Universidad Complutense de Madrid
Facultad de Informática


# BGP-Wedgies: configuración en un entorno emulado

Proyecto Sistemas Informáticos
Curso 2009-2010

*Autor:*
Mercedes Bernal Pérez

*Tutores:*
Juan Carlos Fabero
Pedro Andrés Aranda Gutiérrez

**D. Juan Carlos Fabero**, Profesor Titular de la Universidad Complutense de Madrid del Grupo de Arquitectura y Tecnología de Computadores, como tutor académico y **D. Pedro Andrés Aranda Gutiérrez**, Especialista Tecnológico en Redes y Protocolos de Red, como co-tutor en Telefónica Investigación y Desarrollo

CERTIFICAN:

Que el trabajo titulado *"BGP-Wedgies: configuración en un entorno emulado"* ha sido realizado satisfactoriamente por **Dña. Mercedes Bernal Pérez**, con D.N.I. 71031656 C y constituye la memoria que presenta para optar a la superación de la asignatura Sistemas Informáticos de la titulación Ingeniería en Informática en la Facultad de Informática de la Universidad Complutense de Madrid.

Madrid, a 14 de septiembre de 2010.

Dr. D. Juan Carlos Fabero

D. Pedro A. Aranda Gutiérrez

*A mis dos tutores, amigos y familia, y en especial a ti...*

*Resumen.-* Entre la amplia variedad de protocolos de encaminamiento que se pueden encontrar hoy en Internet, BGP es el protocolo de facto para el encaminamiento entre dominios. Idealmente se diseñó para permitir a cada dominio elegir la mejor ruta dadas las alternativas propuestas por sistemas vecinos. Desafortunadamente, al igual que en otras ramas de la informática, muchos agentes que persiguen de forma independiente un óptimo local no siempre convergen en un óptimo global. En particular, se ha estudiado una clase de configuraciones para la que hay más de un resultado potencial que BGP puede seleccionar, de forma no determinista, y donde los estados destino distintos a los previstos son igualmente estables. Para este fin se ha realizado una modificación de Quagga que permite almacenar la información de encaminamiento en formato comprimido. Además, se ha desarrollado otra modificación para aprovechar las ventajas de *multihoming* mediante la ejecución dos demonios BGP en parelelo y así poder evitar este tipo de comportamientos no deseados.

*Palabras clave*: **BGP, Quagga, Netkit, Wedgies, Inestabilidad, Multihoming, Ingeniería de tráfico**

*Abstract.-* Among the wide variety of routing protocols that can be found today in the Internet, BGP is the de facto interdomain routing protocol. Ideally it was designed to let each domain choose the best route given the alternatives proposed by neighboring systems. Unfortunately, as it is in other branches of computer science, many agents that independently pursue a local optimum do not always converge into a global optimum. In particular, it has been studied a class of configurations for which there is more than one potential outcome that BGP may select, in a non-deterministic manner, and where forwarding states other than the intended state are equally stable. For this purpose, it has been done a Quagga modification to be able to store routing information in compressed format. Moreover, it has been developed another modification to take multihoming advantages by running two BGP daemons at the same time and so avoid these undesired behavior.

*Keywords*: **BGP, Quagga, Netkit, Wedgies, Inestability, Multihoming, Traffic Engineering**

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

Las redes de comunicación han alcanzado gran tamaño y complejidad hoy en día. Internet, que nació como una red experimental de conexión de un puñado de institutos de investigación voluntarios, ha crecido hasta convertirse en un enorme sistema distribuido de interconexión de más de 700 millones de máquinas [51]. El protocolo estándar para el **encaminamiento** entre dominios en Internet es BGP-4. Debido a la escala, a la heterogeneidad y a la autonomía de Internet, las rutas entre dominios se calculan usando complejas políticas proporcionadas localmente en cada red, con poca coordinación global. Se ha demostrado que la interacción de estas políticas puede producir **anomalías** globales, por ejemplo, oscilaciones del protocolo o encaminamientos no deterministas, como BGP-Wedgies, objeto de estudio del primero de los dos escenarios implementados.

Ligado a este auge, multitud de corporaciones publican en Internet aplicaciones críticas para el ejercicio de su actividad empresarial que requieren mayor tolerancia a fallos, lo cual no puede garantizarse con un único operador, sino que es necesario una conexión mediante múltiples operadores, surgiendo el concepto de **multihoming**, objeto de estudio del segundo escenario.

Con este rápido crecimiento de Internet y el incremento de demandas de accesos, muchas organizaciones han tenido o tienen dificultades para cubrir las necesidades de hardware y software que requieren los servicios demandados. En consecuencia, para afrontar la adquisición de dispositivos de red para la conexión a Internet ha surgido gran cantidad de software de encaminamiento que brinda la posibilidad de convertir equipos en encaminadores, lo cual supone un ahorro en costes considerable. En concreto, se ha elegido el paquete software Quagga, el cual es de libre distribución y se ejecuta bajo

el sistema operativo Linux. Además, presenta la ventaja de que permite un aprendizaje importante sobre la configuración de encaminadores y puesta en práctica del encaminamiento dinámico, ya que los comandos utilizados para la configuración de los protocolos de encaminamiento son muy similares a que los que se utilizan para configurar equipos encaminadores de proveedores como Cisco.

Por último, de forma paralela para ahorrar costes, las herramientas software de emulación han evolucionado permitiendo facilitar la implementación y el análisis de sistemas de comunicación cada vez más complejos, como es el caso de Netkit.

El presente proyecto *BGP-Wedgies: configuración en un entorno emulado* pretende dar una visión realista de las distintas posibilidades prácticas que existen en el mercado para hacer uso del protocolo BGP-4, particularmente la técnica de multihoming, y además, las posibles anomalías que se pueden producir como consecuencia de la ingeniería de tráfico realizada. Para ello, se han implementado dos escenarios en los cuales se explica la configuración realizada y se analizan los resultados obtenidos tras monitorizar las tablas de encaminamiento y los mensajes capturados.

Se ha estructurado en seis capítulos. En primer lugar se hace un estudio teórico del protocolo de encaminamiento de pasarela externa BGP-4, así como de los atributos que lo caracterizan y de las diferentes funcionalidades que soporta. Posteriormente, se introduce el software de emulación utilizado para la implementación de los escenarios, que ocupan los dos últimos capítulos, previa explicación del software de encaminamiento elegido.

