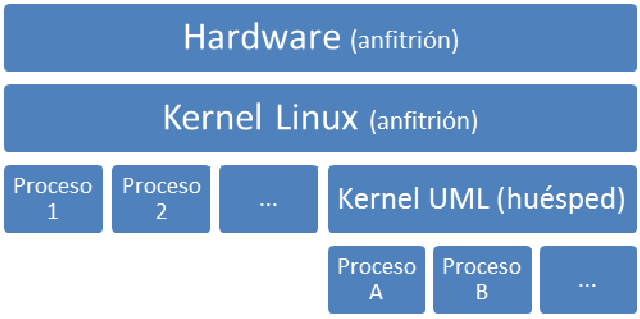


Figura 3-1. Representación de los niveles de ejecución en UML

Partiendo de la estructura funcional básica descrita en la Figura 4-1, podemos particularizar el uso de UML mostrando un ejemplo característico del uso de esta tecnología.

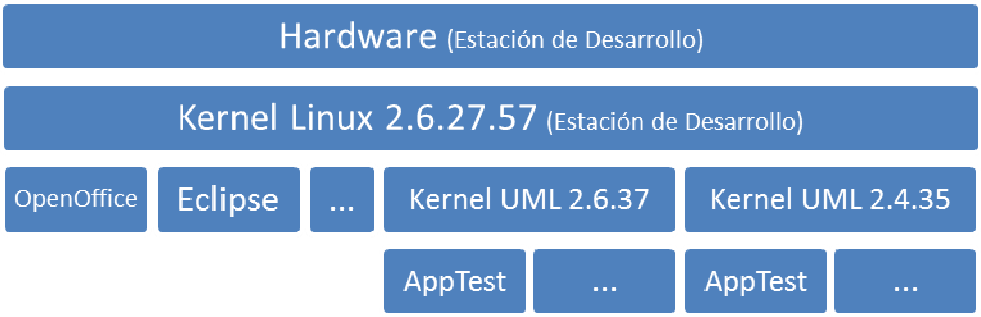


Figura 3-2. Ejemplo de ejecución con diferentes versiones de Kernel Linux

En entornos de desarrollo y prueba de aplicaciones, al virtualización juega un papel fundamental a la hora de evaluar el comportamiento y la compatibilidad de las aplicaciones en construcción.

El ejemplo de la figura 4-2. Ilustra un caso en el que un PC, que actúa de Estación de Desarrollo, ejecuta varias aplicaciones de usuario tales como OpenOffice, o el entorno de desarrollo Eclipse, además de dos instancias de Kernel UML.

Gracias a la virtualización proporcionada por UML, el desarrollador es capaz de probar en su propia estación el comportamiento de la aplicación que está desarrollando (AppTest) con dos versiones de Kernel, una muy reciente e inestable (2.6.37) y otra antigua (2.4.35), además de en la de su propio sistema.

3.3 Linux Kernel

Para poder ejecutar un Kernel de Linux en UML, es necesario configurarlo y compilarlo para esta función, ya que en este modo, el Kernel no interactuará directamente con los recursos hardware del anfitrión, sino que lo hará a través del Kernel Linux del sistema en el que se ejecuta.

UML es significativamente diferente del resto de sistemas virtualización, tanto de código abierto como comerciales, ya que se presenta como una mera plataforma ligera de ejecución virtualizada, sin gestión ni otras funcionalidades avanzadas. Es precisamente su sencillez lo que le permite destacar en cuanto al consumo de recursos.

Se estima que UML puede introducir una ralentización de, como máximo, el 20% respecto de un sistema “real”. Esto es factible gracias a que, al contrario que la mayoría de sistemas de virtualización, el Kernel UML huésped interactúa con el hardware a través del Kernel del sistema anfitrión.

Por el contrario, la mayoría de sistemas de virtualización añaden capas de abstracción del hardware, sumando middlewares que añaden complejidad y consumo de recursos a la estructura.

3.4 Sistema de archivos

En el apartado del almacenamiento, UML utiliza el driver UBD (User Block Device) para tratar archivos como si se trataran de dispositivos de bloques, emulando una unidad de disco duro , y permitiendo complementar el sistema virtualizado con Kernel UML con una imagen del sistema de archivos Linux huésped empaquetado en un sólo fichero alojado en el sistema de archivos anfitrión.

Además de permitir la lectura y escritura en esta misma imagen del sistema de ficheros del huésped, UML también contempla la utilización de un método de escritura separada llamado COW (CopyOnWrite) y que permite que varios

huéspedes virtualizados compartan el mismo sistema de archivos sin peligro de corromperlo, pues la escritura se realiza en ficheros independientes.
COW funciona generando un segundo archivo, vinculado a la imagen del sistema de ficheros, donde se escribe todos los cambios que el sistema huésped realiza sobre éste.

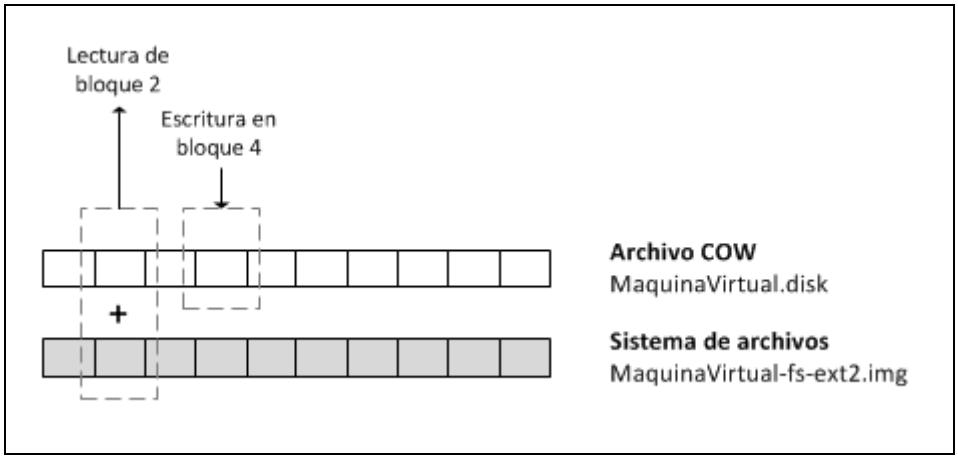


Figura 3-3. Lectura y Escritura con dispositivos de bloques en UML

El uso de ficheros COW, permite la construcción de escenarios virtualizados con múltiples huéspedes que comparten un mismo Kernel UML y un mismo sistema de archivos, pero que a la vez son independientes, escribiendo los cambios en sus ficheros COW particulares.

En el Anexo II se recoge una comparativa entre los principales sistemas de virtualización existentes, justificando la utilización de UML.

4 NetKit

4.1 Introducción

NetKit es un entorno basado en User-Mode Linux y desarrollado por la Universidad de Roma, destinado a la creación de entornos virtuales en red, que permitan realizar experimentos de forma sencilla y sobre hardware de uso doméstico y bajo coste.

Su objetivo es el de permitir la definición y ejecución de múltiples nodos virtuales de red y emular la comunicación entre ellos, soportando la formación de topologías complejas.

Emular una red Ethernet con NetKit, es cuestión de describir la topología a nivel de enlace para cada nodo de la red, así como configurar cada uno de ellos para que desempeñe su función en la red. La configuración de los nodos virtuales se realiza de la misma forma que en una red real, ya que los nodos de NetKit son máquinas virtuales basadas en el Kernel de Linux UML.

Netkit se encarga de iniciar cada uno de los nodos virtuales de la red y unirlos siguiendo la topología descrita, emulando así la red diseñada.

4.2 Emulación vs Simulación

Los sistemas de Emulación y Simulación, ponen a disposición del usuario la posibilidad de crear entornos de virtuales que pueden ser trabajados, estudiados y analizados con fines de experimentación, prueba, medidas, etc.

Éstos conforman una herramienta indispensable a la hora de ofrecer un respaldo para la práctica experimental en investigación, desarrollo, formación o integración de sistemas telemáticos y de información.

Cuando hablamos de virtualización de redes, es importante destacar que este tipo de herramientas se dividen en dos grandes grupos cuya diferencia se centra, básicamente, en el enfoque de las prestaciones:

- **Sistemas de simulación**, son los que tienen como objetivo reproducir o predecir las prestaciones de un sistema "real" en términos de latencia, velocidad de transmisión, etc.

El ejemplo más representativo de simulación de redes es el proyecto de código abierto Network Simulator[2].

- **Sistemas de emulación**, son los que se centran en reproducir fielmente las funcionalidades, configuraciones, arquitecturas y protocolos de los sistemas "reales", dejando en un segundo plano el rendimiento y las prestaciones de red.

El objetivo último de este tipo de herramientas es que las mismas configuraciones, protocolos y funcionalidades probadas en los entornos virtuales sean trasladables de forma directa a entornos reales.

NetKit es, debido a su fuerte dependencia de UML, un software de emulación de redes ya que al estar basado en virtualización de Kernel, sus nodos o máquinas virtuales son representaciones exactas de los sistemas Linux que podemos encontrar en entornos reales de producción, con sus mismas funcionalidades, configuraciones, arquitecturas y protocolos de comunicación.

4.3 Arquitectura

Los nodos virtuales que NetKit permite crear son, en realidad, imágenes del Kernel y el Sistema de archivos de Linux, ejecutados en UML y conectados entre sí a través de "Hubs virtuales" como dominios de colisión.

[2] NS, Network Simulator http://www.isi.edu/nsnam/ns/

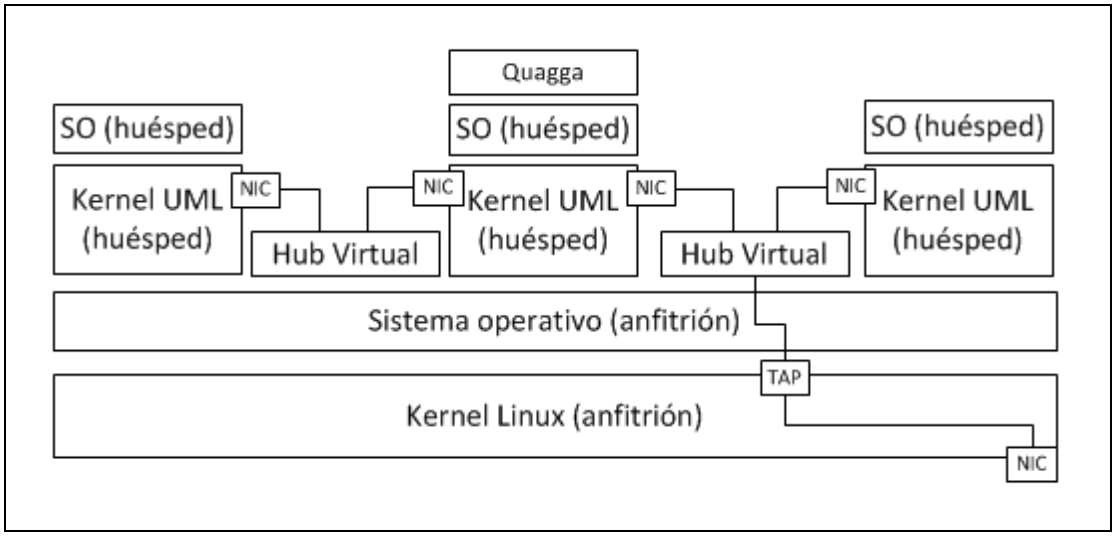


Figura 4-1. Estructura multicapa de NetKit y UML

Estos dominios de colisión pueden estar conectados con el mundo exterior a través de interfaces virtuales "tap" que, creadas en el Kernel anfitrión, permiten a un nodo (o a toda la red virtual de NetKit) conectarse a redes "reales", a través de los recursos de red del anfitrión.

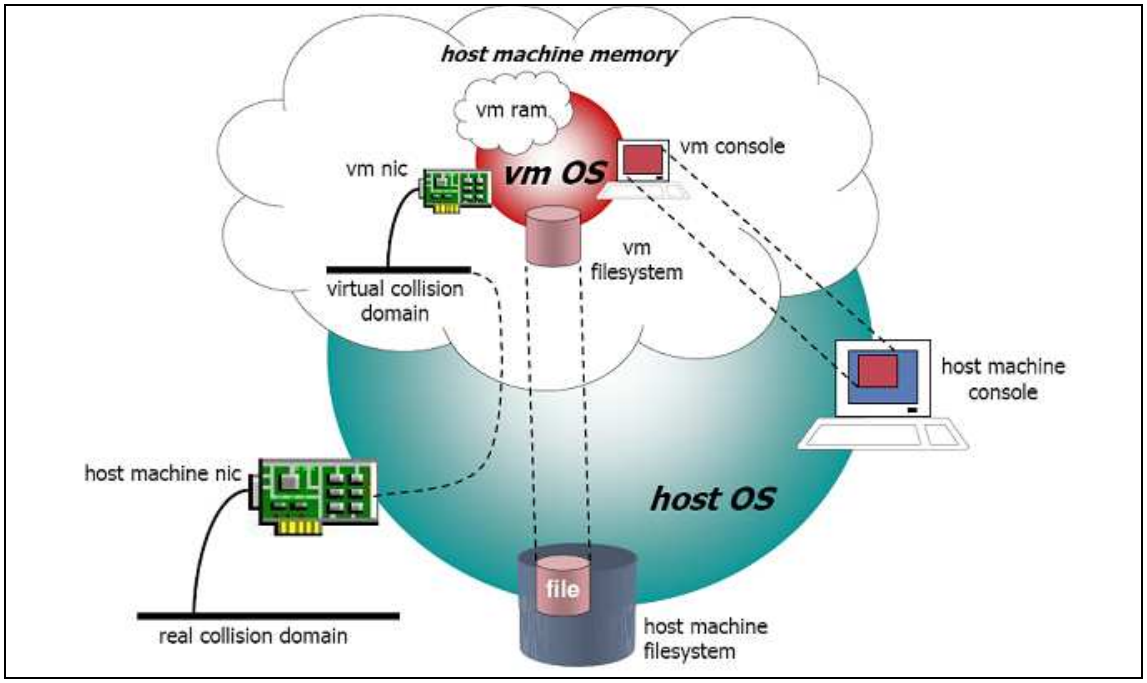


Figura 4-2. Arquitectura de virtualización NetKit. (Roma Tre University)

Cada una de las máquinas virtuales cuenta con acceso a unos recursos básicos:

- Consola (ventana de terminal) a través de la cual se accede al intérprete de comandos del sistema huésped.

- Memoria RAM (NetKit permite asignar una porción de la memoria RAM del sistema anfitrión).
- Sistema de archivos (almacenado en un archivo de imagen en el anfitrión)
- Una o más interfaces de red.

En el momento de redacción de este proyecto, NetKit se encuentra en la versión 2.7 y ofrece un Kernel UML y sistema de archivos basado en la distribución Debian:

- Netkit kernel version 2.8 (Basado en el Kernel de Linux 2.6.26.5 UML)
- Netkit filesystem versión 5.1

Los paquetes instalados en el sistema de archivos pueden consultarse en el sitio Wiki de NetKit[3].

El Anexo III recoge un resumen de las tecnologías compatibles con NetKit.

4.4 Aplicaciones

Dado su origen, NetKit nace en un marco académico con una fuerte proyección como sistema vehicular para la docencia y la investigación.

No es de extrañar pues, que existan multitud de escenarios pre configurados especialmente diseñados como apoyo práctico y organizados por temática en diferentes “laboratorios virtuales” que cubren, desde temáticas básicos como encaminamiento estático o RIP, aplicaciones como DNS o Correo electrónico o avanzados como MPLS, Sistemas Autónomos ó BGP .

Los laboratorios oficiales han sido creados por profesores y colaboradores de la Universidad de Roma y se ofrecen, junto con el material docente que acompaña a las clases teóricas, en la sección de descarga del sitio Wiki de Netkit[4].

[3] http://wiki.netkit.org/download/netkit-filesystem/installed-packages-i386-F5.1
[4] Laboratorios Oficiales de NetKit. http://wiki.netkit.org/index.php/Labs_Official

5 OpenWRT

5.1 Introducción

OpenWRT es una distribución de Linux originalmente concebida para dispositivos embebidos. Concretamente fue diseñado para su utilización en enrutadores domésticos Linksys WRT54G, basados en un firmware de código abierto.

El fabricante Linksys, cumpliendo con las directivas de la licencia de GNU GPL, liberó el código fuente del dispositivo, convirtió al WRT54G en la primera plataforma de routers de bajo coste compatibles con Open Source.

Figura 5-1. Linksys WRT54G. Plataforma hardware inicial del proyecto OpenWRT.

En pocos años, este dispositivo fue acogido como plataforma base de numerosos proyectos de comunidades inalámbricas que añadieron las más avanzadas funcionalidades de red disponibles en los Kernel de Linux más recientes con el fin de potenciar sus proyectos con el uso de estos nodos.

Una de las distribuciones más destacadas y con más soporte y desarrollo hasta la fecha es OpenWRT.

OpenWRT cuenta con herramientas de construcción abiertas al público que permiten portar este sistema operativo a multitud de sistemas, como enrutadores comerciales domésticos de varios fabricantes, a x86 y a User Mode Linux, entre otras.