- Capítulo 1: Introducción
- Capítulo 2: BGP-4: Border Gateway Protocol
- Capítulo 3: Software de emulación Netkit
- Capítulo 4: Software de encaminamiento Quagga
- Capítulo 5: Escenario BGP-Wedgies

- Capítulo 6: Escenario Multihoming

# Capítulo 2

# BGP-4: Border Gateway Protocol

## 2.1. Introducción

Durante los últimos años Internet se ha convertido en un componente crítico de nuestra infraestructura de comunicación. La mayor parte de la comunicación en Internet está basada en transferencias de datos sobre conexiones entre pares de máquinas. Para este propósito los datos son divididos en pequeños paquetes de datos, cada uno de los cuales cruza Internet independientemente del resto. La mayoría de las veces, el dato no es transferido a través de una conexión física directa entre emisor y receptor. En su lugar, los paquetes viajan a través de intermediarios denominados **encaminadores**, los cuales tienen que decidir para cada paquete de datos a través de qué encaminador vecino lo envían para que llegue a su destinatario final.

Los encaminadores tienen conocimiento sobre los posibles destinatarios. Tienen que saber dónde está situado el destinatario, si es alcanzable y con qué camino. Por eso, cuando la topología cambia, el encaminador tiene que ser informado tan pronto como sea posible para tomar una decisión de encaminamiento. El tiempo que tardan las tablas de rutas de los encaminadores en estar en un estado de uniformidad que refleje la situación real después de un cambio en la topología es el **tiempo de convergencia**. Su duración es un interés crítico de la estabilidad de Internet.

Dado que Internet abarca un gran número de redes, el encaminamiento es una tarea compleja. Una parte de esta tarea es realizada por **BGP-4** [76],

utilizado por los encaminadores de las diferentes redes para intercambiar lo que se denomina información de alcance. BGP-4 suele mostrar un tiempo de convergencia del orden de minutos y reacciona dinámicamente a los cambios de topología. Desafortunadamente, la convergencia puede retrasarse por factores desconocidos. Trabajos recientes de la comunidad de investigación [28] [45] [42] [44] [57] [58] [59] [60] [68] [82] [83] han demostrado que las dinámicas del protocolo son poco conocidas.

En general, BGP-4 es difícil de analizar debido a su complejidad, al tamaño de Internet de hoy y a su carácter distribuido.

## 2.2. Algunos conceptos básicos

Internet se compone de muchas redes heterogéneas. Los nodos pueden conectarse a través de diversos medios físicos, cada uno con su propio método de transportar mensajes. Las redes pueden comunicarse entre sí, siempre y cuando los nodos y los medios de conexión física que los conectan compartan una interfaz común **Internet Protocol (IP)** (Protocolo de Internet) [72].

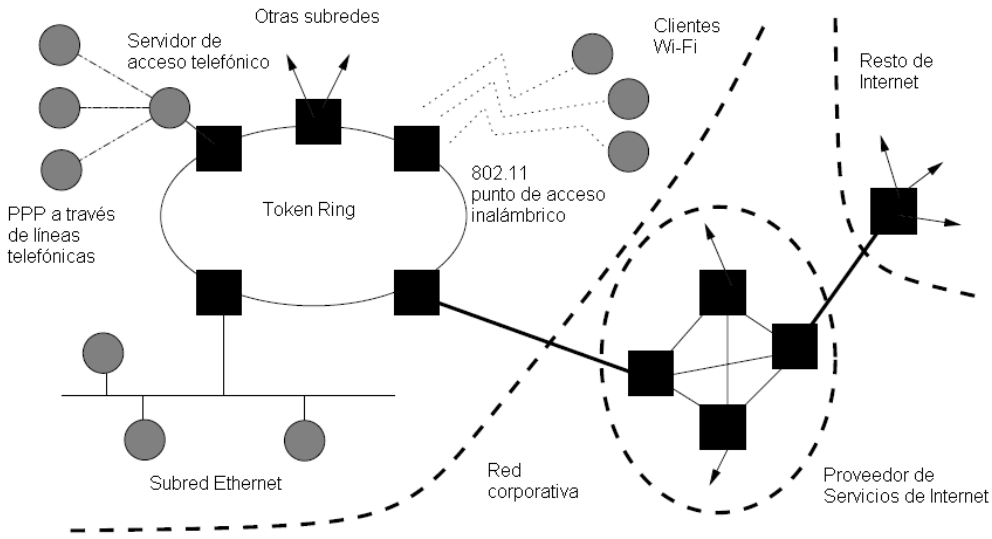



Figura 2.1: Internetworking

Los paquetes en los que se dividen los datos para ser enviados a través de

Internet se denominan datagramas. IP es el protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados, como es Internet. Las cabeceras IP contienen las direcciones de las máquinas de origen y destino (direcciones IP), direcciones que serán usadas por los encaminadores para decidir el tramo de red por el que reenviarán los paquetes.

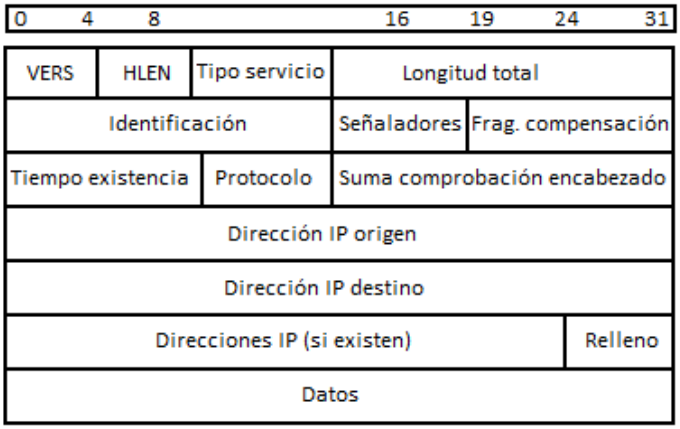



Figura 2.2: Campos de la capa de red

Quizás los aspectos más complejos de IP son el direccionamiento y el encaminamiento. El encaminamiento es el mecanismo por el que en una red los paquetes de información se hacen llegar desde su origen a su destino final, siguiendo un camino o ruta a través de la red. En una red grande o en un conjunto de redes interconectadas el camino a seguir hasta llegar al destino final puede suponer transitar por muchos nodos intermedios. Por ello, Internet se divide en un gran número de diferentes regiones bajo un control administrativo autónomo, los denominados Sistemas Autónomos o **ASs**[48], los cuales deciden la topología de red, los protocolos de encaminamiento para su infraestructura y cómo el tráfico viaja a través de dicha infraestructura.

Por tanto, un Autonomous System (AS) (Sistema Autónomo) es un conjunto de redes, o de encaminadores, que tienen una única política de encaminamiento y que se ejecuta bajo una administración común, utilizando habitualmente un único Interior Gateway Protocol (IGP) (Protocolo de Pasarela Interno). Para el mundo exterior, el AS es visto como una única entidad.

Cada AS tiene un **identificador** [79], que se le asigna mediante un Regional Internet Registry (RIR) (Registro Regional de Internet) (como RIPE,

ARIN, o APNIC, ver Figura 2.3), o un Internet Service Provider (ISP) (Proveedor de Servicios de Internet) en el caso de los ASs privados.

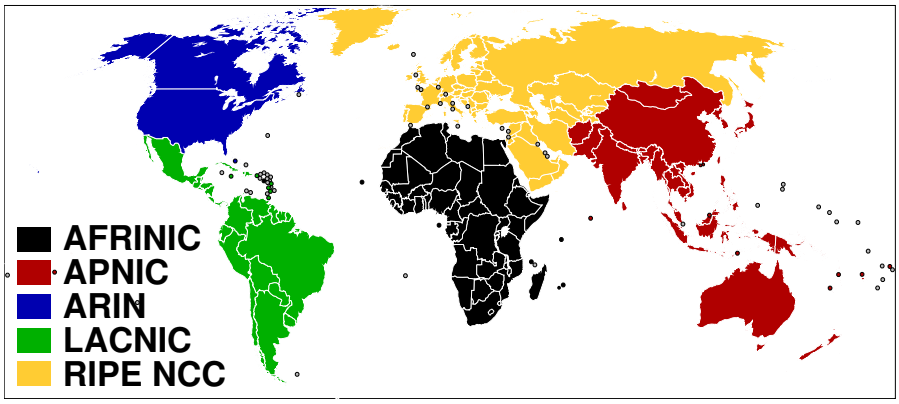



Figura 2.3: Mapa mundial de los Registros de Internet Regionales

Inicialmente, los identificadores de ASs eran de 16 bits [39]. Debido al agotamiento de este espacio de direccionamiento, el 1 de enero de 2009, Réseaux IP Européens (RIPE), Centro de Coordinación de redes IP europeas, comenzó la asignación de números de 32 bits [40] (o de cuatro bytes) de sistemas autónomos de forma predeterminada. Esto es un acuerdo con una política común acordada en todos los RIR y descrito en el documento [79], para evitar problemas de agotamiento del número de ASs. En la Figura 2.4[38] se observa el incremento del número de ASs respecto al tiempo.

En consecuencia, se añadieron ciertas capacidades a BGP-4 para soportar los números de ASs de cuatro octetos [95]. Más concretamente, se introducen dos nuevos atributos, `AS4_PATH` y `AS4_AGGREGATOR`, que pueden ser utilizados para propagar información del camino de ASs de cuatro octetos a través de los encaminadores de BGP-4 que no admiten los números de AS de cuatro octetos.

## 2.3. Protocolos de encaminamiento

Los protocolos de encaminamiento son el conjunto de reglas utilizadas por un encaminador cuando se comunica con otros encaminadores con el fin de compartir información de encaminamiento. Dicha información se usa para construir y mantener las tablas de encaminamiento. Por consiguiente,

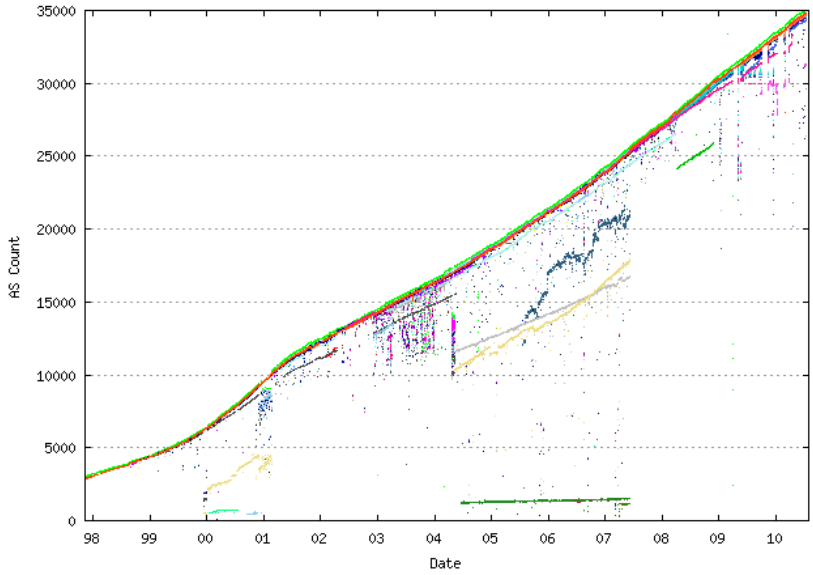



Figura 2.4: Evolución del número de AS

no todos los encaminadores trabajan de la misma manera, depende del tipo de red en la que se encuentren.

Existen dos grandes tipos de protocolos de encaminamiento, Figura 2.5. En primer lugar, los que se usan para intercambiar información de encaminamiento dentro de un AS, es decir, un encaminador habla con otro que tiene el mismo número de AS, se conocen como **protocolos interiores** o IGP [47]. Entre ellos destacan Routing Information Protocol (RIP) [64], Open Shortest Path First (OSPF) [69] e Intermediate System to Intermediate System (ISIS) [70]. En segundo lugar, los que se usan para intercambiar información de encaminamiento entre sistemas autónomos, es decir, entre encaminadores con diferente número de AS, se conocen como **protocolos exteriores** o Exterior Gateway Protocol (EGP) [80] y en la actualidad, el único ejemplo es BGP-4.

### 2.3.1. Protocolos IGP y EGP

El objetivo de un protocolo **IGP** es establecer un conjunto de las mejores rutas coherentes en cada encaminador de destino intradominio. Por coherencia, se entiende que todos los subcaminos de una mejor ruta elegida

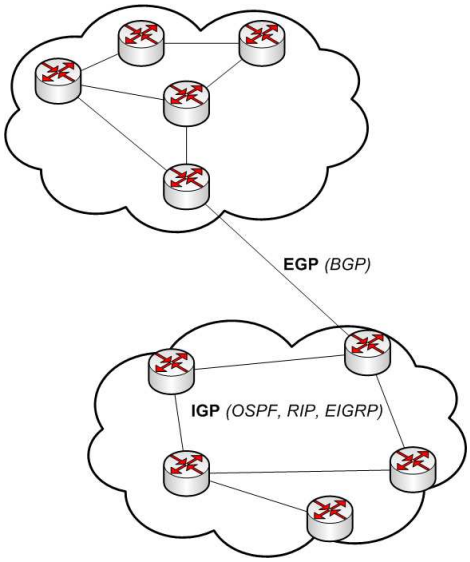



Figura 2.5: Protocolos de encaminamiento IGP y EGP

es también una mejor ruta elegida (la incoherencia puede provocar bucles de reenvío IP). Debido a que los dominios operan bajo la misma autoridad administrativa se suelen aplicar generalmente las siguientes hipótesis:

1. Los dominios son por lo general lo suficientemente pequeños para que la información de estado acerca de los enlaces de red pueda ser difundida por inundación, dando a cada encaminador la capacidad para calcular la topología de la red.