5.2 Preparación del entorno

Requisitos previos

Para poder compilar y construir OpenWRT para las plataformas objetivo que se utilizan en este proyecto, es necesario instalar los siguientes paquetes de software:

asciidoc, binutils, bzip2, fastjar, flex, g++, gcc, autoconf, gawk, bison, libgtk2.0-dev, intltool, jikes, zlib1g-dev, libssl-dev, patch, perl-modules, rsync, ruby, sdcc, unzip, wget, sdcc-nf, gettext, xsltproc, zlib1g-dev

Si contamos con una distribución de Linux basada en Ubuntu, bastará con ejecutar la siguiente orden en consola, que incluye los grupos de paquetes necesarios:

```
$>sudo apt-get install build-essential subversion libncurses5-dev zlib1g-dev gawk flex
```

Versiones de OpenWRT

El primer paso a realizar para construir OpenWRT es descargar las fuentes de cualquiera de sus versiones.

Actualmente existen 4 versiones vigentes:

- **Kamikaze 8.09**: Penúltima y más longeva versión estable de OpenWRT. Data de septiembre de 2008, aunque ha recibido mejoras a principios de 2009.
  Basada en el Kernel de Linux 2.6.25.

- **Backfire 10.03**: Última versión estable.
  Data de Abril de 2010 y está basada en el Kernel de Linux 2.6.32.

- **Versión Trunk**: Es la versión más reciente, se utiliza con fines de desarrollo e incorpora las últimas mejoras y correcciones publicadas. La versión actual está basada en el Kernel de Linux 2.6.32.25

Las tres están disponibles a través del repositorio subversion de OpenWRT y la tercera está disponible también a través de GIT.
En el desarrollo de este proyecto utilizaremos la versión Trunk ya que es la que incorpora el Kernel de Linux más reciente y se comporta mejor en conjunción con NetKit.

Descarga del código fuente

Para descargar el código del repositorio utilizaremos el comando svn:

```
$>svn co svn://svn.openwrt.org/openwrt/trunk/
```

Este comando descargará los aproximadamente 64 MBytes que ocupa el entorno de construcción de OpenWRT en el directorio actual.

El paquete constructor de OpenWRT recibe el nombre de “Buildroot” y, básicamente, es un conjunto de archivos Makefile que configuran y compilan el software de OpenWRT con las opciones adecuadas para cada sistema.

Buildroot incluye también los paquetes necesarios para el uso de la herramienta de compilación cruzada o cross-compilación (entre ellos gcc, binutils y uClibc).

Normalmente hay un archivo Makefile por cada paquete de software y estos se dividen en tres secciones, alojadas en la raíz del directorio donde se descarga Buildroot (PFC, en este caso)

- **Packages:** Contiene los Makefile y archivos asociados para todas las herramientas que Buildrootcross-compilará para la arquitectura objetivo. Cada herramienta está alojada en su propio directorio.

  En este directorio deben añadirse las fuentes de los paquetes software que se desee compilar para su integración en el sistema de archivos objetivo o bien para la construcción de los paquetes de software instalables ipk.
- **Toolchain:** Contiene los Makefile y archivos del set de herramientas que componen el software de compilación cruzada: binutils, ccache, gcc, gdb, kernel-headersyuClibc.

- **Target:** Contiene los Makefile y archivos necesarios para generar la imagen del sistema de archivos y el Kernel de Linux para las diferentes arquitecturas soportadas, entre ellas UML.

El sistema Buildroot estándar incluye solamente una selección limitada de paquetes software. Para obtener las fuentes de todos los paquetes disponibles en OpenWRT se deben descargar los paquetes adicionales (21 MBytes):

```
$>svn co svn://svn.openwrt.org/openwrt/packages
```

El repositorio de paquetes software extra incluye, entre otros, protocolos de encaminamiento como bmx y olsr, además de otras muchas aplicaciones y servicios como OpenVPN, asterisk y multitud de otros proyectos de código abierto orientados a redes, multimedia, etc.

En el sitio web de OpenWRT para desarrolladores pueden consultarse los paquetes software incluidos en el Buildroot de serie[5], así como los extras[6].

[5] https://dev.openwrt.org/browser/trunk/package
[6] https://dev.openwrt.org/browser/packages

6 Herramientas Modeladoras

6.1 Introducción

En este capítulo se abordarán las dos herramientas principales utilizadas en este proyecto para la emulación de las condiciones del medio radio en NetKit. Se trata de Netem y Netfilter.

Debido a las limitaciones de NetKit en cuanto a la creación de dominios de colisión y a las caprichosas topologías que pueden darse en entorno radio, se hace necesaria la creación de un modelo de emulación de red lo suficientemente flexible como para permitir reproducirlas con la mayor similitud posible.

Como se ha visto en el capítulo de NetKit, las opciones que éste ofrece en cuanto a simulación de redes, se limitan a los dominios de colisión en forma de hubs virtuales, lo que resulta insuficiente a la hora de reproducir las condiciones de entornos reales y complejos.

6.2 Casuística

Existen diversos factores, no emulables mediante NetKit, que resultan básicos a la hora de reproducir virtualmente redes inalámbricas.

A continuación se enumeran los factores más básicos, agrupados en dos categorías:

- Relativos al medio
  - Atenuación
    - Distancia
    - Interferencias
    - Obstáculos

- Relativos a los equipos
  - Antena
    - Orientación
    - Direccionalidad
    - Ganancia
  - Electrónica
    - Potencia de transmisión

La mayor parte de ellos son emulables mediante Netem y Netfilter, aunque los factores de característica exclusiva de las interfaces inalámbricas como la ganancia de las antenas y la potencia de transmisión no son directamente emulables en redes Ethernet ya sean reales o virtuales.

A continuación se introducen las herramientas software utilizadas.

6.3 Netem, Emulación de red.

Netem es un conjunto de herramientas que permiten emular las condiciones propias de una gran variedad de redes mediante técnicas que reproducen retardo variable, pérdida, duplicación y desorden de los paquetes de datos.

El emulador de red Netem se encuentra incluido de serie en el kernel 2.6 de Linux y sus funcionalidades se controlan mediante el comando 'tc', incluido en el paquete iproute2.

En el caso particular de OpenWRT, 'tc' se ofrece como un paquete de software independiente y requiere el módulo de Kernel de priorización de tráfico 'kmod-sched' (trafficscheduling).

TC, Control de Tráfico en Linux

TC, o Traffic Control, es el comando principal del paquete Netem y sirve para controlar el sistema de colas que intervienen en la recepción y emisión de paquetes en un enrutador basado en Linux.

Mediante esta herramienta, que se ejecuta con permisos de root,se programa el tratamiento de los paquetes:

- **De ingreso**: Cuando y a qué velocidad de transmisión deben aceptarse a través de las interfaces de entrada.
- **De egreso**: Qué paquetes transmitir, en qué orden y a qué velocidad de trasmisión deben emitirse a través de las interfaces de salida.

En la gran mayoría de casos, el control de tráfico consiste en una cola simple que recoge los paquetes recibidos y los despacha a medida que el hardware libera recursos para procesarlos. Este modelo se conoce como FIFO (First in, Firstout) y representa la política más básica para el reenvío de paquetes, transmitiéndolos en el orden por el que llegaron a la interfaz de entrada.

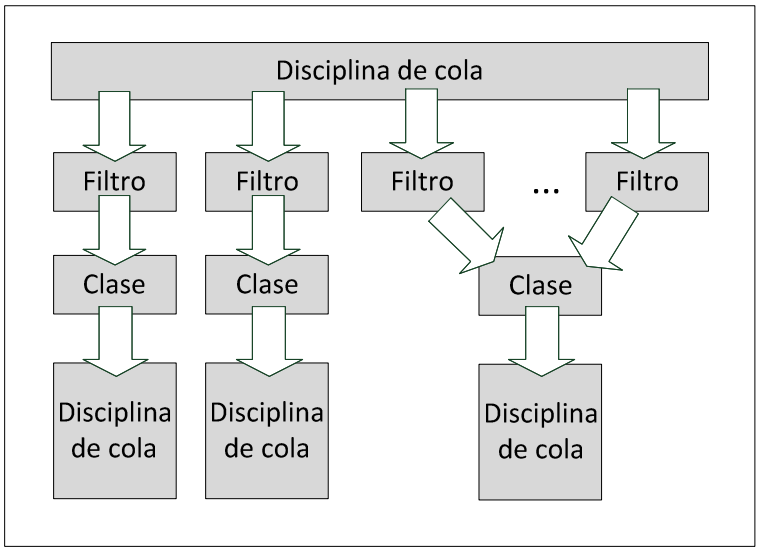


Figura 6-1. Esquema de control de tráfico en Linux

El comando 'tc' permite tanto configurar como mostrar la configuración de control de tráfico en el Kernel de Linux en base a tres herramientas:

- **Disciplinas de Cola (qdisc)**

  Qdisc es la abreviatura de 'queuing discipline', disciplina de colas, y representa la base del control de tráfico. Cuando el Kernel de Linux necesita enviar un paquete a una interfaz, éste es insertado en la cola y, por lo tanto, queda sujeto a la disciplina de cola configurada para dicha interfaz.

- **Clases**

  Las clases permiten crear estructuras con las disciplinas de cola con el fin de dotar de mayor complejidad y granularidad a la clasificación del tráfico.

- **Filtros**

  Los filtros se utilizan para seleccionar los paquetes de datos que llegan a una interfaz de red y clasificarlos dentro de las clases disponibles en base a una serie de criterios programados.

Como podemos observar, tc permite crear estructuras muy completas de filtrado, clasificación y encolado de paquetes con el fin de adecuar o reducir el tráfico a las condiciones del receptor o de la red por la que se va a transmitir mediante la retención o el descarte de tramas.

En este proyecto nos centraremos concretamente en un tipo particular de cola que permite emular pérdidas. Se invoca con el comando tc de la siguiente forma:

```
#>tc qdisc add dev eth0 root netem loss 30%
```

De este modo, se añade una nueva disciplina de cola a la interfaz eth0 que emulará una pérdida del 30% de los paquetes.

El Kernel de Linux descartará aleatoriamente 5 de cada 100 paquetes de la cola, creando este efecto, afectando a todas las aplicaciones de usuario y servicios del sistema que utilicen dicha interfaz de red.

Existe un parámetro adicional para añadir una correlación al porcentaje de pérdidas, por ejemplo:

```
#>tc qdisc add dev eth0 root netem loss 5% 75%
```

De este modo, se modifica la función aleatoria, añadiendo en este caso un 75% de dependencia del resultado anterior.

En el caso de ejemplo, la probabilidad de perder la trama enésima es en un 25% resultado de la función aleatoria “random” + un 75% del resultado anterior.

$$Pperd_n = Random(0,05) * 0,25 + Pperd_{n-1} * 0,75$$

Este parámetro es especialmente útil si queremos simular ráfagas de pérdidas en lugar de pérdida aleatoria de tramas, acercando el resultado todavía más a la realidad del entorno radio.

En el capítulo 6.5, se muestra la programación de colas mediante tc con más ejemplos prácticos.

Además de pérdidas, Netem permite emular retardos, duplicación, corrupción y desorden de paquetes y reducir la velocidad de transferencia del enlace mediante el uso de varios mecanismos de control de flujo.

6.4 Netfilter, Filtrado de paquetes.

Netfilter es otro set de herramientas disponibles en el Kernel de Linux, como Netem, desde la versión 2.4. Sus funcionalidades giran en torno a la interceptación y filtrado de paquetes de red.

Los comandos de Netfilter más utilizados son ebtables e iptables, representadas en la figura 6-2.

Ambas aplicaciones de usuario que permiten, entre otras funciones, configurar el filtrado de paquetes y la traducción de direcciones (NAT) en el Kernel de Linux para el nivel de enlace y red, respectivamente.

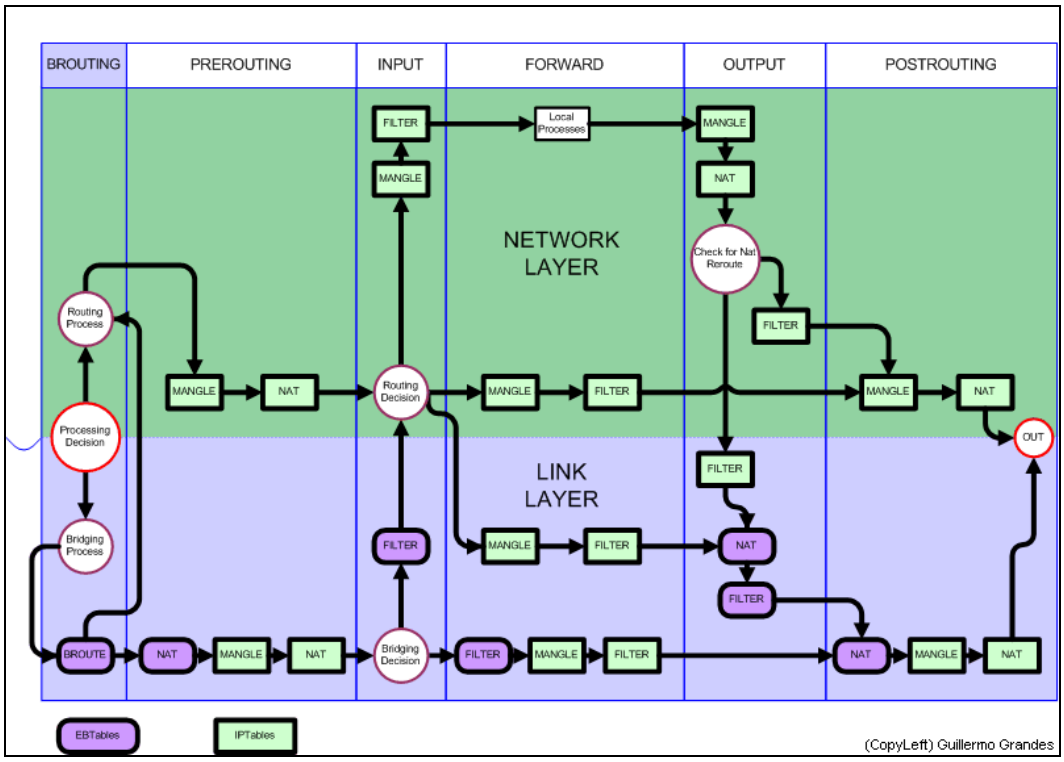


Figura 6-2. Netfilter ebtables e iptables (Wikipedia)

En este proyecto, nos centraremos en iptables y más concretamente en sus funciones de filtrado para IPv4.

Iptables,

El comando iptables invoca la aplicación de gestión de Netfilter y, del mismo modo que tc, requiere permisos de root para su ejecución.

La sintaxis detallada del comando iptables está documentada en su página[7] del manual de Linux.

El diagrama de la Figura 6-2, refleja de forma muy básica el funcionamiento de netfilter y su interacción con los conjuntos de reglas programadas mediante iptables.

Netfilter cuenta con 3 tablas gestionables mediante regles, que responden a los criterios establecidos por la regles configuradas mediante Iptables.

- Mangle: Orientada a la modificación de los paquetes de datos.
- NAT: Orientada a la reescritura de direcciones IP o puertos.
- Filter: Orientada al filtrado de paquete.

En este proyecto nos centraremos en la operativa básica de entrada, salida y reenvío de la tabla de filtrado, correspondiente a las cadenes de INPUT, OUTPUT y FORWARD, respectivamente.

[7] Página del Manual de Linux en la web para Iptables: http://linux.die.net/man/8/iptables

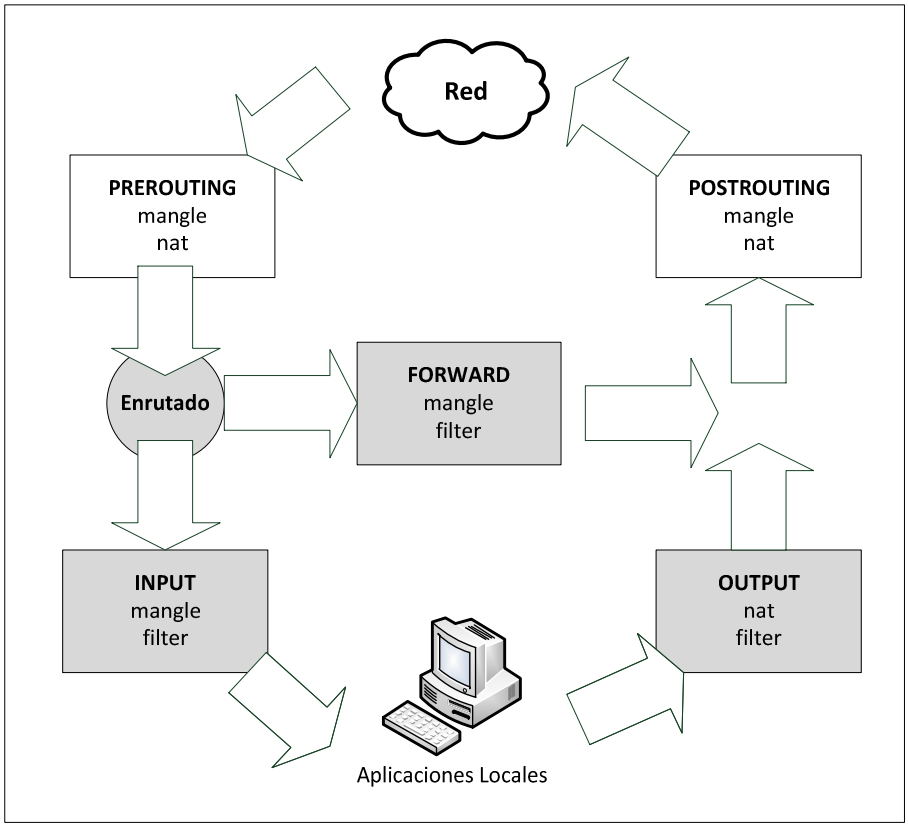


Figura 6-3. Flujo de enrutado y procesos de Netfilter IPtables

Tal y como refleja el diagrama, en primera instancia tras la recepción de un paquete de datos, el kernel de Linux inspecciona las cabeceras del paquete para determinar su destino y discernir si va dirigido al sistema local o a un tercero.