2. Los dominios tienen la misma noción de ”mejor”, que a menudo es más corto (menor número de saltos) o de más bajo costo (suponiendo que los bordes se le puede asignar un costo de tránsito, por ejemplo, lo que corresponde a la congestión o la distancia).

Por estas razones, algunos de los protocolos IGP más comunes, por ejemplo, OSPF, son protocolos de estado de enlace, como se verá en la Sección 2.3.3.

En un protocolo **EGP**, el encaminamiento a nivel inter-dominio es más complejo debido a la escala de Internet y la autonomía deseada por los administradores en la configuración de rutas. Una vez que el paquete se reenvía a

otro dominio, ese dominio tiene el control total sobre la ruta y su trayectoria futura.

Cuando está disponible más de una ruta a un destino, la elección de la mejor ruta depende de la configuración del encaminador local en materia de política; los tipos de políticas dependerán del protocolo específico y el hardware, y con frecuencia son bastante expresivos, limitados sólo por el lenguaje de configuración proporcionado por los vendedores del encaminador. Los mensajes EGP no contienen información acerca de las rutas utilizadas dentro de un dominio. Esto permite a un dominio preservar su autonomía con respecto a las rutas internas y mantener su topología de la red privada de otros. Por lo tanto, los protocolos EGP suelen ser protocolos de vector distancia, cuyo nombre proviene del mecanismo que se utiliza para evitar bucles.

Para el protocolo EGP, el estándar de facto es BGP-4. En el límite de cada sistema autónomo, los llamados encaminadores de borde o frontera intercambian información de encaminamiento usando BGP-4.

### 2.3.2. Protocolos de encaminamiento estático y dinámico

Básicamente existen dos formas de encaminamiento a otros nodos fuera del nodo local: utilizando encaminamiento estático o encaminamiento dinámico. Cada método tiene ventajas e inconvenientes, pero cuando una red crece, finalmente el encaminamiento dinámico es la única manera factible de gestionar la red.

En los protocolos de encaminamiento **estáticos**, el administrador configura manualmente las entradas de la tabla de rutas. Presenta las ventajas de tener más control pero, sin embargo, presenta limitaciones como poca escalabilidad y lentitud en la adaptación a los fallos de red. Pueden ser útiles para mantener las tablas de rutas cuando sólo hay una ruta o camino a una red de destino en particular. El otro tipo son los protocolos de encaminamiento **dinámicos**: los encaminadores intercambian información de alcance de la red usando protocolos de encaminamiento para calcular las mejores rutas. Como ventajas, se pueden adaptar rápidamente a los cambios en la topología de la red y presentan buena escalabilidad. Como contrapartida, los algoritmos distribuidos son complejos y presentan mayor consumo de CPU, ancho de banda y memoria.

| | **Encaminamiento dinámico** | **Encaminamiento estático** |
|---|---|---|
| **Complejidad configuración** | Independiente del tamaño de la red | Se incrementa con el tamaño de la red |
| **Conocimientos requeridos** | Se requiere conocimiento avanzado | No se requieren conocimientos adicionales |
| **Cambios de topología** | Se adapta automáticamente a los cambios de la topología | Se requiere la intervención del administrador |
| **Escalabilidad** | Adecuado para topologías simples y complejas | Adecuado para topologías simples |
| **Seguridad** | Es menos seguro | Más seguro |
| **Uso de recursos** | Utiliza CPU, memoria y ancho de banda de enlace | No se requieren recursos adicionales |
| **Capacidad de predicción** | La ruta depende de la topología actual | La ruta hacia el destino es siempre la misma |

Tabla 2.1: Comparativa protocolos de encaminamiento estático y dinámico

En la Tabla 2.1 se pueden comparar las características de ambos protocolos.

### 2.3.3. Protocolos de estado enlace y vector distancia

Los protocolos de **vector distancia**, como RIP versión 1, fueron principalmente diseñados para topologías de red pequeñas. El término "vector distancia" deriva del hecho de que el protocolo incluye en sus actualizaciones de encaminamiento un vector de distancias (número de saltos), requiriendo que cada nodo calcule por separado la mejor ruta para cada destino. Este tipo de protocolos determinan la dirección y la distancia hacia cualquier enlace de la red. Normalmente cuanto menor es este valor, mejor es la ruta.

Mediante el uso del número de saltos, los protocolos de vector distancia no tienen en cuenta la sobrecarga de envío de información a través de un enlace específico. Enlaces de baja velocidad son tratados por igual o, a veces de forma preferente, a un enlace de alta velocidad, en función del número de saltos calculados para llegar a su destino. Esto daría lugar a comportamientos de encaminamiento subóptimos e ineficientes.

Debido generalmente a cambios en la topología, se pueden producir bucles de encaminamiento que producen entradas de encaminamiento incoherentes. Si un enlace de un encaminador A se vuelve inaccesible, los encaminadores vecinos no se dan cuenta inmediatamente, por lo que se corre el riesgo de que el encaminador A crea que puede llegar a la red perdida a través de sus vecinos que mantienen entradas antiguas. Así, añade una nueva entrada a su tabla de encaminamiento con un coste superior. A su vez, este proceso se repetiría una y otra vez, incrementándose el coste de las rutas, hasta que de alguna forma se detenga dicho proceso. Algunos de los métodos utilizados para evitar este comportamiento son los que siguen:

1. **Horizonte Dividido**. No permite a los encaminadores anunciar las redes en la dirección en la que se aprendieron, por eso reduce el tiempo de convergencia.

2. **Envenenamiento de rutas**. Cuando una red de un encaminador falla, este envenena su enlace creando una entrada para dicho enlace con coste infinito. Cuando los encaminadores vecinos ven que la red ha pasado a un coste infinito, envían una actualización inversa indicando que la ruta no está accesible.

3. **Definición de Máximo**. Cada vez que la ruta pasa por un encaminador incrementa el número de saltos en uno. En el protocolo RIP, un destino a una distancia mayor que 15 se considera inalcanzable.

4. **Temporizadores**. Los temporizadores hacen que los encaminadores no apliquen ningún cambio que pudiera afectar a las rutas durante un periodo de tiempo determinado. Si llega una actualización con una métrica mejor a una red inaccesible, el encaminador se actualiza y elimina el temporizador. Si no recibe cambios óptimos dará por caída la red al transcurrir el tiempo de espera.