Los datagramas dirigidos al sistema local son procesados con las reglas INPUT. Del mismo modo, los datagramas generados por las aplicaciones del sistema local pasarán el filtrado de las reglas OUTPUT.

Finalmente las tramas que no van dirigidas al sistema local y que deben ser encaminadas, pasan el filtrado de las reglas de reenvío FORWARD.

En la siguiente sección, se abordará el filtrado selectivo con iptables con algunos ejemplos, destacando su utilidad en el emulado de redes inalámbricas con Netkit.

6.5 Emulación de redes inalámbricas

6.5.1 Alineación

Debido a la orientación y a la direccionalidad de la antena equipada en cada interfaz, las topologías formadas pueden resultar complejas y difícilmente reproducibles mediante hubs virtuales.

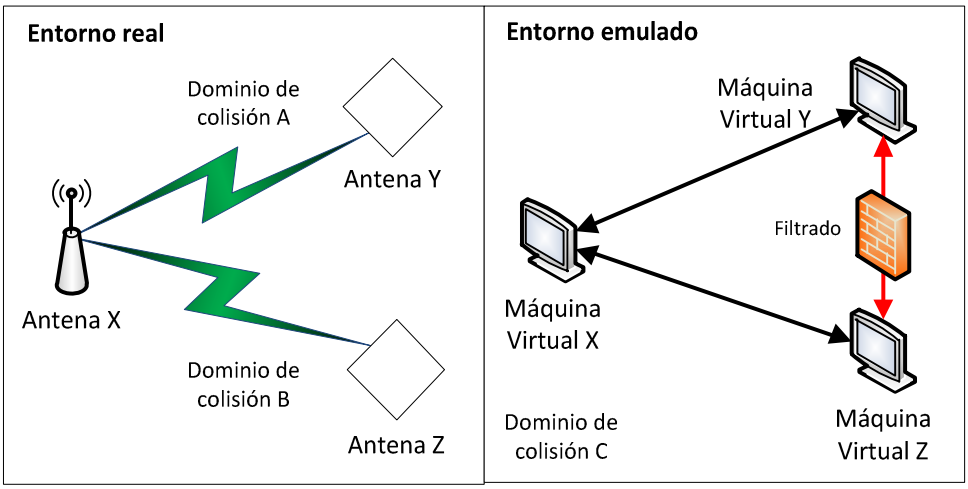


Figura 6-4. Ejemplo de topología inalámbrica punto a multipunto

En el ejemplo representado en la figura 6-4, la interfaz de la antena X (equipada co una antena omnidireccional) recibe señal de las antenas Z e Y.

Estas últimas, debido a su característica directiva y orientación física, no reciben señal la una de la otra, pero sí que reciben de la antena X.

En NetKit no soporta emular un escenario tan sencillo como este, ya que cada interfaz de red puede formar parte de un sólo dominio de colisión, lo que imposibilita asociar la antena X a los dominios de colisión A y B.

Como alternativa, se propone la creación de un único dominio de colisión, C, y filtrar el tráfico entre las interfaces que, en el escenario real, no tendrían conectividad.

Este filtrado puede llevarse a cabo mediante el set de herramientas Netfilter, concretamente, el comando iptables.

En la siguiente figura, se muestra un escenario virtual de ejemplo con tres nodos y dos enlaces inalámbricos emulados.

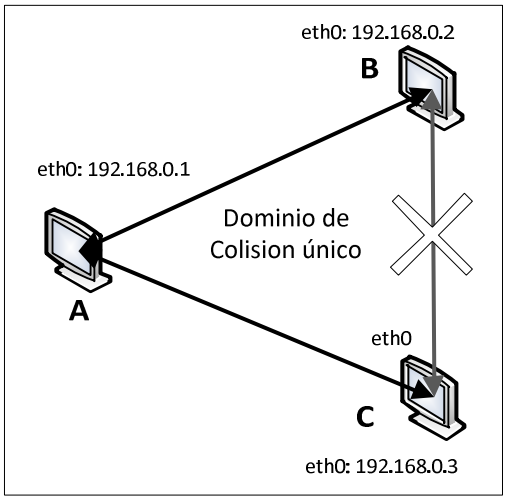


Figura 6-5. Esquema punto a multipunto.

La utilización de un dominio de colisión único provoca que las aplicaciones, servicios y protocolos de comunicaciones que utilicen mensajes broadcast o multicast se transmitan a través de enlaces virtuales que, como el enlace B-C en la figura 6-2, no existen en la topología que se pretende emular.
Utilizando el comando iptables, pueden programarse reglas de Netfilter para descartar el tráfico directo entre los nodos B y C.
Para ello se programan sendas reglas en los terminales de las máquinas virtuales B y C:

```
#B>iptables –A INPUT –s 192.168.0.3 –j DROP
```

```
#C>iptables –A INPUT –s 192.168.0.2 –j DROP
```

A continuación, se arranca el servicio del protocolo BMX en las tres máquinas virtuales

```
#> bmxd dev=eth0
```

BMX elaborará un mapa de la red en cada nodo. A continuación mostramos las tablas elaboradas por BMX en cada máquina virtual antes y después de aplicar las reglas de Netfilter con iptables.

NodoA (eth0: 192.168.0.1)

Previa a la programación de las reglas de filtrado en las máquinas virtuales B y C:

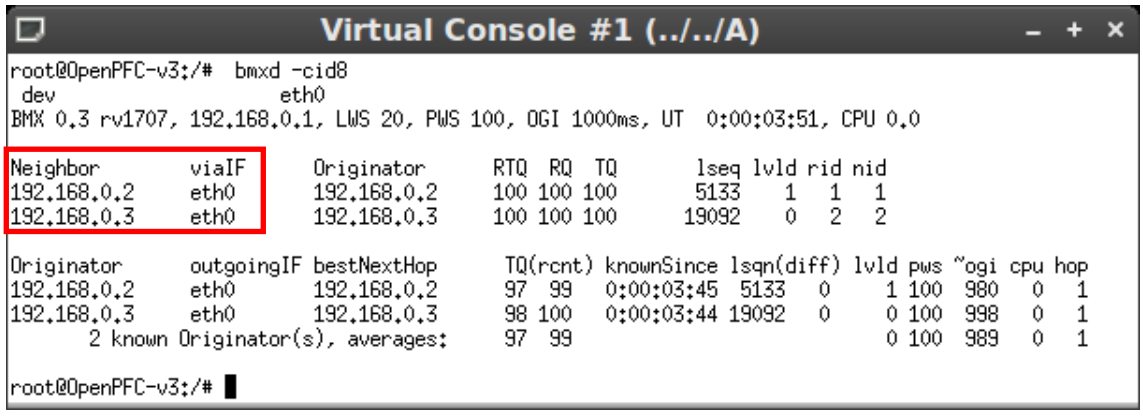


Figura 6-6. Vecinos del Nodo A, antes de aplicar la regla de filtrado

Posterior a la programación de las reglas de filtrado en las máquinas virtuales B y C:

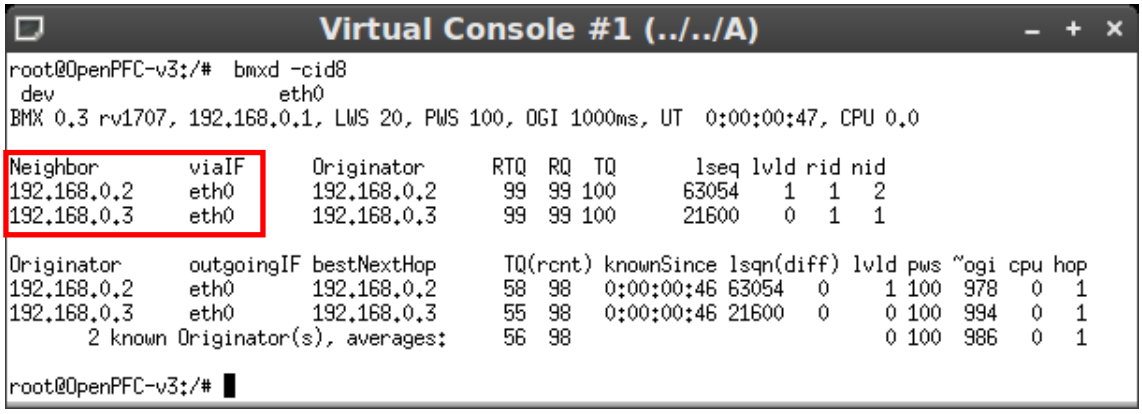


Figura 6-7. Vecinos del Nodo A, tras aplicar la regla de filtrado.

En este caso y al no verse afectado el nodo A por dichas reglas, la tabla confeccionada por BMX a posteriori es idéntica a la inicial.

Las figuras muestran a los dos vecinos B (192.168.0.2) y C (192.168.0.3) alcanzables a través de la interfaz eth0 a 1 salto de distancia cada uno.

NodoB (eth0: 192.168.0.2)

Previa a la programación de las reglas de filtrado en las máquinas virtuales B y C:

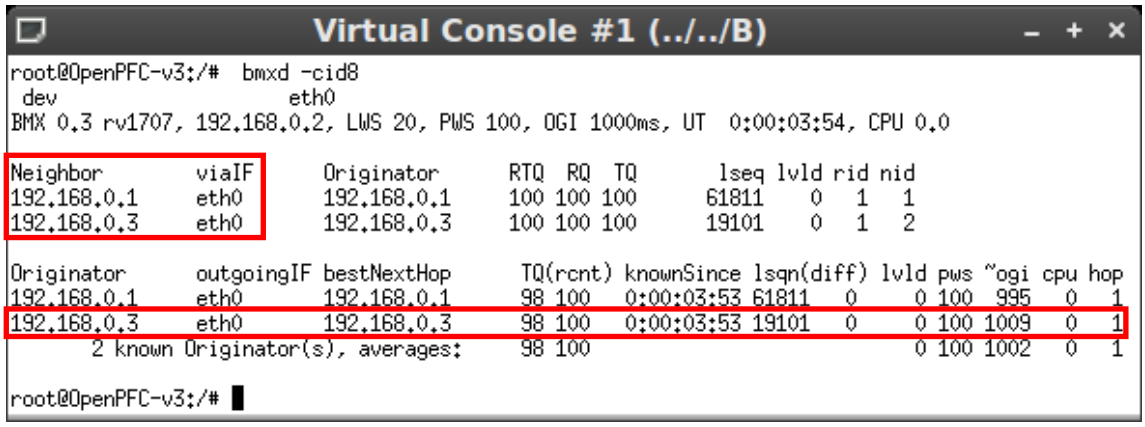


Figura 6-8. Vecinos del Nodo B, antes de aplicar la regla de filtrado

Posterior a la programación de las reglas de filtrado en las máquinas virtuales B y C:

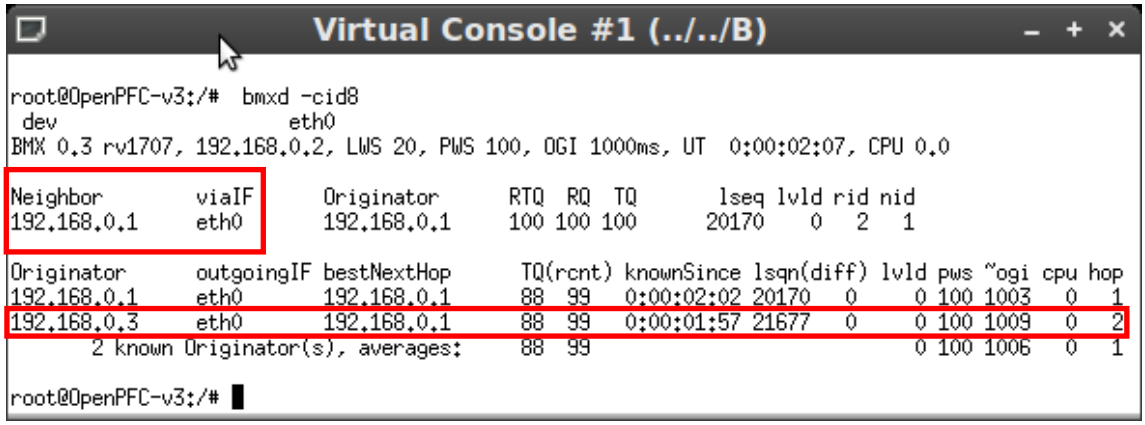


Figura 6-9. Vecinos del Nodo B, tras aplicar la regla de filtrado.

En este caso el filtrado del tráfico proveniente del nodo C (192.168.0.3) suprime la recepción de sus mensajes Originator BMX, eliminándolo de la tabla de vecinos.

En la segunda tabla, el mapa de la red se ha modificado para reflejar la nueva topología. El nodo A sigue estando a un salto de distancia, mientras que el nodo C (192.168.0.3) pasa a estar a dos saltos, alcanzable a través del nodo A (192.168.0.1).

Nodo C (eth0: 192.168.0.3)

Previa a la programación de las reglas de filtrado en las máquinas virtuales B y C:

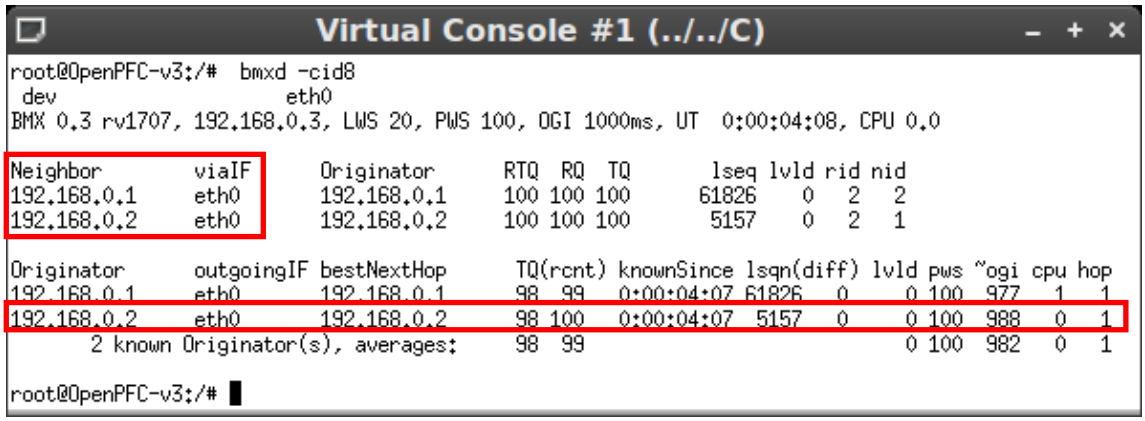


Figura 6-10. Vecinos del Nodo C, antes de aplicar la regla de filtrado

Posterior a la programación de las reglas de filtrado en las máquinas virtuales B y C:

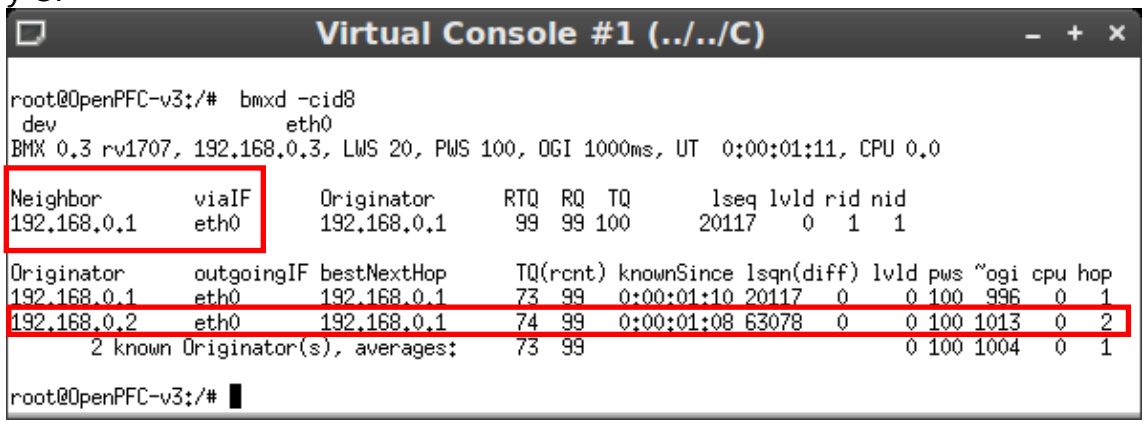


Figura 6-11. Vecinos del Nodo C, tras aplicar la regla de filtrado

Del mismo modo que ocurre en la máquina virtual del nodo B, el protocolo BMX ejecutado en el nodo C tras la aplicación del filtrado redescubre la topología de la red modificando sus tablas para reflejar la nueva situación: Un solo vecino, el nodo A, y nueva ruta para llegar hasta el nodo B, a través del nodo A.