Otra característica es la forma en que se intercambia la información de encaminamiento. Los algoritmos de vector distancia trabajan sobre el concepto de que los encaminadores intercambian las tablas de ruta a través de difusiones periódicas. Sin embargo, protocolos más evolucionados como RIP versión 2, son capaces de anunciar rutas por multidifusión. Las actualizaciones regulares entre encaminadores comunican los cambios en la topología; por tanto, visualizan la red desde la perspectiva de los vecinos.

| **Vector distancia** | **Estado de enlace** |
|---|---|
| Vista de la topología de la red desde la perspectiva del vecino | Se incrementa con el tamaño de la red |
| Añade vectores de distancias de encaminador a encaminador | Calcula la ruta más corta hasta otros encaminadores |
| Frecuentes actualizaciones periódicas, convergencia lenta | Actualizaciones activadas por eventos, convergencia rápida |
| Pasa copias de la tabla de encaminamiento a los encaminadores vecinos | Pasa las actualizaciones de encaminamiento de estado del enlace a los otros encaminadores |
| Poca carga computacional | Mayor carga computacional |

Tabla 2.2: Comparativa protocolos de vector distancia y estado de enlace

El otro tipo de protocolos, de **estado de enlace**, como OSPF e ISIS, son protocolos de encaminamiento más avanzados que han solventado las deficiencias de los protocolos de vector distancia. En la Tabla 2.2 se puede comparan las características de ambos. En este tipo de protocolos, los encaminadores intercambian elementos de información, llamados estados de enlace, que llevan información sobre los enlaces y nodos. Esto significa que los encaminadores no intercambian las tablas de encaminamiento. Su métrica se basa en el retardo, ancho de banda, carga y confiabilidad, de los distintos enlaces posibles para llegar a un destino. En base a esos parámetros, el protocolo prefiere una ruta sobre otra. Por tanto, según los encaminadores reciben el estado de enlace que contiene el estado de los vínculos a los encaminadores vecinos, pueden actualizar su vista de la topología de la red, que puede ser representada como un grafo ponderado, de forma que recrean la topología exacta de toda la red. Todas estas características derivan en una mejor convergencia, debido a que el ancho de banda de los enlaces y los retardos se tienen en cuenta cuando se calcula el camino más corto a un destino.

Aunque los algoritmos estado de enlace han facilitado una mejor escalabilidad de encaminamiento, lo que les permite ser utilizados en más grandes y complejas topologías, todavía deben limitarse a los de encaminamiento in-

terior.

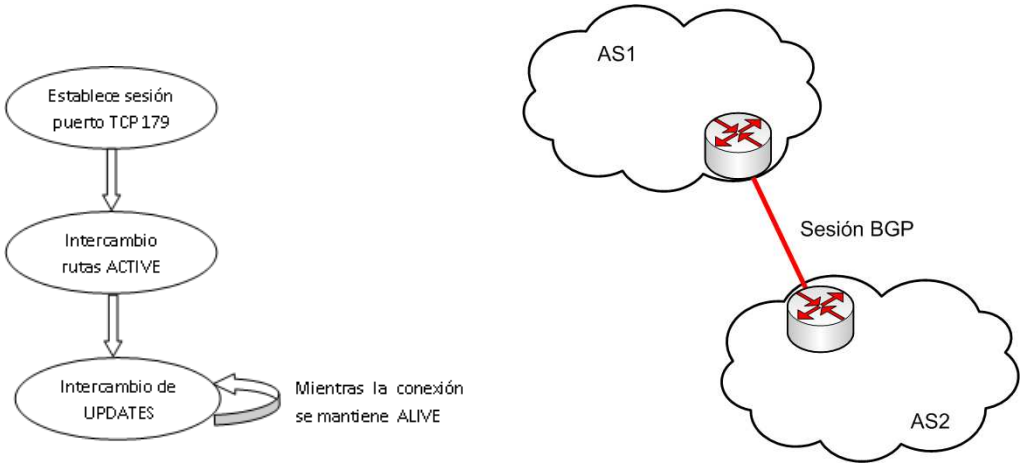
## 2.4. Atributos y proceso de selección de rutas en bgp

BGP-4 es una variante de la clase de los protocolos de vector distancia. En lugar de distribuir la información de costos propaga información de ruta completa a cada destino a fin de evitar los bucles. Si dos sistemas autónomos desean intercambiar tráfico, establecen una **sesión BGP-4** entre dos encaminadores; llamados BGP-4 *peers*. Se utiliza el protocolo Transmission Control Protocol (TCP) (Protocolo de Control de Transmisión) [73] como base de mecanismo de transporte fiable y el puerto de escucha 179. En una sesión de intercambio de BGP-4, los pares pueden enviar cuatro tipos de mensajes:

- **OPEN** y **NOTIFICATION**: se utilizan para abrir una sesión o para cerrarla debido a algún error, respectivamente.
- **KEEPALIVE**: se utilizan entre pares para asegurarse de que la conexión sigue existiendo durante los períodos de inactividad.
- **UPDATE**: mensajes de actualización para llevar información de accesibilidad de la red. Una actualización, o bien anuncia un prefijo, o retira un prefijo anunciado anteriormente.

A su vez, el establecimiento de una sesión BGP-4 atraviesa los siguientes estados como se observa en la Figura 2.7:

- **Idle**: el encaminador está buscando en la tabla de encaminamiento la ruta para alcanzar el vecino.
- **Connect**: el encaminador ha encontrado una ruta hacia el vecino y ha completado el inicio de sesión TCP.
- **Open sent**: se ha enviado el mensaje de inicio con los parámetros para establecer la sesión BGP-4.
- **Open confirm**: el encaminador ha recibido la confirmación de los parámetros para el inicio de sesión.




(a) Mensajes sesión BGP-4 (b) Sesión BGP-4

Figura 2.6: BGP-4

- **Active**: en el caso de que no se obtiene respuesta para el mensaje `open`.
- **Established**: la sesión se ha establecido y comienza el encaminamiento.

```
3d21h: BGP: 10.0.0.2 went from Idle to Active
3d21h: BGP: 10.0.0.2 open active, delay 21531ms
3d21h: BGP: 10.0.0.2 open active, local address 10.0.0.1
3d21h: BGP: 10.0.0.2 open failed: Connection refused by remote host
3d21h: BGP: 10.0.0.2 passive open
3d21h: BGP: 10.0.0.2 went from Active to Idle
3d21h: BGP: 10.0.0.2 went from Idle to Connect
3d21h: BGP: 10.0.0.2 rcv message type 1, length (excl. header) 26
3d21h: BGP: 10.0.0.2 rcv OPEN, version 4
3d21h: BGP: 10.0.0.2 went from Connect to OpenSent
3d21h: BGP: 10.0.0.2 sending OPEN, version 4, my as: 100
3d21h: BGP: 10.0.0.2 rcv OPEN w/ OPTION parameter len: 16
3d21h: BGP: 10.0.0.2 rcvd OPEN w/ optional parameter type 2 (Capability) len 6
3d21h: BGP: 10.0.0.2 OPEN has CAPABILITY code: 1, length 4
3d21h: BGP: 10.0.0.2 OPEN has MP_EXT CAP for afi/safi: 1/1
3d21h: BGP: 10.0.0.2 rcvd OPEN w/ optional parameter type 2 (Capability) len 2
3d21h: BGP: 10.0.0.2 OPEN has CAPABILITY code: 128, length 0
3d21h: BGP: 10.0.0.2 OPEN has ROUTE-REFRESH capability(old) for all address-families
3d21h: BGP: 10.0.0.2 rcvd OPEN w/ optional parameter type 2 (Capability) len 2
3d21h: BGP: 10.0.0.2 OPEN has CAPABILITY code: 2, length 0
3d21h: BGP: 10.0.0.2 OPEN has ROUTE-REFRESH capability(new) for all address-families
3d21h: BGP: 10.0.0.2 went from OpenSent to OpenConfirm
3d21h: BGP: 10.0.0.2 send message type 1, length (incl. header) 45
3d21h: BGP: 10.0.0.2 went from OpenConfirm to Established
3d21h: %BGP-5-ADJCHANGE: neighbor 10.0.0.2 Up
```

Figura 2.7: Estados sesión BGP-4

Existen dos tipos de protocolos BGP-4: el que se usa entre vecinos ubicados en diferentes sistemas autónomos, Exterior BGP-4 (eBGP), y el que se usa dentro del mismo sistema autónomo, Interior BGP-4 (iBGP). Para que dentro de un AS todos los encaminadores BGP-4 conozcan todas las redes propagadas vía iBGP, es necesario que el AS se encuentre totalmente mallado (*fully meshed*), es decir, que se establezcan sesiones iBGP-4 entre todos los encaminadores BGP-4, como se observa en la Figura 2.8. En consecuencia, para n encaminadores se necesitarán $\frac{n(n-1)}{2}$ sesiones iBGP-4, lo cual puede resultar inviable en un AS con un número considerable de encaminadores. Por ello, para hacer escalable el protocolo iBGP y reducir el número de sesiones iBGP dentro de un sistema autónomo se utilizan dos técnicas: Confederaciones BGP-4 y Reflectores de Rutas.

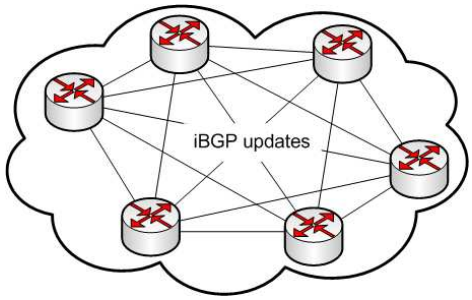



Figura 2.8: AS completamente mallado (iBGP-4)

### 2.4.1. Confederaciones BGP-4

Dentro de un AS es posible definir ASs internos denominados **confederaciones** [89], los cuales se comportan como un solo AS general de cara al exterior. La finalidad de una confederación BGP-4 es reducir el mallado iBGP dentro de un AS. De este modo, en el interior de cada AS interno se utilizará un mallado total entre sus encaminadores de borde y cada AS se conectará con el resto de ASs dentro de la confederación.

Aunque los encaminadores de borde de ASs diferentes dentro de la confederación intercambian información de encaminamiento mediante sesiones eBGP, dicha información se intercambia como si se tratase de sesiones iBGP, es decir, que se preservan atributos como `NEXT_HOP`, `METRIC` y `LOCAL_PREF`. Además, desde el punto de vista de los ASs externos a la confederación, el grupo de ASs de la confederación se ven como un solo AS.

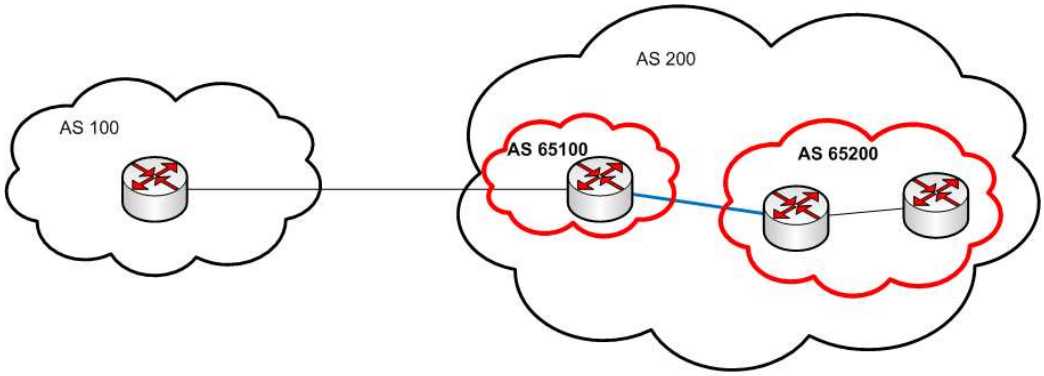



Figura 2.9: Ejemplo de confederación BGP-4

En el caso de la Figura 2.9, se supone que se dispone del AS 200, al cual pertenecen tres encaminadores de borde con sesiones eBGP con otros encaminadores de borde de otros ASs. Si no se utilizase una confederación en la que se divida el AS 200 en otros ASs, cada encaminador tendría tres sesiones establecidas (dos sesiones iBGP con el resto de encaminadores de borde del AS 200 y una sesión eBGP con un vecino de otro AS).

La solución al problema del exceso de sesiones iBGP puede conseguirse definiendo el AS 200 como una confederación, dentro de la cual se tienen múltiples ASs (AS 65100 y AS 65200). El identificador de la confederación tendrá el valor 200, de modo que el resto de ASs externos verá al conjunto de ASs de la confederación como un AS con número 200.

### 2.4.2. Reflectores de rutas

Una regla a cumplir por los encaminadores BGP-4 para evitar bucles es que no deben anunciar una ruta a un vecino iBGP-4 si dicha ruta la aprendió de otro vecino mediante iBGP-4. Como excepción a la regla anterior, un encaminador con la propiedad de **Route Reflector (RR)** (Reflector de Rutas) [14] puede anunciar una ruta aprendida por iBGP-4 a otro vecino iBGP-4, lo cual permite reducir considerablemente el número de sesiones iBGP-4 en un AS.

La combinación de un RR y sus clientes se denomina *cluster*, Figura 2.10. El resto de vecinos iBGP-4 son considerados como no clientes, por lo que el RR no les anunciará mediante iBGP-4 las rutas aprendidas de los clientes

iBGP-4 (aunque sí podría hacerlo mediante IGP).