6.5.2 Pérdidas

Debido a la influencia de interferencias radioeléctricas de sistemas externos o como resultado de la interposición de obstáculos fijos o móviles en el espacio de propagación de las señales, la transmisión de datos puede sufrir pérdidas de diversa consideración.

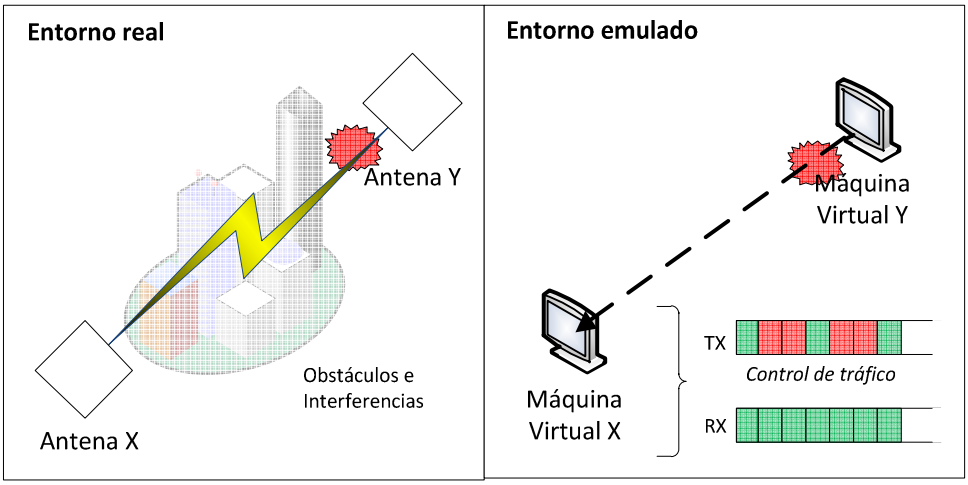


Figura 6-12. Pérdida de información en enlace punto a punto.

Esta problemática es frecuente en entornos urbanos y, como pone de relieve la figura 6-12, puede afectar de forma diferente a los dos sentidos del enlace.

Este efecto puede emularse mediante el descarte de paquetes de datos, haciendo uso de las herramientas de Netem a través del comando tc en los buffers de salida o de entrada de las interfaces de los dispositivos de la red.

Concretamente en el caso de la figura anterior, nos encontramos con pérdidas en la recepción de la Antena Y que no afectan a su capacidad para transmitir con éxito y que la Antena X reciba sus datos correctamente.

Para ello, se generan una pérdida de datos deliberada en la transmisión de datos de la Máquina Virtual X para trasladar las pérdidas a la recepción de datos en la Máquina Virtual Y.

Como puede observarse en la figura, la recepción de datos en la máquina Virtual X no sufre pérdida alguna, reflejando fielmente el fenómeno de pérdidas asimétricas en entorno radio.

En la siguiente figura, se muestra un escenario virtual de ejemplo con cuatro nodos y tres enlaces inalámbricos emulados.

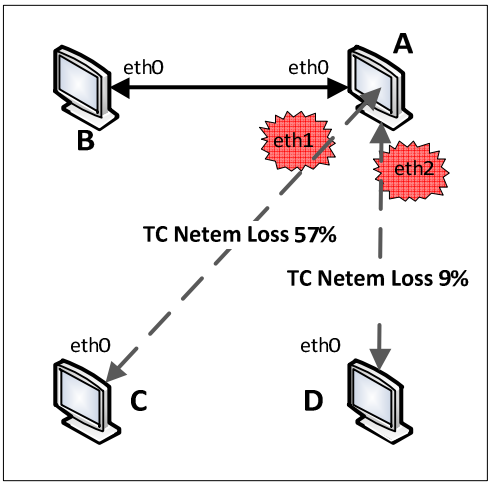


Figura 6-13. Escenario con pérdidas del 9% y el 57%

Como refleja la figura, dos de ellos sufren pérdidas de diversa consideración:

- Dominio de colisión A_B:
  - A (eth0: 192.168.3.1) – B (eth0: 192.168.3.2): Sin pérdidas de datos.
- Dominio de colisión A_C:
  - A (eth1: 192.168.1.1) – C (eth0: 192.168.1.2): Pérdidas del 57%.
- Dominio de colisión A_D:
  - A (eth2: 192.168.2.1) – D (eth0: 192.168.2.2): Pérdidas del 9%.

La configuración de la cola de pérdidas con tc puede realizarse en cualquiera de las interfaces extremo. Para clarificar el ejemplo, las configuraciones se harán en los nodos extremo, C y D.

Estas funciones dependen del módulo de Kernel “sch_netem.ko”, que debe estar cargado en el momento de utilización de la disciplina de pérdidas:

```
#> insmod /lib/modules/2.6.32.25/sch_netem.ko
```

Y comprobamos su correcta carga mediante el comando:

```
#> lsmod | grep sch_netem
sch_netem                4152  0
```

En el terminal de la máquina virtual “C” creamos una cola que descartará el 57% de los paquetes.

```
#C> tc qdisc add dev eth0 root netem loss 57%
```

```
Comprobamos que la cola se ha creado correctamente:
```

```
#C> tc qdisc show
>qdisc netem 8001: dev eth0 root refcnt 2 limit 1000 loss 57%
```

La siguiente figura ilustra el resultado de un ICMP ping desde la máquina virtual “C” a la máquina virtual “A”, donde quedan reflejadas unas pérdidas muy cercanas (54%) al valor configurado.

```
root@C:/# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=1 ttl=64 time=1.666 ms
64 bytes from 192.168.1.1: seq=2 ttl=64 time=1.327 ms
64 bytes from 192.168.1.1: seq=4 ttl=64 time=1.329 ms
[…]
64 bytes from 192.168.1.1: seq=97 ttl=64 time=1.283 ms
64 bytes from 192.168.1.1: seq=98 ttl=64 time=1.327 ms
64 bytes from 192.168.1.1: seq=104 ttl=64 time=1.263 ms
^C
--- 192.168.1.1 ping statistics ---
105 packets transmitted, 48 packets received, 54% packet loss
round-trip min/avg/max = 0.570/43.382/1004.597 ms
```

Figura 6-14. ICMP ping con pérdidas del 57%

A continuación, realizamos la misma operación en el terminal de la máquina virtual “D”, para emular pérdidas del 9% en su único enlace, con la máquina virtual “A”.

```
root@D:/# tc qdisc add dev eth0 root netem loss 9%
```

Como en el caso anterior, comprobamos que la cola se ha creado correctamente:

```
root@D:/# tc qdisc show
>qdisc netem 8001: dev eth0 root refcnt 2 limit 1000 loss 9%
```

A continuación comprobamos que la configuración causa el efecto deseado en el entorno virtual.

```
root@D:/# ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1): 56 data bytes
64 bytes from 192.168.2.1: seq=0 ttl=64 time=2.738 ms
64 bytes from 192.168.2.1: seq=1 ttl=64 time=1.714 ms
64 bytes from 192.168.2.1: seq=2 ttl=64 time=1.429 ms
[…]
64 bytes from 192.168.2.1: seq=105 ttl=64 time=1.355 ms
64 bytes from 192.168.2.1: seq=106 ttl=64 time=1.332 ms
64 bytes from 192.168.2.1: seq=107 ttl=64 time=1.249 ms
^C
--- 192.168.2.1 ping statistics ---
108 packets transmitted, 94 packets received, 12% packet loss
round-trip min/avg/max = 0.590/1.341/2.738 ms
```

Figura 6-15. ICMP Ping con pérdidas del 9%

Del mismo modo, el resultado de un ICMP ping desde la máquina virtual “D” a la máquina virtual “A”, en la figura 6-1 muestra unas pérdidas (12%) cercanas al valor configurado.

Aunque el método mostrado es el más directo y el que menos recursos consume, también se pueden emular pérdidas en recepción mediante el uso de colas de pérdida en los búferes de recepción del nodo local.

Para ello, es necesario compilar y cargar el módulo ifb en el Kernel de Linux y utilizarlo para crear un dispositivo ethernet lógico que, actuando como intermediario, emulará pérdidas en la recepción de datos, en lugar de en la transmisión.

6.5.3 Tasa de transmisión

A pesar de que Netem no dispone de mecanismos específicos de control de velocidad de transferencia, sí que cuenta con disciplinas que permiten conformar el tráfico y su velocidad de salida, como el Token Bucket (TBF, Token Bucket Filter).

Para utilizar esta disciplina de colas es necesario cargar el módulo de Kernel “sch_tbf.ko”:

```
#> insmod /lib/modules/2.6.32.25/sch_tbf.ko
```

Y comprobamos su correcta carga mediante el comando:

```
#> lsmod | grep sch_tbf
sch_tbf                 3128  1
```

Para limitar la tasa de transmisión de una interfaz de red, basta con añadir una disciplina de cola fijando tres parámetros básicos del filtro Token Bucket:

- **Rate**: Velocidad de transferencia en Kbps.
- **Burst**: Tamaño máximo de la ráfaga de datos en Bytes.
- **Latency**: Latencia añadida por la cola, en milisegundos. Este parámetro puede fijarse al mínimo (1 ms) si no se desea añadir retardo a la conformación de tráfico.

A modo de ejemplo, se limita la velocidad de un enlace a 4 Mbps, correspondiente a la interfaz eth0:

```
#> tc qdisc add dev eth0 root tbf burst 1540 rate 4096 latency 1
```

```
#> tc qdisc show dev eth0
qdisc tbf 8003: root refcnt 2 rate 4096bit burst 1539b lat 1us
```

7 Escenarios de prueba

7.1 Emulación de un escenario real

Para este apartado, se ha creado diseñado un pequeño escenario de topología sencilla pero que aúna dos de las particularidades más características de las topologías radio.

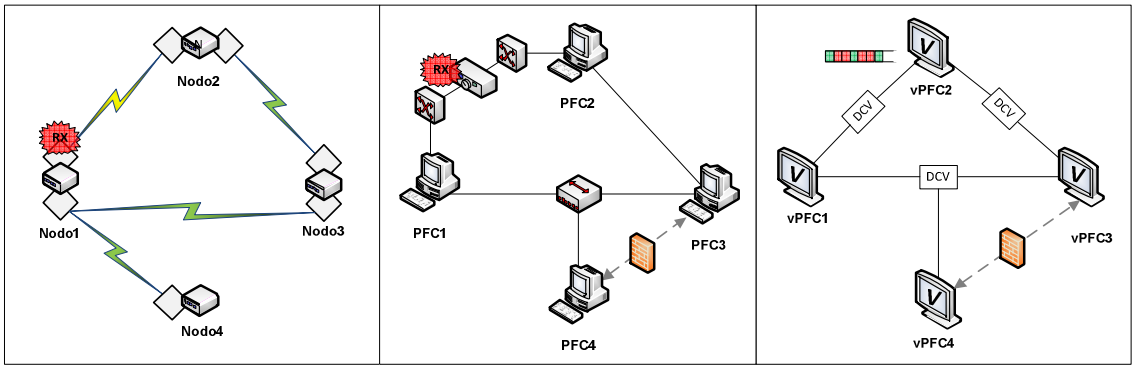


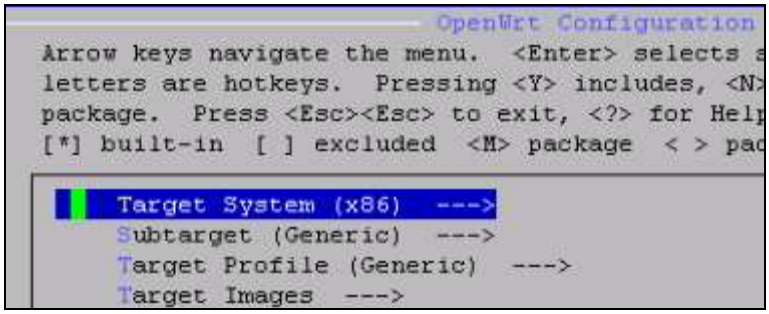
Figura 7-1. Escenarios real, emulado y virtual.

Dicho escenario se encuentra representado en el primer cuadro de la figura 9-1, en el que existe un enlace (el une a los Nodos 1 y 2) con pérdidas en un sentido de la transmisión y un enlace punto a multipunto (el que une a los Nodos 1, 3 y 4).

El objetivo de este capítulo es el de crear un escenario real que represente con la mayor fidelidad posible la topología diseñada y realizar sobre éste una serie de mediciones. Estas mediciones serán posteriormente contrastadas con las obtenidas en una representación virtual del mismo, basada en NetKit y las herramientas modeladoras descritas en los capítulos anteriores.

Escenario Real

Para la creación de escenario contamos con 4 PCs con la misma distribución de OpenWRT utilizada en los nodos virtuales pero, en este caso, compilada para la plataforma x86 en lugar de UML.



Con el objeto de maximizar la semejanza entre la distribución de OpenWRT utilizada en los PCs y la de los nodos virtuales, la única diferencia introducida son los drivers de las interfaces de red.

Dadas las importantes restricciones que sufren las distribuciones Linux diseñadas para dispositivos embebidos como OpenWRT, es necesario seleccionar el hardware adecuado para evitar cargar el sistema con drivers adicionales como USB, SATA o PCI Express.

Los enlaces radio serán simulados con enlaces Ethernet cableados a efectos de facilitar la inducción programada de pérdida de datos, al ser más sencillo y preciso desconectar la transmisión o recepción de un cable Ethernet que provocar un determinado porcentaje de pérdidas generando interferencias en un sistema radio.

Se estima que las diferencias en las capas OSI 1 y 2 correspondientes a nivel físico y enlace que existen entre ambas tecnologías no afectan a los resultados de las pruebas realizadas.
Para simular la pérdida de datos en el enlace que une PFC1 y PFC2, se utiliza un interruptor destinado a interrumpir el par de hilos de recepción del nodo PFC1.

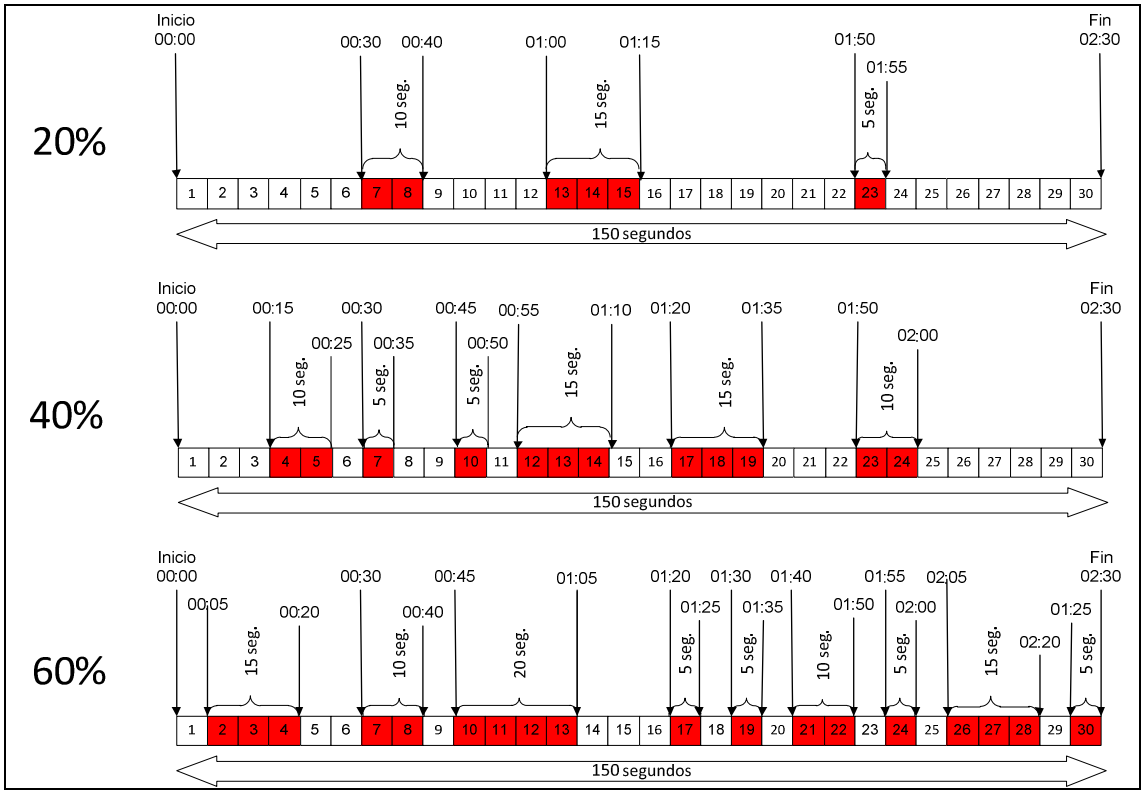


Figura 7-3. Esquema de pérdidas programadas.

Mediante las secuencias temporizadas de la figura 9-3 se provocan pérdidas del 20%, 40% y 60%, monitorizando el comportamiento del protocolo de

encaminamiento BMX y evaluando los cambios en la topología de encaminamiento.

En lo referente a la topología punto a multipunto presente entre los nodos PFC1, PFC3 y PFC4, se utiliza un hub Ethernet, uniendo las interfaces eth0 de los tres nodos en un mismo dominio de colisión físico. Para reproducir el aislamiento entre PFC4 y PFC3 se utilizará la misma técnica que en el escenario virtual (véase Netfilter, en el capítulo Herramientas Modeladoras) al no ser posible producir el mismo efecto con una red Ethernet.