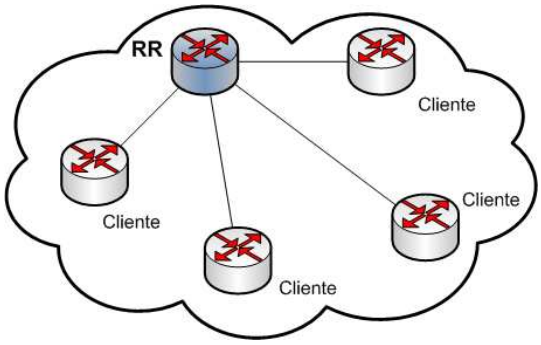



Figura 2.10: Reflector de rutas

Para solucionar el problema de no propagar rutas aprendidas de un vecino iBGP a otro vecino iBGP, los reflectores de rutas utilizan la siguiente información:

- `Originator-id`: Atributo opcional de cuatro bytes cuya función es guardar el identificador del encaminador que originó la ruta. De este modo, si debido a una inadecuada configuración una ruta es anunciada a su encaminador origen, dicha información será ignorada.

- `Cluster-list`: Atributo de una ruta en el que se van añadiendo el atributo `cluster-id`, entero identificador del clúster, al que pertenece cada RR por el que va pasando la ruta. Este atributo es útil para evitar bucles en el caso de múltiples RR en el interior de un mismo clúster, ya que un RR puede detectar si su `cluster-id` se encuentra ya en la lista y evitar así un bucle ignorando la ruta.

El despliegue de los RR se lleva a cabo dividiendo el *backbone* en varios *cluster*, de manera que cada grupo disponga de al menos un RR y múltiples clientes. En general, para aumentar la redundancia y evitar que todo el encaminamiento iBGP deje de funcionar en caso de fallo del RR, se suele configurar más de un RR dentro de un mismo *cluster*. Los RR establecen una malla completa iBGP entre ellos y pueden ser configurados de forma jerárquica, de forma que en un *cluster* se pueden tener RR clientes de otros RR de un nivel superior, Figura 2.11.

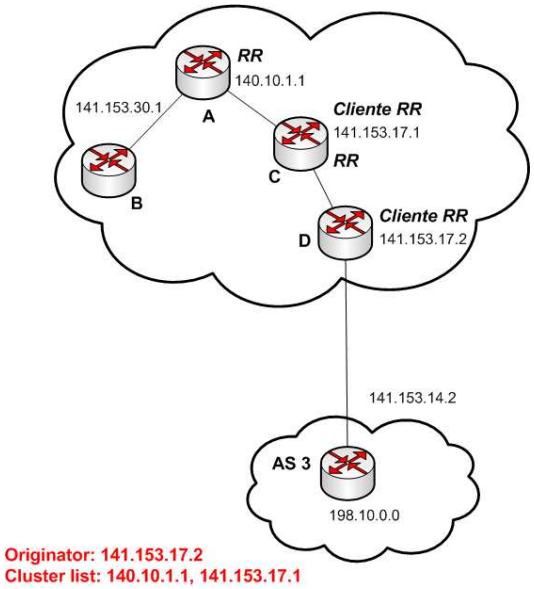



Figura 2.11: Información originator y cluster-id

El uso de RR reduce el número de vecinos en toda la red, reflejándose en menos procesamiento para los equipos encaminadores y brindando mejores tiempos de convergencia además de facilitar la administración de la red.

### 2.4.3. Atributos BGP-4

BGP-4 es un protocolo basado en políticas de encaminamiento, Policy Based Routing (PBR). Las rutas son seleccionadas no en función de métricas sino de dichas reglas o políticas. Para definir dichas políticas de encaminamiento se utilizan lo que se denominan **atributos BGP-4**. No todos los atributos tienen que estar presentes en todos los anuncios. Algunos de los más importantes se pueden ver en la Tabla 2.3 y se clasifican en las cuatro siguientes categorías:

- **Específicos obligatorios**. Deben ser reconocidos por todas las implementaciones de BGP y deben estar presentes en todas los mensajes de actualización: `ORIGIN`, `AS_PATH`, `NEXT_HOP`
- **Específicos discretos**. Deben ser reconocidos por todas las implementaciones de BGP pero, sin embargo, pueden no ser incluidos en los

| Valor | Código | Referencia |
|---|---|---|
| 1 | ORIGIN | [RFC1771] |
| 2 | AS_PATH | [RFC1771] |
| 3 | NEXT_HOP | [RFC1771] |
| 4 | MULTI-EXIT-DISC | [RFC1771] |
| 5 | LOCAL_PREF | [RFC1771] |
| 6 | ATOMIC-AGGREGATE | [RFC1771] |
| 7 | AGGREGATOR | [RFC1771] |
| 8 | COMMUNITY | [RFC1997] |
| 9 | ORIGINATOR-ID | [RFC2796] |
| 10 | CLUSTER-LIST | [RFC2796] |
| 11 | DPA | [Chen] |
| 12 | ADVERTISER | [RFC1863] |
| 13 | RCI D-PATH / CLUSTER-ID | [RFC 1863] |
| 14 | MP-REACH-NLRI | [RFC2283] |
| 15 | MP-UNREACH-NLRI | [RFC2283] |
| 16 | EXTENDED COMMNUNITIES | [Rosen] |
| ... | | |
| 255 | Reservado para desarrollo | |

Tabla 2.3: Atributos BGP-4

mensajes de actualización: LOCAL_PREF

- **Opcionales transitivos**. Una implementación de BGP puede no soportarlo como atributo, pero debe enviarlo a todos los encaminadores vecinos: AGGREGATOR

- **Opcionales no transitivos**. Una implementación de BGP puede no soportalo como atributo y no enviarlo a los encaminadores vecinos: MED

Estos atributos se utilizan para seleccionar la mejor ruta durante el **proceso de selección de rutas**. Cuando una ruta llega a un encaminador es filtrada por un filtro de entrada, el cual puede o no, bien rechazar la ruta, u opcionalmente modificar los atributos y pasarla al proceso de decisión. La ruta es evaluada por el proceso de encaminamiento y finalmente instalada en la tabla de rutas. Si es así, dicha ruta pasa a través de un filtro de salida

y por último, se anuncia a algunos o todos los vecinos. Este proceso está representado en la Figura 2.12 [9].