Escenario Virtual

El escenario real está igualmente compuesto de 4 nodos siguiendo la misma topología e imagen de OpenWRT que la utilizada en el escenario real.

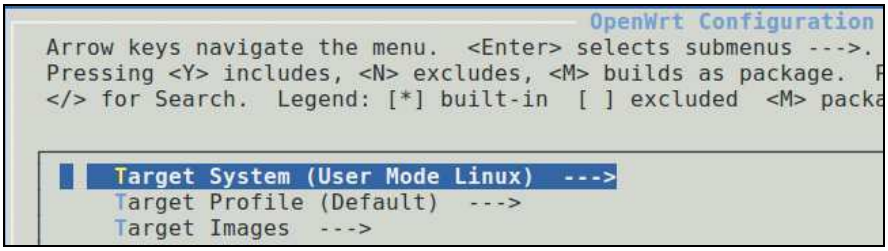


Figura 7-4. Compilación de OpenWRT para el sistema UML.

El único cambio respecto de la compilación de la imagen utilizada en los nodos del escenario real es, como muestra la figura 9-4, es el sistema objetivo. UML en este caso.
Para arrancar el escenario completo basta con ejecutar el comando vstart de NetKit con los parámetros adecuados para iniciar cada uno de los nodos:

```
$> vstart PFC1 --eth0=12 --eth1=134 --eth2=A --con0=xterm --
con1=xterm --verbose --mem=32 -m openpfc.img -k openpfc-uml-
vmlinux --append=root=98:0 --append=init=/etc/preinit
```

```
$> vstart PFC2 --eth0=12 --eth1=23 --eth2=B --con0=xterm --
con1=xterm --verbose --mem=32 -m openpfc.img -k openpfc-uml-
vmlinux --append=root=98:0 --append=init=/etc/preinit
```

```
$> vstart PFC3 --eth0=23 --eth1=134 --eth2=C --con0=xterm --
con1=xterm --verbose --mem=32 -m openpfc.img -k openpfc-uml-
vmlinux --append=root=98:0 --append=init=/etc/preinit
```

```
$> vstart PFC4 --eth0=134 --eth1=D --con0=xterm --con1=xterm --
verbose --mem=32 -m openpfc.img -k openpfc-uml-vmlinux --
append=root=98:0 --append=init=/etc/preinit
```

En este caso y al contrario que en el escenario real las pérdidas se emularán mediante tc. Para ello se realizarán medidas en el sistema sin pérdidas y se compararán tras emular un 20%, un 40% y finalmente un 60% de pérdidas en la interfaz eth0 de PFC2.

```
#>tcqdisc add dev eth0 root netem loss 20%
#>tcqdisc add dev eth0 root netem loss 40%
#>tcqdisc add dev eth0 root netem loss 60%
```

Esta configuración, aplicada en las colas de salida de la interfaz eth0 de PFC2, afecta directamente a la recepción de la interfaz eth0 de PFC1.
De este modo, emulamos un problema de recepción en PFC1, solamente para esa interfaz en particular.

Configuración de los nodos

La configuración de ambos nodos es exactamente la misma, por lo que se puede aplicar a ambos escenarios Virtual y Real sin necesidad de efectuar ningún cambio o adaptación.
La configuración básica de red se muestra en la siguiente figura.

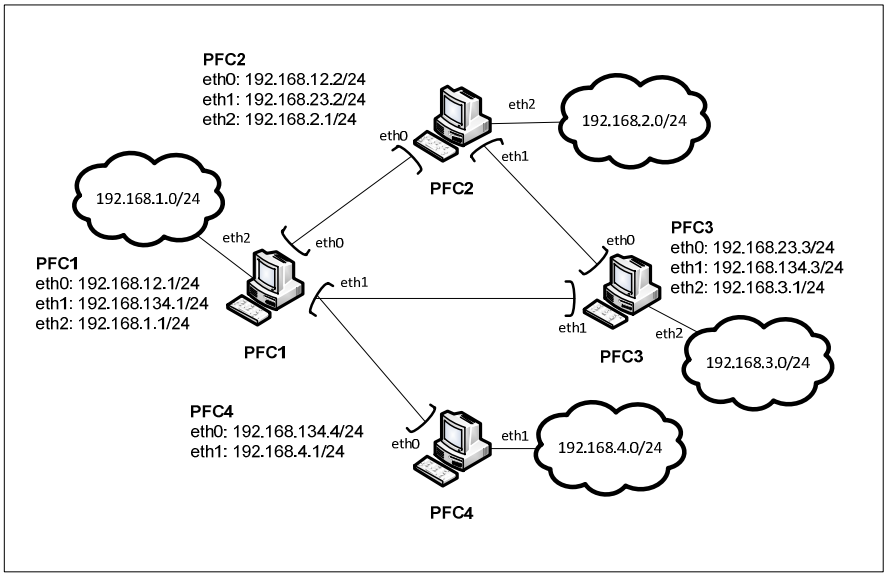


Figura 7-5. Esquema de red utilizado en las pruebas.

Por lo que respecta a la configuración de BMX, se inicia el servicio en todas las interfaces conectadas a la red BMX (todas excepto eth2 en PFC1, 2 y 3 y eth1 en PFC4).
Para el arranque se genera un pequeño script que, alojado en "/etc/init.d/script-arranque" puede ser habilitado mediante el comando:

```
#> /etc/init.d/script-arranque enable
```

Este comando generará automáticamente los scripts de arranque y detención para todos los runlevels de Linux, permitiendo que el script se arranque y detenga con el sistema.

```
#!/bin/sh /etc/rc.common

START=99
STOP=15

start() {
echo start
bmxddev=eth0 dev=eth1
                    bmxd -c --dev eth0 /ttl 50
                    bmxd -c --dev eth0 /hide 0
                    bmxd -c --dev eth1 /ttl 50
                    bmxd -c --dev eth1 /hide 0
                    bmxd -c -a 192.168.1.0/24
}

stop() {
echo stop
# commandstokillapplication
}
```

Figura 7-6. Script de configuración de bmxd para el nodo PFC1

Como puede observarse en la anterior figura, se utiliza el TTL por defecto (50 saltos) para los mensajes OGM, así como el parámetro “hide” desactivado, para enviar estos mensajes a través de todas las interfaces y no solamente desde la primaria (en este caso eth0, la primera declarada).

Finalmente, también se incluye el anuncio HNA de la red de clientes conectada al nodo PFC1, la 192.168.1.0/24.

Cualquiera de los valores puede ser modificado y aplicado simplemente editando el script y ejecutando:

```
#> /etc/init.d/script-arranque disable
#> /etc/init.d/script-arranque enable
```

Para actualizar automáticamente los scripts de arranque en los runlevel.

Análisis, comparativa y conclusiones

Durante la realización de las pruebas de este capítulo se realizaron dos tipos de mediciones para determinar el comportamiento de BMX en relación con el porcentaje de pérdida de datos.
La primera medición consiste en enviar paquetes ICMP de un extremo a otro de la red mediante la herramienta traceroute.

```
root@PFC4:/# traceroute 192.168.2.1
traceroute to 192.168.2.1 (192.168.2.1), 30 hops max, 38 byte packets
 1  192.168.12.1 (192.168.12.1)  1.275 ms  1.860 ms  0.736 ms
 2  192.168.2.1 (192.168.2.1)  11.729 ms  1.099 ms  0.976 ms
```

Figura 7-7. Traceroute desde el nodo PFC4 a PFC2. Pérdidas entre el 0 y 20%.

Como revela la figura 9-7, tanto en el instante inicial como con una inducción de pérdidas del 20% en el enlace entre PFC1 y PFC2, los paquetes ICMP de

traceroute van de PFC4 a PFC2 pasando directamente por este enlace en ambos escenarios Virtual y Real.

```
root@PFC1:/#  ip route list table 65
192.168.4.0/24 via 192.168.134.4 deveth1  proto static  src 192.168.12.1
192.168.3.0/24 via 192.168.134.3 deveth1  proto static  src 192.168.12.1
192.168.2.0/24 via 192.168.12.2 deveth0  proto static  src 192.168.12.1
throw 192.168.1.0/24  proto static
throw 127.0.0.0/8  proto static
```

Figura 7-8. Tabla de rutas HNA. Pérdidas entre el 0 y 20% entre PFC1 y PFC2.

Si, simultáneamente, listamos la tabla de rutas 65 del nodo PFC1, correspondiente a las rutas HNA, puede comprobarse como los paquetes dirigidos a la subred cliente de PFC1 (192.168.2.0/24) se encaminan directamente a través del enlace PFC1 - PFC2 (via 192.168.12.2, la interfaz eth0 de PFC2).
Si incrementamos el nivel de pérdidas en el enlace PFC1 – PFC2 y repetimos las mediciones, nos encontramos con la siguiente situación:

```
root@PFC4:/# traceroute 192.168.2.1
traceroute to 192.168.2.1 (192.168.2.1), 30 hops max, 38 byte packets
 1  192.168.12.1 (192.168.12.1)  2.406 ms  0.625 ms  0.488 ms
 2  192.168.23.3 (192.168.23.3)  2.097 ms  0.689 ms  0.675 ms
 3  192.168.2.1 (192.168.2.1)  2.396 ms  1.066 ms  0.895 ms
```

Figura 7-9. Traceroute desde el nodo PFC4 a PFC2. Perdidas entre el 40% y el 60%.

Como puede observarse en la figura 9-9, en esta ocasión, los paquetes ICMP son encaminados a través de PFC3, con el fin de evitar el enlace PFC1 – PFC2 que, en esta fase de la prueba, sufre unas pérdidas del 40 – 60%.
Si volvemos a listar la tabla de rutas 65, podemos comprobar que la ruta para la red destino 192.1682.0/24 se ha actualizado y utiliza como siguiente salto la interfaz eth1 de PFC3 (192.168.134.3).

```
root@PFC1:/#  ip route list table 65
192.168.4.0/24 via 192.168.134.4 deveth1  proto static  src 192.168.12.1
192.168.3.0/24 via 192.168.134.3 deveth1  proto static  src 192.168.12.1
192.168.2.0/24 via 192.168.134.3 deveth1  proto static  src 192.168.12.1
throw 192.168.1.0/24  proto static
throw 127.0.0.0/8  proto static
```

Figura 7-10. Tabla de rutas HNA. Pérdidas entre el 40 y 60% entre PFC1 y PFC2.

Para completar el estudio de la comparativa en ambos escenarios real y virtual, pasamos a analizar las capturas del protocolo BMX, con una especial atención al parámetro RTQ, calculado por BMX y con el que se caracteriza la calidad bidireccional de cada enlace de la red.

Escenario Real			
PFC1			
Sin Pérdidas	20% Pérdidas	40% Pérdidas	60% Pérdidas
100	**81,08**	**61,19**	**47,00**
PFC2			
Sin Pérdidas	20% Pérdidas	40% Pérdidas	60% Pérdidas
100	**81,85**	**63,37**	**44,61**

Escenario Virtual			
PFC1			
Sin Pérdidas	20% Pérdidas	40% Pérdidas	60% Pérdidas
100	**79,59**	**58,56**	**45,75**
PFC2			
Sin Pérdidas	20% Pérdidas	40% Pérdidas	60% Pérdidas
100	**81,33**	**60,49**	**49,45**

Tabla 7-1. RTQ Medida en PFC1 y PFC2 en escenarios Real y Virtual.

Como puede observarse en la tabla 9-1, Ambos escenarios real y virtual muestran unas desviaciones mínimas entre la calidad de envío y recepción RTQ emulada en el escenario virtual y la obtenida en el escenario real.

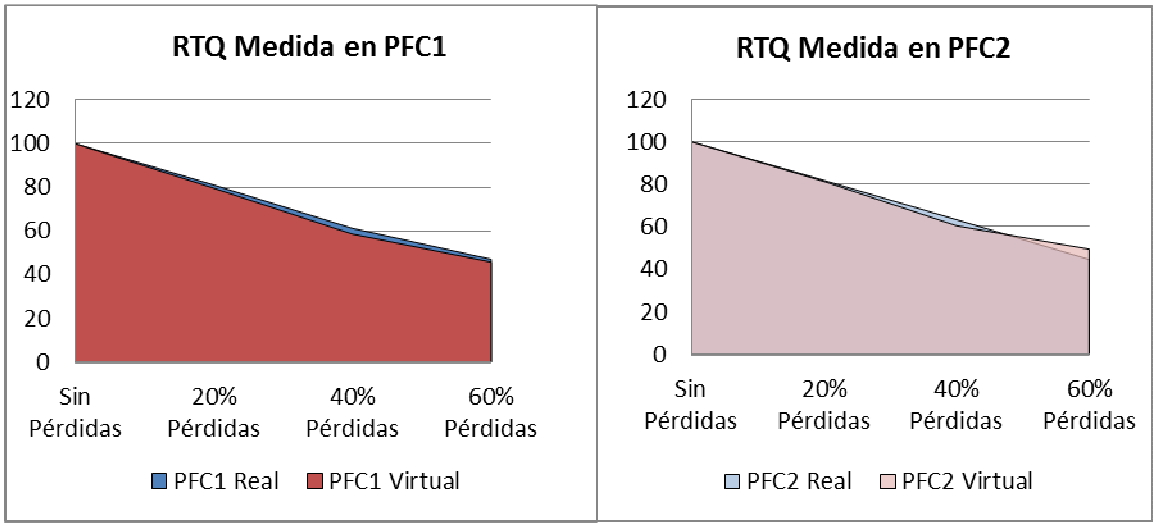


Figura 7-11. Comparativa de RTQ en escenarios Real y Virtual.

La anterior figura muestra los valores medios de RTQ medidos a lo largo de la prueba en ambos extremos del enlace con pérdidas (en los nodos PFC1 y PFC2).

Escenario Real vs Virtual			
ΔRTQ @ PFC1			
Sin Pérdidas **0%**	20% Pérdidas **1,49%**	40% Pérdidas **2,63%**	60% Pérdidas **1,25%**
ΔRTQ @ PFC2			
Sin Pérdidas **0%**	20% Pérdidas **0,52%**	40% Pérdidas **2,88%**	60% Pérdidas **4,84%**

Tabla 7-2. Desviación de RTQ en escenarios Real y Virtual.

Como puede verse en la tabla 9-2, La diferencia entre las medidas obtenidas en el entorno emulado y el real son menores al 5%, similitud que se refleja en la figura 9-11, al contraponer los valores medios medidos en ambos escenarios.

La conclusión a la que se llega tras analizar los resultados del experimento es que, mediante las herramientas y procedimientos descritos en este proyecto, es posible simular pérdidas de datos en enlaces de una red emulada mediante NetKit y predecir así el comportamiento que este mismo fenómeno provocaría en los protocolos de comunicación de una red real.

7.2 Estudio de escalabilidad

Para este apartado se ha creado un escenario de gran complejidad, destinado a evaluar la degradación en las prestaciones de una red mallada a medida que ésta crece hasta doblar su tamaño.

Inicialmente se parte de una topología de 4 nodos semejante a la del apartado anterior, que representa una distribución muy frecuente en redes abiertas y metropolitanas.

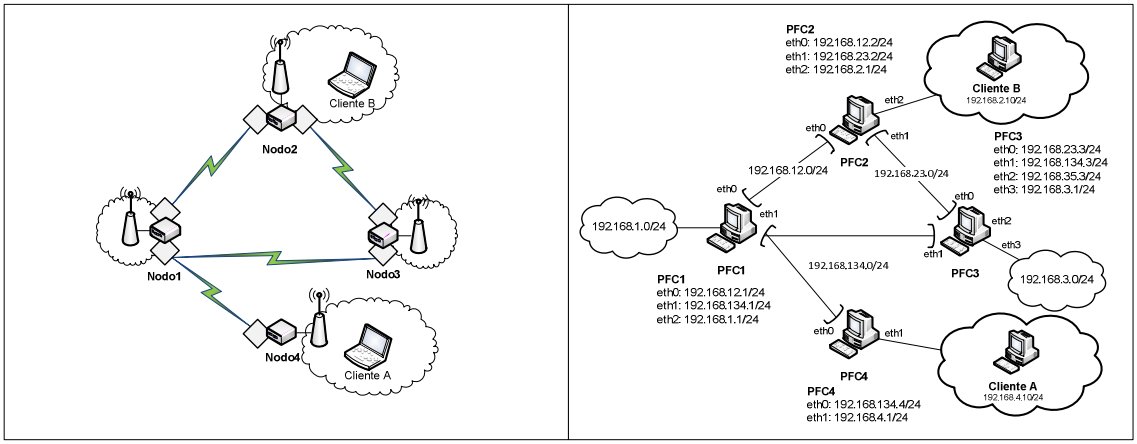


Figura 7-12. Nube inicial de 4 nodos.

Como refleja la Figura 9-12, los nodos están interconectados mediante una red troncal de enlaces punto a punto y punto a multipunto y cada uno cuenta con una interfaz adicional en modo infraestructura, (red de acceso) que actúa de punto de acceso Wi-Fi para los clientes cercanos.

Para comenzar la prueba, se realiza una transferencia TCP con iperf, equivalente a una descarga FTP entre el Cliente A y el Cliente B (ambos máquinas virtuales basadas en la distribución Linux Debian).

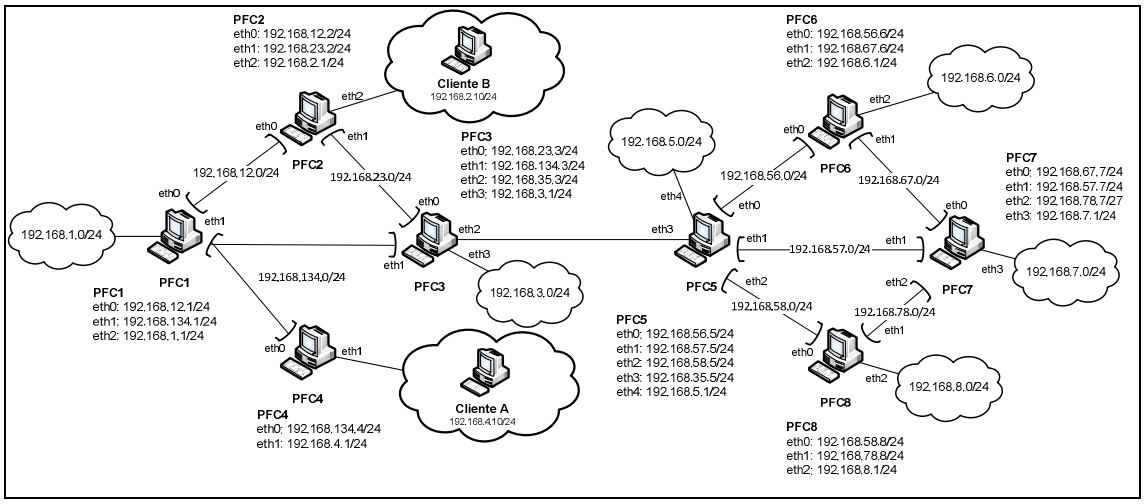


Figura 7-13. Escenario completo de la prueba.

Se realizan un total de 20 transferencias de 60 segundos de duración cada una, dando una throughput medio de 21,02 Mbps.

Tras conectar dos nodos adicionales a la red (PFC5 y PFC7), se realizan 20 medidas más, y se registra una pérdida en el throughput con una tasa media de 20,33 Mbps.

Del mismo modo, se repite la prueba con los 8 nodos conectados, dando como resultado un throughput medio de 19,4 Mbps.

La tabla 9-3 recoge los datos resultantes de las tres fases del escenario, representado por completo en la figura 9-13 en su forma final con 8 nodos, 8 redes cliente, 2 máquinas virtuales cliente, 2 enlaces de acceso, 8 enlaces punto a punto, y 1 enlace punto a multipunto.

	Tasa de transferencia (Mbps)		
Tiempo (seg)	4 nodos	6 nodos	8 nodos
60	21,70	20,40	19,60
120	22,00	20,50	19,80
180	20,20	20,30	19,70
240	20,20	20,30	19,50
300	20,20	20,50	19,40
360	21,20	20,20	19,10
420	21,40	20,50	19,40
480	20,90	20,10	19,30
540	21,00	20,30	19,40
600	20,90	20,40	19,60
660	20,90	20,20	18,70
720	20,90	20,50	19,60
780	21,10	20,30	19,80
840	21,10	20,30	19,80
900	21,00	20,30	17,60
960	20,90	20,50	19,70
1020	21,20	20,20	19,50
1080	21,10	20,20	19,40
1140	21,20	20,20	19,40
1200	21,20	20,30	19,60
Promedio (Mbps)	21,02	20,33	19,40

Tabla 7-3. Resultados de las mediciones de escalabilidad.

Tal y como se describe en el capítulo 4 en la concepción de NetKit no priman las prestaciones, por lo que la magnitud de la tasa de transferencia no es comparable con el escenario real, aunque las proporciones sí que lo son.

De esta comparación se extrae que existe una caída del decremento medio del throughput del 1,99% por cada nodo adicional que se une a la red.

Con el fin de maximizar la estabilidad en el consumo de los recursos de la máquina anfitrión, los 8 nodos y las 2 máquinas virtuales se encuentran iniciadas desde el comienzo de la prueba y se conectan paulatinamente a la red, a medida que esta se expande.

```
root@PFC1:/# bmxd -cid8
BMX 0.3 rv1707, 192.168.12.1, LWS 20, PWS 100, OGI 1000ms, UT  0:00:15:04,

Neighbor        viaIF     Originator      RTQ  RQ  TQ      lseq lvld rid nid
192.168.12.2    eth0       192.168.12.2    100 100 100     12154    0   2   0
192.168.134.3   eth1       192.168.134.3   100 100 100     63013    0   2   0
192.168.134.4   eth1       192.168.134.4   100 100 100     25437    0   1   1

Originator      outgoingIF bestNextHop      TQ(rcnt) knownSince lsqn(diff) lvld pws ~ogi cpu hop
```

```
192.168.12.2    eth0      192.168.12.2    99 100   0:00:14:53 12154   0    0 100 1020   0   1
192.168.23.2    eth0      192.168.12.2    98  99   0:00:14:52 12154   0    0 100 1020   0   1
192.168.23.3    eth1      192.168.134.3   96  93   0:00:14:53 63013   0    0 100 1004   0   1
192.168.35.3    eth1      192.168.134.3   98  99   0:00:14:55 63013   0    0 100 1004   0   1
192.168.35.5    eth1      192.168.134.3   98  99   0:00:09:32  2136   0    0 100  982   0   2
192.168.56.6    eth1      192.168.134.3   92  97   0:00:09:32 35839   0    0 100 1040   0   3
192.168.56.5    eth1      192.168.134.3   95  94   0:00:09:31  2136   0    0 100  985   0   2
192.168.57.5    eth1      192.168.134.3   95  95   0:00:09:31  2136   0    0 100  982   0   2
192.168.57.7    eth1      192.168.134.3   83  59   0:00:09:31 22893   0    0 100  988   0   4
192.168.58.5    eth1      192.168.134.3   95  91   0:00:09:31  2136   0    0 100  982   0   2
192.168.58.8    eth1      192.168.134.3   95  99   0:00:09:27 35821   0    0 100 1020   0   3
192.168.67.2    eth1      192.168.134.3   49  51   0:00:09:30 35839   0    0 100 1034   0   3
192.168.67.7    eth1      192.168.134.3    8  30   0:00:01:30 22817   0   77 100  995   0   4
192.168.78.7    eth1      192.168.134.3   74  76   0:00:09:30 22894   0    0 100 1008   0   3
192.168.78.8    eth1      192.168.134.3   97  99   0:00:08:43 35821   0    0 100 1020   0   3
192.168.134.3   eth1      192.168.134.3   99 100   0:00:14:57 63013   0    0 100 1004   0   1
192.168.134.4   eth1      192.168.134.4   98  99   0:00:15:03 25437   0    0 100  990   0   1
    17 known Originator(s), averages:     86  87                        4 100 1004   0   2
```

Figura 7-14. Mapa de la red troncal confeccionado por BMX en el nodo PFC1

A efectos ilustrativos, la figura 9-14, muestra el mapa troncal confeccionado por el protocolo BMX, donde aparecen los 3 nodos vecinos de PFC1 (PFC2, PFC3 y PFC4) así como los 17 Originators (El resto de interfaces que implementan BMX en los 7 nodos restantes, de los que se han recibido mensajes OGM)

```
root@PFC1:/# ip route list table 65
192.168.8.0/24 via 192.168.134.3 dev eth1  proto static  src 192.168.12.1
192.168.7.0/24 via 192.168.134.3 dev eth1  proto static  src 192.168.12.1
192.168.6.0/24 via 192.168.134.3 dev eth1  proto static  src 192.168.12.1
192.168.5.0/24 via 192.168.134.3 dev eth1  proto static  src 192.168.12.1
192.168.4.0/24 via 192.168.134.4 dev eth1  proto static  src 192.168.12.1
192.168.3.0/24 via 192.168.134.3 dev eth1  proto static  src 192.168.12.1
192.168.2.0/24 via 192.168.12.2 dev eth0  proto static  src 192.168.12.1
```

Figura 7-15. Tabla de rutas HNA generada por BMX en el nodo PFC1

A su vez, la figura 9-15 muestra la tabla de rutas HNA del mismo nodo, donde aparecen todas las redes cliente de la red y gracias a la cual, cualquier cliente conectado a su zona Wi-Fi, puede comunicarse con cualquier otro cliente o servidor de la red.

Mediante esta experiencia, podemos comprobar cómo, a medida que crece el número de nodos en la red, existe un incremento en el volumen de la señalización que,

8 Conclusiones

Durante el desarrollo de este proyecto se ha tomado consciencia de la problemática presente en el diseño e implementación de las pequeñas y medianas redes inalámbricas malladas que componen los proyectos de conectividad ciudadana como Guifi.net.

Con un contacto más cercano con los miembros de la comunidad Gràcia Sense Fils, se han identificado algunos de los retos derivados del crecimiento y unificación de los núcleos de conectividad que, en muchos casos heterogéneos, permiten la expansión de la red.

A modo de conclusiones generales podemos establecer que, mediante las herramientas y procedimientos descritos en este proyecto, es posible modelar redes malladas virtuales, utilizando los mismos nodos y los mismos sistemas operativos, servicios y protocolos de comunicación que se utilizan en los entornos reales.

Sobre estas redes virtuales, es posible recrear condiciones características del medio radio, como la atenuación y las pérdidas de datos asociadas, o los relativos a los equipos como la orientación y direccionalidad, que dan lugar a topologías de red únicas.

También nos es posible evaluar el impacto derivado del crecimiento de redes malladas en cómo el incremento de la señalización consume recursos, afectando al desempeño de la red.

El análisis de dichos resultados resulta útil a la hora de diseñar el compartimentado de la red mallada, determinando el tamaño óptimo de las zonas encaminadas mediante protocolos IGP, tales como BMX, y la ubicación de los nodos frontera.

Todos estos factores, resultan decisivos a la hora de estudiar la escalabilidad y plantear, desarrollar e implementar estrategias para el crecimiento de redes inalámbricas malladas como la de Guifi.net.

Líneas futuras de trabajo

A modo de introducción a las potenciales líneas de desarrollo e investigación basadas en este proyecto, se podrían definir dos ejes de actuación.

El primero englobaría la recreación de escenarios adicionales que implementen otros protocolos de encaminamiento utilizados en redes inalámbricas malladas, como OLSR, AODV o BGP.

Esta experiencia permitiría validar la teoría y comprobar, de forma práctica y efectiva, la compatibilidad de la metodología y las herramientas descritas con otros protocolos de encaminamiento.

El segundo eje, fomentaría la explotación de los procedimientos descritos mediante la creación de los procesos de provisión automática de los nodos virtuales, para facilitar la rápida creación de escenarios con configuraciones UCI predefinidas.

9 Bibliografía

Guifi net
http://guifi.net

Gràcia Sense Fils
http://graciasensefils.net

LUGRo, Linux User Group Rosario
http://www.lugro.org.ar

Ninux, Wireless Network Community of Rome
http://www.ninux.org

Roma Tre University – NetKit
http://wiki.netkit.org

Open Mesh
http://www.open-mesh.org

Iperf, Network performance and throughput measurement
http://iperf.sourceforge.net/

The Linux Documentation Project
http://tldp.org/

Linux Manual Online
http://linux.die.net/man

The Linux Foundation
http://www.linuxfoundation.org/

Netfilter Project, Firewalling, NAT and Packet Mangling for Linux
http://www.netfilter.org/

A. Neumann. ***Reference Manual of B.A.T.M.A.N.-eXperimental.*** **Open-Mesh.org**

W. Tsai, ***B.A.T.M.A.N. Daemon HowTo*****. Open-Mesh.org**

J. Vargas Ayora. ***Estudio y propuestas de mejora de la red GUIFI.NET (II).*** **TFC EPSC, UPC.**

E. Duran. ***Wifi mesh network nodes on guifi.net.*** **PFC EPSC, UPC.**

10 Glosario de acrónimos

AODV:	Ad-hoc On Demand Distance Vector
ARM:	Advanced RISC Machine
BATMAN:	Better Approach To Mobile Adhoc Networks.
BGP:	Border Gateway Protocol
BMX:	BATMAN Experimental
COW:	Copy On Write
CPU:	Central Processing Unit
DHCP:	Dynamic Host Configuration Protocol
DNS:	Domain Name Service
EQ:	Echo Quality
GPL:	General Public License
GRE:	Generic Routing Encapsulation
HNA:	Host and Network Association
HTTP(S):	(Secure) Hyper Text Transfer Protocol
IEEE:	Institute of Electrical and Electronics Engineers.
ICMP:	Internet Control Message Protocol
IDF:	Is-Direct Flag
IGP:	Interior Gateway Protocol
IMAP:	Internet Message Access Protocol
IP:	Internet Protocol
IPSec:	Internet Protocol Security
LAN:	Local Area Network
LDP:	Label Distribution Protocol
LIR:	Local Internet Registry
MPLS:	Multi-Protocol Label Switching
NAT:	Network Address Translation
NFS:	Network File System
OGM:	Originator Message
OLSR:	Optimized Link State Routing
OSPF:	Open Shortest Path First
PC:	Personal Computer
RADIUS:	Remote Authentication Dial In User Service
RAM:	Random Access Memory
(R)ARP:	(Reverse) Address Resolution Protocol
RIP:	Routing Information Protocol
RIPE:	Réseaux IP Européens
RISC:	Reduced Instruction Set Computing
RQ:	Reception Quality
RTQ:	Round Trip Quality
SMTP:	Simple Network Management Protocol
SPARC:	Scalable Processor Architecture
SSH:	Secure Shell
TCP:	Transmission Control Protocol
(T)FTP:	(Trivial) File Transfer Protocol
TQ:	Transmission Quality
TTL:	Time To Live
UCI:	Unified Configuration Interface
UDP:	User Datagram Protocol
UML:	User Mode Linux
VLAN:	Virtual Local Area Network.

A1) Anexo I

Parámetros de configuración de BMX

Detección de enlace bidireccional

El protocolo BMX solamente tiene en cuenta los mensajes OGM provenientes de los vecinos conectados mediante enlaces bidireccionales.

Cada nodo retransmite por todas sus interfaces BMX los OGM que recibe de sus vecinos. Por lo que respecta al enlace por el que ha recibido ese OGM, éste se retransmite con el IDF activado (Is Direct Flag) para dar a conocer al nodo emisor que existe un nodo BMX vecino al otro lado del enlace.

Si el emisor recibe un reenvío de su propio OMG original con el IDF activo, añade al vecino a su tabla y reconoce el enlace como bidireccional.

Si, por el contrario, no lo recibe tras un tiempo determinado, marcara el enlace como unidireccional, descartando los mensajes OGM recibidos a través de dicha interfaz.

El timeout está representado por un número del 1 al 100 que representa el número de intervalos de OGM sin respuesta que se deben esperar antes de declarar el enlace unidireccional. De darse este caso, los OGM recibidos a través del enlace en cuestión se marcarían con el flag UDF (Uni Directional Flag) y se descartarían.

Los valores más bajos son más estrictos en el control de la bidireccionalidad y permiten evaluar con mayor frecuencia el estado de la red y detectar con rapidez enlaces con gran asimetría. Los valores altos, por el contrario, admiten mayor tolerancia y hacen reaccionar a BMX con mayor lentitud ante los cambios.

Parámetro: --bi-link-timeout <valor>
/b <valor>

Hace referencia a una interfaz determinada.
El rango de valores está comprendido entre 1 y 100ypor defecto es 2.

Ventana de ranking vecinal

BMX selecciona el mejor vecino para llegar a cada nodo contabilizando el número de OGM recientes recibidos de cada vecino.
La ventana del ranking de vecinos determina la duración del periodo considerado como “reciente” para esta clasificación.

Parámetro: --window-size<valor>

El rango de valores está comprendido entre 1 y 250. Por defecto es 128.

A mayor tamaño de ventana, mayor es el tiempo de convergencia del protocolo y más lenta es su reacción a los cambios de la red.
También es importante considerar que un tamaño de ventana pequeño puede dar excesiva relevancia a pérdidas puntuales si estas se han producido durante ese intervalo de tiempo.

Compromiso entre rutas largas versus cortas pero inestables

A la hora de generar las rutas de encaminamiento, el algoritmo de Batman-III favorece los caminos cortos frente a los largos, incluso aunque los primeros tengan más pérdidas que los segundos o cuenten con velocidades de transferencia más lentas.

Este comportamiento se debe a que sólo se tiene en cuenta el OGM más rápido (el primero recibido desde un origen determinado) a la hora de elaborar el ranking de vecinos para llegar a ese origen.

Es evidente que, aunque un conjunto de enlaces sean capaces de retransmitir correctamente un mensaje OGM puntual, no significa que también sean capaces de soportar una transferencia prolongada de información de forma estable y sin pérdidas.

Existe un parámetro configurable que permite alterar el comportamiento por defecto y tomar en consideración no solo el primer (más rápido) OGM recibido, sino también los siguientes.

Parámetro: --dups-ttl<valor>

Permite aceptar varios OGM (duplicados) del mismo origen para que no sólo se tenga en cuenta el primero o más rápido.

El valor a fijar define la tolerancia a aceptar en el TTL, entre 0 y 10, siendo 0 desactivado (por defecto) y 10 el número que, restado al TTL del primero OGM recibido, establece el TTL más bajo tomado en consideración.

Este parámetro suele utilizarse en conjunción con alguno de los dos siguientes, que permiten fijar el tipo de ponderación aplicada al resto de OGMs tomados en consideración.

Parámetro: --dups-rate<valor>
Permite ajustar la probabilidad total con la que serán tomados en consideración el resto de OGMs. El <valor> es un porcentaje.

Parámetro: --dups-ttl-degradation<valor>

A diferencia del anterior, el valor fija un porcentaje de degradación de la probabilidad por cada salto adicional que recorre un OGM, comparado con el primer OGM recibido.

Impacto de enlaces asimétricos

En entorno radio, la asimetría de los enlaces es una característica muy común que no sólo es frecuente sino que, además, puede variar con el tiempo.

El tipo de asimetría que se analizará es la relativa a la calidad o estabilidad del enlace, cuando uno de los sentidos sufre una pérdida de datos significativa en comparación con el otro.

Esta característica es especialmente importante cuando BMX trata de determinar si un enlace con un vecino puede ser o no utilizado y en qué medida puede serlo, mediante la asignación de una puntuación de calidad.

- Calidad de los enlaces

El algoritmo de BMX, establece las rutas entre nodos basándose en el número de mensajes OGM recibidos a través de un determinado vecino. De este modo el algoritmo promociona las rutas con una buena recepción, pero es necesario tener en cuenta que estas rutas se utilizarán a posteriori para enviar tráfico al nodo origen de los OGM y, como se ha visto, estas rutas pueden fácilmente atravesar enlaces asimétricos.

- Calidad de transmisión del enlace (TQ)

Existen dos parámetros que permiten ajustar la función de aceptación de OGMs en función de la calidad de transmisión TQ con el vecino que ha entregado el mensaje.

--asymmetric-exp<valor>: Ignora los OGM recibidos para respetar enlaces asimétricos. Ignorar con probabilidad TQ^<valor>. El valor de exponente aceptado va de 0 a 3 y por defecto es 0.

--asymmetric-weigh<valor>: Ignora los OGM para respetar enlaces asimétricos. El rango de valores es la probabilidad de ignorar el mensaje. Por defecto es 0.

- Calidad de recepción del enlace (RQ)

--send-clones <valor>
/c <valor>

Hace referencia a una interfaz determinada.
Permite especificar la probabilidad de reenvío de OGMs. Por defecto es del 100% pero admite valores desde 0 a 300.

Permite modificar la probabilidad con la que la interfaz reenviará los mensajes OGM ofreciendo la posibilidad de hasta triplicarlos.
De esta manera, aumentan las probabilidades de que el mensaje llegue con éxito al siguiente salto a consta de aumentar el volumen de tráfico de señalización.

Anuncios HNA (configuración)

Como complemento a la descripción de HNA, en el apartado 2.2.2, se incluyen los comandos de BMXd utilizados para controlar la publicación de mensajes NHA:

-c -a<red/máscara>: comienza la publicación de mensajes HNA para la subred definida.

-c -a -<red/máscara>: detiene la publicación de mensajes HNA para la subred definida.

Ambos comandos van precedidos del binario ejecutable “bmxd” y, a diferencia del resto de parámetros expuestos en este anexo, pueden invocarse en tiempo de ejecución, sin necesidad de detener el servicio de BMXd.

A2) Anexo II

Comparativa entre sistemas de Virtualización

A continuación se describen brevemente las características de los paquetes de software de virtualización más extendidos.

VMware:

Se trata de un software comercial para Windows y Linux que utiliza virtualización completa mediante una capa de abstracción (middleware) que recrea hardware virtual contra el que interactúa la máquina virtual.

Cuenta con básicamente dos familias de producto, las de escritorio como VMware Workstation (de pago) y VMWare Player (gratuito) y las orientadas a servidor, como VMware Server (gratuito) y VMWareESXi/vSphere/vCloud (de pago), especializados en la ejecución y supervisión del estado de pequeñas o grandes granjas de máquinas virtuales.

Todas las versiones cuentan con interfaz gráfica (vía web en el caso de la gama servidor) y un gestor de redes para gestionar las conexiones de las interfaces virtuales.

Virtual Box:

Se trata de un software de código abierto bajo licencias GNU GPL y LGPL para Windows, Linux, Mac OS y OpenSolaris. Al igual que VMWare, utiliza virtualización completaydispone de interfaz gráfica.
Está orientado a su uso en entorno de escritorio.

Qemu:

Es un software liberado bajo licencias GNU GPL y LGPL cuyo sistema de virtualizaciónse basa en la emulación de procesadores mediante traducción dinámica de binarios. De este modo es capaz de ejecutar código para diversas arquitecturas huésped (ARM, SPARC, x86, MIPS, x86_64, etc.) en un sistema anfitrión.

Cuenta con un módulo de aceleración llamado kqemu, también de código abierto, que permite optimizar la ejecución de código inter-plataforma que permite eludir la emulación de las instrucciones ejecutables directamente en la CPU del anfitrión, logrando una mayor eficiencia.

No dispone de interfaz gráfica pero existen proyectos de código abierto (como Qemu Manager) que añaden esta funcionalidad.

Xen:

Xen es otro entorno de virtualización de código abierto con licencia GNU GPL que permite ejecutar múltiples sistemas operativos como Windows, Linux, Solaris o BSD en diversas arquitecturas como x86, x86_64, IA64, ARM, etc.

Su funcionamiento se basa en la paravirtualización, con acceso directo a los recursos del sistema, lo que mejora su rendimiento pero supone una limitación, ya que son necesarias modificaciones ajustes en los huéspedes (por ejemplo, un Kernel específico para Xen, en las máquinas virtuales Linux).

No cuenta con entorno gráfico pero existe una gama comercial de herramientas de gestión e interfaz gráfica desarrolladas por Citrix para el núcleo de Xen.

A modo de resumen, podemos concluir que, de entre las herramientas puramente de código abierto, UML aúna simplicidad y bajo consumo de recursos, los dos ejes fundamentales sobre los que se sustenta la filosofía de NetKit.

A3) Anexo III

Tecnologías compatibles con NetKit

Teniendo en cuenta que los nodos NetKit son sistemas Linux completos, virtualizados en UML, NetKit es capaz de emular el comportamiento de multitud de dispositivos de red, concretamente todos aquellos que se puedan emular con un sistema Linux.

Los más frecuentes son:

- **Máquina Virtual**: Representa una sistema de usuario o un servidor en la que se pueden ejecutar las mismas aplicaciones que en cualquier entorno basado en Linux.
- **Hub Virtual**: NetKit permite unir las interfaces de los diferentes nodos utilizando dominios de colisión compartidos.
- **Switch, Router, Firewall, etc.** : Máquinas virtuales configuradas para comportarse de forma equivalente a su dispositivo de red homólogo en un entorno real.

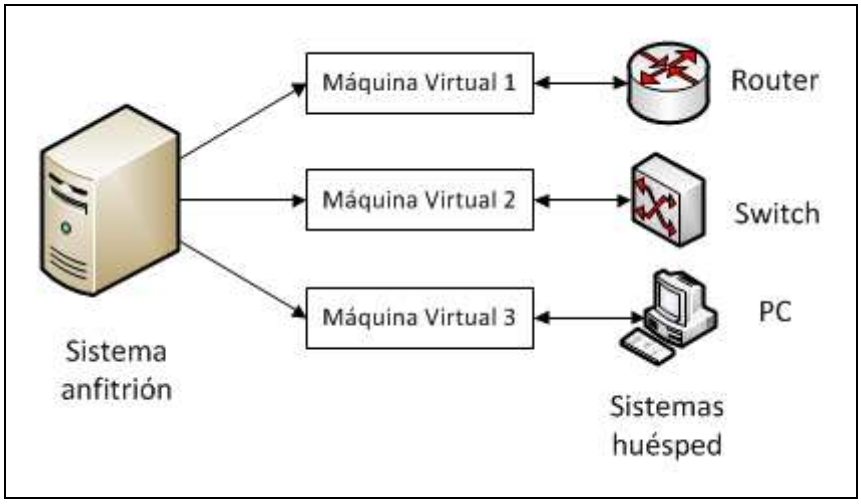


Figura A3-1. Virtualización de dispositivos de red mediante NetKit

Llegados a este punto, resulta evidente que las tecnologías soportadas por NetKit comprenden todas aquellas soportadas por Linux. De forma oficial, NetKit soporta:

Protocolos de red

- Capa física

    - Capa física de Ethernet (incluido en Kernel)
- Capa de enlace
    - 802.1D bridging and Spanning Tree Protocol (brctl)
    - 802.1Q VLAN tagging (vconfig)
    - PPP

- Switching basado en etiquetas
    - Reenvío basado en etiquetas MPLS
    - Distributionde etiquetas mediante LDP

- Capa de red
    - Protocolos de resolución de direcciones ARP y RARP (incluido en Kernel)
    - Mensajes de control ICMP (ping, traceroute)
    - Encaminamiento IPv4 e IPv6 (incluido en Kernel)
- Capa de transporte
    - TCP (incluido en Kernel)
    - UDP (incluido en Kernel)
- Capa de aplicación
    - Autoconfiguración IP por DHCP
    - Servidor y cliente DNS
    - Transferencia de E-mail mediante SMTP, POP, IMAP
    - Servidor y cliente FTP y TFTP
    - Servidor HTTP y HTTPS
    - NFS
    - Telnet
    - Samba
    - SSH
    - Web proxy

Encaminamiento

- Conmutación de etiquetas (MPLS)
- Protocolos de encaminamiento
    - BGP
    - OSPF
    - RIP
    - Balanceo de carga
- Multicast
    - PIM-SM multicast

Seguridad

- IPsec en modo transporte y modo túnel, ESP y AH
- IKE
- IDS, Sistemas de detección de intrusiones

- RADIUS

Manipulación de paquetes

- Encapsulación
  - Túneles GRE
  - Túneles MPLS
- Captura y análisis de paquetes
  - Ettercap
  - Ssldump
  - Tcpdump
  - Tcpreen
  - Tethereal
- Filtrado de paquetes
  - Filtrado y mangling de paquetes mediante Netfilter
- Generación de paquetes
  - Dsniff
  - Hping
  - Sendip
  - Tcpreplay

Y un largo etcétera de aplicaciones, servicios, protocolos y lenguajes de programación y scripting.

Las versiones del Kernel de Linux UML y el sistema de archivos Debian que incorpora de serie NetKit soportan la mayoría de protocolos y aplicaciones compatibles con Linux, y tienen garantizado el soporte de futuras versiones de aplicaciones y protocolos mediante dos vías:

- **A través de los gestores de paquetes.**

  Al tratarse de un sistema Linux Debian virtualizado, cuenta con herramientas de gestión e instalación de paquetes tales como dpkg[8] y aptitude/APT[9].

  El primero nos permitirá instalar cualquier software UML empaquetado para Linux Debian. El segundo nos permitirá simplificar la instalación en múltiples máquinas virtuales, permitiendo obtener los paquetes de software pre compilado desde los repositorios adecuados.

Cuando no existen repositorios o no encontramos fuentes desde donde descargar los paquetes de software requeridos o nos interesa instalar versiones experimentales o desarrollos propios, la opción a seguir es compilar desde el código fuente.

- **Compilación cruzada.**

---

[8] dpkg, Debian Package. http://es.wikipedia.org/wiki/Dpkg

[9] Aptitude. http://es.wikipedia.org/wiki/Aptitude
APT, Advanced Packaging Tool. http://es.wikipedia.org/wiki/Advanced_Packaging_Tool

La opción recomendada para cualquier sistema embebido o virtualizado mediante UML.
Puede compilarse cualquier aplicación, servicio o protocolo para UML desde el anfitrión o desde cualquier otro sistema Linux para ser ejecutado en las máquinas virtuales de los nodos de NetKit.

Existe una tercera vía y, aunque técnicamente posible, se desaconseja en todos los casos pues comporta compilar directamente en el sistema destino.

Cuando tratamos con arquitecturas destino diferentes a las del sistema anfitrión, normalmente hablamos de arquitecturas embebidas, dispositivos móviles o, como en el caso de UML, máquinas virtuales. En cualquiera de los casos, los dispositivos destino tienen unas limitaciones de proceso y memoria (tanto RAM como de almacenamiento) muy significativas, hasta el punto de hacer inviable la compilación y construcción de paquetes de software.

Las compilaciones de pequeños paquetes software que en un ordenador doméstico se completan en el orden de minutos, en un sistema embebido tardarían del orden de días.

Instalación del entorno NetKit

Requerimientos del sistema

Uno de los objetivos de NetKit es el de funcionar en anfitriones con un bajo nivel de recursos y, tal y como reza en su página principal[10] “El sistema del hombre pobre para experimentar con redes de computación”.

Aunque en el desarrollo de este proyecto se ha utilizado un PC con prestaciones superiores, Los requerimientos mínimos de sistema para NetKit son una CPU de más de 600 MHz, suficiente RAM como para las máquinas virtuales y el sistema anfitrión y alrededor de 650 MBytes de espacio en disco.

Proceso de instalación

Comenzamos por descargar los paquetes de software disponibles en la sección de descargas del sitio web de Netkit[11].

```
$> cd /usr/share
```

```
$>wget http://wiki.netkit.org/download/netkit/netkit-2.7.tar.bz2
```

```
$>wget http://wiki.netkit.org/download/netkit-filesystem/netkit-
filesystem-i386-F5.1.tar.bz2
```

```
$>wget http://wiki.netkit.org/download/netkit-kernel/netkit-
kernel-i386-K2.8.tar.bz2
```

[10] Página Principal de NetKithttp://wiki.netkit.org/index.php/Main_Page
[11] Última versión de NetKithttp://wiki.netkit.org/index.php/Download_Official

A continuación comprobamos las sumas MD5 de los archivos descargados[12]

```
$> md5sum netkit-2.7.tar.bz2
>ab1e685730a4cce58c80cc46f85eb57f  netkit-2.7.tar.bz2
```

```
$> md5sm netkit-filesystem-i386-F5.1.tar.bz2
>d140f80a684b092b352db93512cc14ef  netkit-filesystem-i386-
F5.1.tar.bz2
```

```
$> md5sum netkit-kernel-i386-K2.8.tar.bz2
>1b4297abd1c29e6c4563be70f7480562  netkit-kernel-i386-
K2.8.tar.bz2
```

Finalmente descomprimimos y desempaquetamos los archivos

```
$> bunzip2 *.tar.bz2
$> tar –xvf netkit-2.7.tar
$> tar –xvf netkit-filesystem-i386-F5.1.tar.bz2
$> tar –xvf netkit-kernel-i386-K2.8.tar.bz2
```

Ejecutando los anteriores comandos desde /usr/share, los binarios de NetKit y sus ficheros auxiliares quedarán desempaquetados en /usr/share/netkit.

El Kernel de Linux de la máquina virtual de serie quedará alojada en /usr/share/kernel y su sistema de archivos en /usr/share/fs

Finalmente, añadiremos la inicialización de las variables de entorno de NetKit, editando el fichero /etc/profile y añadiendo las siguientes líneas al final:

```
NETKIT_HOME="/usr/share/netkit"
MANPATH="$MANPATH:$NETKIT_HOME/man"
PATH="$PATH:$NETKIT_HOME/bin"
export NETKIT_HOME MANPATH PATH
```

Figura A3-2. Contenido del fichero /etc/profile

A continuación reiniciamos el sistema para que cargar las variables de entorno.

Validación final

Para comprobar que el sistema detecta correctamente la instalación de NetKit, éste incluye el script de chequeo en la raíz del directorio de instalación.

Basta con ejecutarlo para comprobar la correcta instalación de NetKit así como los paquetes externos requeridos y otras dependencias:

```
$> /usr/share/netkit/check_configuration.sh
```

[12] Las sumas MD5 funcionan exclusivamente con la versión de los archivos descargados durante el proyecto. Comprobar su vigencia en http://wiki.netkit.org/index.php/Download_Official

```
>  Checking path correctness... passed.
>  Checking environment... passed.
>  Checking for availability of man pages... passed.
>  Checking for proper directories in the PATH... passed.
>  Checking for availability of auxiliary tools:
          awk         : ok
          basename    : ok
          date        : ok
          dirname     : ok
          find        : ok
          getopt      : ok
          grep        : ok
          head        : ok
          id          : ok
          kill        : ok
          ls          : ok
          lsof        : ok
          ps          : ok
          readlink    : ok
          wc          : ok
          port-helper : ok
          tunctl      : ok
          uml_mconsole : ok
          uml_switch  : ok
passed.
>  Checking for availability of terminal emulator applications:
          xterm         : found
          konsole       : not found
          gnome-terminal : found
passed.

[ READY ] Congratulations! Your Netkit setup is now complete!
EnjoyNetkit!
```

Figura A3-3. Test de conformidad de NetKit

De realizar correctamente los anteriores pasos de la instalación y tener instalados los paquetes software listados como herramientas auxiliares, dará la instalación como válida, tal y como muestra la figura 7-2

En función del sistema de escritorio utilizado y de los paquetes instalados en nuestro sistema, es posible que el test de conformidad no encuentre instalados konsole, gnome-terminal, ninguno de los dos.

Este error es irrelevante siempre y cuando xterm esté instalado, ya que es el cliente de terminal que se utilizará para conectar con las máquinas virtuales. XTerm está presente de serie en la gran mayoría de distribuciones Linux.

A4) Anexo IV

UCI, Unified Configuration Interface

La UCI es, como su nombre indica, una interfaz unificada para la configuración de todos los servicios de un sistema Linux mediante comandos introducidos en el sistema a través del intérprete de comandos.

UCI se ofrece como parte de los paquetes base de OpenWRT (incluida en las versiones posteriores a WhiteRussian: Kamikaze, Backfire y en la versión trunk de desarrollo) y su objetivo es simplificar el proceso de configuración, haciéndolo semejante al de cualquier router comercial de gama alta, cuya configuración IP, protocolos de encaminamiento y demás funcionalidades y servicios están disponibles a través de la misma interfaz de comandos.

Los sistemas Linux adolecen de tener la configuración de sus servicios y aplicaciones atomizada en multitud de ficheros alojados en diversos puntos del sistema de archivos, como /etc/network/interfaces, /etc/dnsmasq.conf o/etc/dhcpd.conf y a menudo utilizan sintaxis diferentes, lo que hace la configuración todavía más heterogénea.

UCI permite unificar estas configuraciones en una serie de ficheros alojados en /etc/config. En la tabla 3-1 se muestran resaltados en negrita los más relevantes para este proyecto.

Fichero	Descripción
/etc/config/bmxd	Configuración del protocolo BMX
/etc/config/dhcp	Configuración de DNSmasq y DHCP
/etc/config/dropbear	Configuración del servidor SSH
/etc/config/firewall	Configuración del firewall
/etc/config/network	Configuración de switching, interfaces de red y routing.
/etc/config/system	Configuraciones misceláneas del sistema
/etc/config/fstab	Configuración de puntos de montaje y memoria swap
/etc/config/mountd	Configuración del demonio de montaje de OpenWRT
/etc/config/ntpclient	Configuración del cliente de ntp
/etc/config/olsrd	Configuración del protocolo OLSR
/etc/config/pure-ftpd	Configuración del servidor Pure-FTPd
/etc/config/qos	Configuración de la Calidad de Servicio
/etc/config/samba	Configuración del servicio de compartición de Microsoft
/etc/config/snmpd	Configuración del servicio de SNMPd
/etc/config/transmission	Configuración del cliente de BitTorrent
/etc/config/uhttpd	Configuración del servidor web (uHTTPd)
/etc/config/wol	Configuración del servicio Wake On LAN.

Tabla A4-1. Muestra de los ficheros UCI de configuración del sistema

Las configuraciones contenidas en estos ficheros pueden leerse y escribirse directamente desde el fichero o, desde consola, a través del comando “uci” en el intérprete de comandos:

```
$>uci
Usage: uci [<options>] <command> [<arguments>]

Commands:
        batch
        export     [<config>]
        import     [<config>]
        changes    [<config>]
        commit     [<config>]
        add<config><section-type>
        add_list<config>.<section>.<option>=<string>
        show       [<config>[.<section>[.<option>]]]
        get<config>.<section>[.<option>]
        set<config>.<section>[.<option>]=<value>
        delete<config>[.<section[.<option>]]
        rename<config>.<section>[.<option>]=<name>
        revert<config>[.<section>[.<option>]]

Options:
        -c <path>  set the search path for config files (default: /etc/config)
        -d <str>   set the delimiter for list values in uci show
        -f <file>  use<file> as input instead of stdin
        -m         when importing, merge data into an existing package
        -n         name unnamed sections on export (default)
        -N         don't name unnamed sections
        -p <path>  add a search path for config change files
        -P <path>  add a search path for config change files and use as default
        -q         quiet mode (don't print error messages)
        -s         force strict mode (stop on parser errors, default)
        -S         disable strict mode
        -X         do not use extended syntax on 'show'
```

Figura A4-1. Salida del comando uci en el intérprete de comandos de OpenWRT

Como etapa final en la configuración, es posible aplicar los cambios de cualquiera de las configuraciones sin reiniciar el sistema.

Esto favorece la flexibilidad a la hora de realizar pruebas en vivo, consiguiendo aplicar los cambios de forma instantánea y sin afectar al resto de elementos del sistema.

Tomando como ejemplo la configuración IP de una tarjeta de red, cuya configuración inicial se muestra en la tabla 3-2:

#> cat /etc/config/network	#>uci show network.lan0
config interface lan0	**network.lan0=interface**

option ifname eth0 **option proto static** **option ipaddr 192.168.11.1** **option netmask 255.255.255.0** **option gateway 192.168.11.254** **option dns 8.8.8.8**	**network.lan0.ifname=eth0** **network.lan0.proto=static** **network.lan0.ipaddr=192.168.11.1** **network.lan0.netmask=255.255.255.0** **network.lan0.gateway=192.168.11.254** **network.lan0.dns=8.8.8.8**
#>ifconfig eth0 **eth0** Link encap:EthernetHWaddr BE:AC:CB:F5:AC:86 **inet addr:192.168.11.1 Bcast:192.168.11.255 Mask:255.255.255.0** UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 RX packets:0 errors:0 dropped:0 overruns:0 frame:0 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B) Interrupt:5	

Figura A4-2. Configuración de red UCI

Puede cambiarse cualquier parámetro de su configuración con sólo 3 comandos.

Comando UCI para fijar un nuevo valor para la dirección IP de la interfaz denominada como lan0 (eth0):

```
#>uci set network.lan0.ipaddr=192.168.11.2
```

Comando UCI para aplicar los cambios realizados:

```
#>ucicommit
```

Reinicio de los servicios de red. Reiniciará las interfaces de red recargando las configuraciones modificadas mediante UCI:

```
#> /etc/init.d/network restart
```

Llegados a este punto, todas las configuraciones modificadas a través de la interfaz UCI, cuyos servicios se hayan reiniciado (como el de network) estarán disponibles en el sistema.
Lo mismo ocurre con las configuraciones del resto de servicios y protocolos (httpd, bmxd, olsrd, etc…).

A5) Anexo V

Trabajo con entorno NetKit

En este capítulo se aborda la configuración de un escenario de red sencillo basado en máquinas virtuales con las imágenes oficiales del Kernel de Linux UML y del sistema de archivos que acompañan a NetKit.

En el Anexo III se incluye información adicional sobre los requisitos del sistema y el procedimiento detallado para instalar el entorno NetKit en Linux.

Creación de Máquinas Virtuales

Una vez realizada y comprobada la instalación de NetKit, procedemos a comprobar el correcto arranque de las máquinas virtuales.

El comando vstart permite crear y/o iniciar máquinas virtuales. Sus parámetros más relevantes son:

- --ethN: Permite añadir interfaces de red a la máquina virtual desde N=0 en adelante y vincular dichas interfaces a dominios de colisión de NetKit o a interfaces tap para interconectar anfitrión y huésped.
- -k: Permite especificar la imagen de kernel para el sistema huésped. Debe ser un Kernel Linux compilado para UML sin compresión y con permisos de ejecución.
- -M: Configura la asignación de memoria RAM. De no especificarse, la reserva por defecto es de 32 MBytes.
- -f: Permite especificar el sistema de archivos para el sistema huésped.
- --xterm: Permite seleccionar el terminal que utilizará NetKit como interfaz de comandos con la máquina virtual.

Para crear una nueva máquina virtual, con las imágenes de Kernel y sistema de archivos por defecto, ejecutamos el siguiente comando

```
$>vstart PFC_test –eth0=X –M 64
```

NetKit creará una nueva máquina virtual etiquetada como “PFC_test” con 64 MBytes de RAM del anfitrión y una interfaz de red ethernet presentada al huésped como eth0 y conectada al dominio de colisión “A”.

Paralelamente, el comando vstart, muestra información detallada sobre la máquina virtual ejecutada, a través de la consola del anfitrión:

```
user@mint ~ $ vstartPFC_test --eth0=A –M 64
============= Starting virtual machine "PFC_test" =============
   Kernel:     /usr/share/netkit/kernel/netkit-kernel
   Modules:    /usr/share/netkit/kernel/modules
   Memory:     64 MB
   Model fs:   /usr/share/netkit/fs/netkit-fs
Filesystem: /home/user/PFC_test.disk (new)
   Interfaces: eth0 @ X   (/home/user/.netkit/hubs/vhub_user_X.cnct)
Hostfs at:  /home/user

Running ==> /usr/share/netkit/bin/uml_switch -hub -unix
/home/user/.netkit/hubs/vhub_user_X.cnct</dev/null 2>&1
Running ==>xterm -e /usr/share/netkit/kernel/netkit-kernel
modules=/usr/share/netkit/kernel/modules name=PFC_test
title=PFC_testumid=PFC_testmem=64M
ubd0=/home/user/PFC_test.disk,/usr/share/netkit/fs/netkit-fs root=98:1
uml_dir=/home/user/.netkit/mconsole
eth0=daemon,,,/home/user/.netkit/hubs/vhub_user_X.cncthosthome=/home/user quiet
con0=fd:0,fd:1 con1=null SELINUX_INIT=0
```

Figura A5-1. Salida del comando vstart, en el arranque de una máquina virtual

Como podemos observar en la figura anterior, vstart muestra todos los recursos de la máquina virtual. A continuación se repasan los más relevantes:

- **Kernel**: Imagen del Kernel de Linux UML utilizado
- **Memory**: Memoria RAM asignada por el anfitrión
- **Modelfs**: Imagen del sistema de archivos tomada como referencia
- **Filesystem**: muestra la ubicación del archive COW con el sistema de archivos, así como si se trata de una máquina virtual ya existente o si lo crea por primera vez (new).
- **Interfaces**: Muestra las interfaces de red de la máquina virtual y el dominio de colisión asociado (“B” en el caso del ejemplo en la figura 7-3)

Tras ejecutar el comando vstart, NetKit lanzará una ventana con el terminal por defecto, XTerm, que utilizaremos como interfaz de comandos para la máquina virtual.

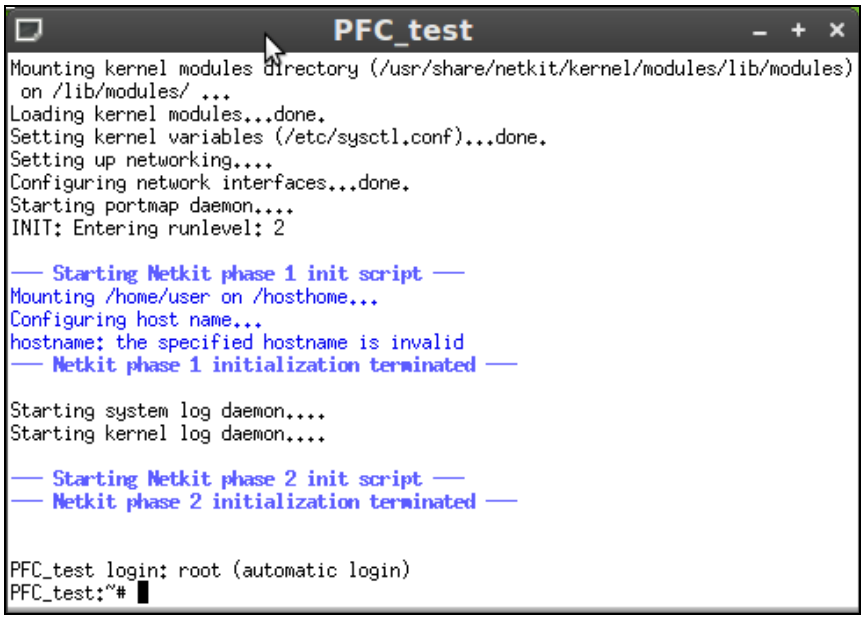


Figura A5-2. XTerm muestra el arranque de una máquina virtual

Como podemos observar en la figura 7-4, el terminal muestra los mensajes de sistema y debug a lo largo del proceso de carga del Kernel, hasta la carga final del intérprete de comandos, que queda a la espera.

Para apagar el sistema huésped, basta con ejecutar

```
#>poweroff
```

Desde el terminal de la máquina virtual, o bien

```
$>vhaltPFC_test
> Halting virtual machine "PFC_test" (PID 3707) owned by user
[...   ]
```

Desde la consola del anfitrión.
Puede re-arrancarse de nuevo la misma máquina virtual, repitiendo el comando vstart.

En la primera ejecución de una nueva máquina virtual, NetKit crea un fichero con la extensión ".disk" con el mismo nombre que la máquina virtual.

```
$>ls -lh *.disk
> -rw-r--r-- 1 user user 11G 2010-12-09 16:01 PFC_test.disk
```

Figura A5-3. archivo COW creado por NetKit.

Este fichero es el archivo COW del sistema de archivos de la máquina virtual y, como podemos advertir en la figura 7-5, su tamaño es semejante al de la imagen del sistema de archivos, alojada en /usr/share/netkit/fs.

Contiene todos los cambios efectuados en la máquina virtual y es imprescindible conservarlo si se pretende volver a arrancar la máquina virtual con los últimos cambios realizados.

A pesar de reportar un tamaño de 11 GBytes, el archivo COW contiene mucha menos información, pero el espacio restante se encuentra completado con ceros hasta igualar el tamaño de su imagen correspondiente, por lo que es altamente comprimible.

Si creamos un archivo comprimido, podremos observar que el nuevo tamaño se corresponde con el espacio realmente ocupado en el fichero COW.

```
$>tar -zcvf PFC_test.tar.gz PFC_test.disk
>PFC_test.disk
$>ls -lh PFC_test.*
> -rw-r--r-- 1 user user 11G 2010-12-09 16:01 PFC_test.disk
> -rw-r--r-- 1 user user 10M 2010-12-09 16:18 PFC_test.tar.gz
```

Figura A5-4. Fichero COW Comprimido.

En el momento del arranque de la máquina virtual, con los paquetes instalados de base, la imagen COW comprimida ocupa tan solo 10 MBytes, como puede observarse en la figura 7-6

Puede encontrarse más información sobre los archivos COW en el capítulo 4.4.

Creación de redes virtuales

Con el fin de probar las funciones de red con NetKit, probaremos un escenario de red sencillo formado por los elementos representados en la siguiente figura:

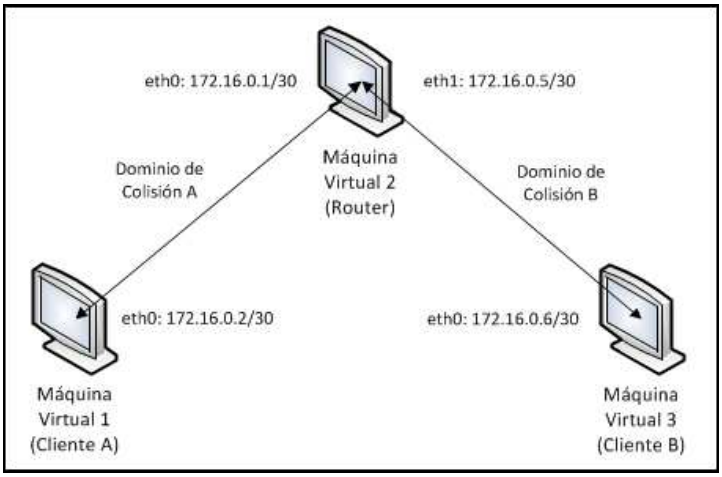


Figura A5-5. Ejemplo básico de red.

Cada dominio de colisión interconectará las interfaces de red de los clientes con una de las interfaces del Router y se crearán rutas de forma manual en los clientes para que puedan acceder conexiones extremo a extremo.

Mediante el comando vstart creamos 3 máquinas virtuales con 32 MBytes de RAM y las interfaces de red pertinentes en cada dominio de colisión:

```
$>vstartCliente_A --eth0=A
$>vstartCliente_B --eth0=B
$>vstart Router --eth0=A --eth1=B
```

Tras configurar con ifconfig las IPs de la figura 7-7 a través de los terminales XTerm de cada máquina virtual, añadimos las rutas necesarias en cada cliente y verificamos la comunicación extremo a extremo a través del router.

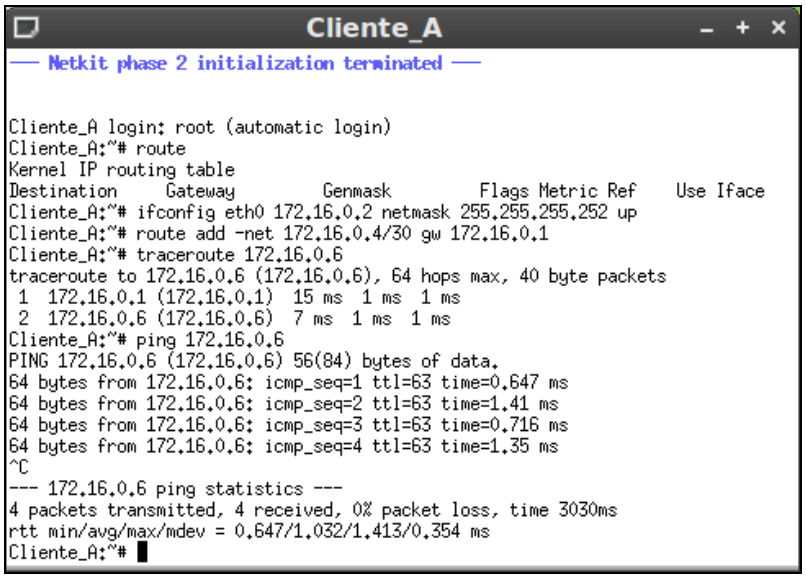


Figura A5-6. Configuración de IP encaminamiento y prueba ICMP

Como puede observarse en la figura 5, la configuración y posterior prueba de conectividad entre las máquinas virtuales se realiza de forma sencilla, utilizando las mismas herramientas que en un sistema “real”.