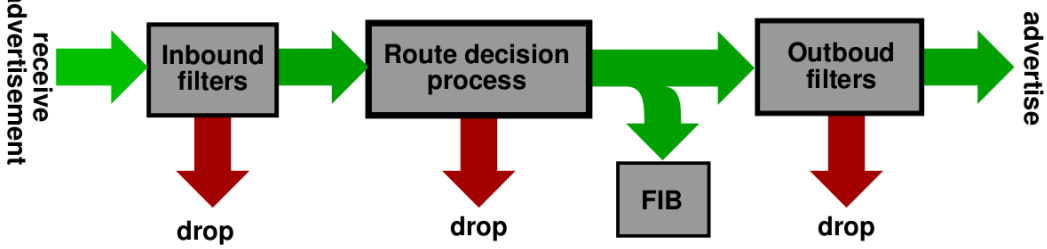



Figura 2.12: Proceso de selección de rutas

Durante el proceso de de decisión, las rutas se evalúan de acuerdo a los siguientes **criterios** [20] [52]:

1. Preferencia de la ruta con mayor `WEIGHT`
2. Preferencia de la ruta con mayor `LOCAL-PREFERENCE`
3. Preferencia de la ruta originada localmente
4. Preferencia de la ruta con `PATH` más corto
5. Preferencia de la ruta con valor `ORIGIN` más bajo
6. Preferencia de la ruta con valor `MED` más bajo
7. Preferencia de ruta eBGP-4 sobre iBGP-4
8. Preferencia de la ruta con valor IGP más cercano
9. Determinar si varias rutas requieren instalación en la tabla de rutas para BGP-4 Multipath
10. Preferencia de la ruta que se recibió primero, es decir, la más antigua.
11. Preferencia de la ruta procedente del encaminador con menor `ROUTER-ID`
12. Si el encaminador es el mismo para varias rutas, preferencia de la ruta con más corto `CLUSTER-LIST`
13. Preferencia de la ruta con menor dirección IP

**AS_PATH** Atributo específico y obligatorio que representa el camino de ASs que se deben atravesar para llegar a la red de destino, ver Figura 2.13. Este atributo es utilizado en la detección de bucles, de modo que un router de un determinado AS no aceptará una ruta si en el atributo `AS_PATH` está contenido su número de AS.

Cuando el atributo tiene más de 255 números de AS se expresa en múltiples segmentos `AS_SEQUENCE`. Esta composición inusual no es manejada correctamente por cualquier versión de Cisco hasta, al menos, la 12.2SRC y 12.4T. Cisco puede limitar la longitud máxima del atributo `AS_PATH` con el comando de configuración `bgp maxas-limit length`. Sin este comando, Cisco acepta todos los prefijos entrantes, pero marca los caminos en los cuales la longitud del atributo `AS_PATH` es superior a 254 números como no válido. Estas rutas se registran en la tabla de rutas BGP-4 pero no son utilizadas.

T. Li, R. Fernando y J. Abley proponen en [61] la creación de un nuevo atributo, `AS_PATHLIMIT`, que llevará una cota superior del número de ASs que puede contener el `AS_PATH`.

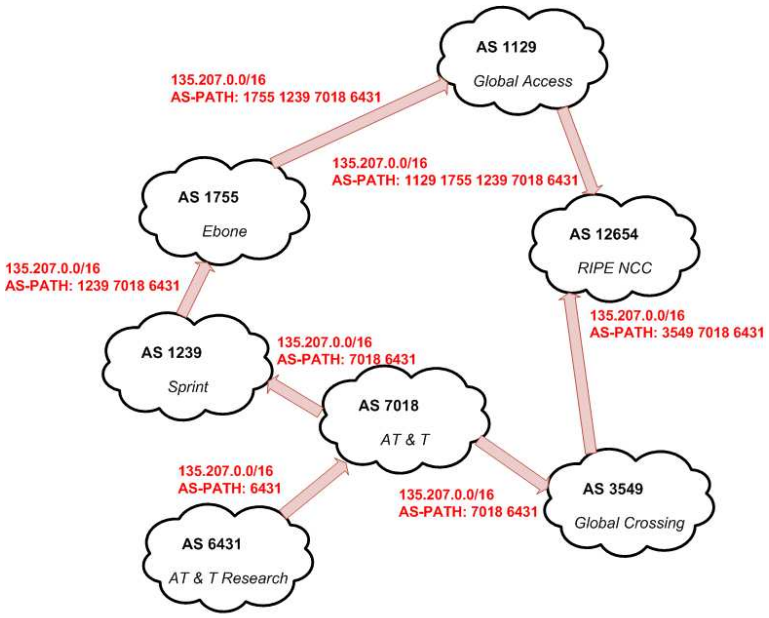



Figura 2.13: Atributo AS_PATH

**NEXT-HOP** Atributo específico y obligatorio que indica el siguiente salto. Cada vez que un anuncio cruza un límite de un AS, el atributo `NEXT-HOP` se cambia a la dirección IP del encaminador frontera que anunció la ruta, ver Figura 2.14.

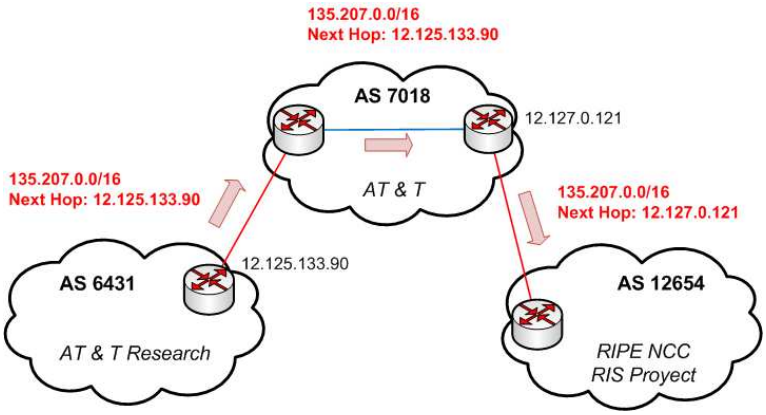



Figura 2.14: Atributo NEXT-HOP

**MULTI-EXIT-DISC** Atributo opcional y no transitivo que informa sobre la preferencia del punto de entrada al AS cuando existen varios enlaces externos, Figura 2.15. Este atributo se puede enviar a otros vecinos iBGP, es decir, dentro del mismo AS, sin embargo, nunca se propagará a otros ASs. Es frecuente utilizarlo cuando existen múltiples enlaces de salida con diferente ancho de banda, de forma que el enlace con mayor ancho de banda es preferido con un valor más bajo de `MED`. Algunos proveedores pueden elegir no tener en cuenta este atributo. Preferencia sobre el valor más bajo.

Mientras que este atributo funciona correctamente en muchos escenarios, pueden surgir algunos problemas cuando se utiliza en topologías dinámicas o complejas. Está relacionado con el problema de encaminamiento de la patata fría y la patata caliente descrita en [87].

Como se observa en la Figura 2.16, en la patata caliente [88] el encaminador reenvía el paquete hacia el camino con la menor demora, por tanto, esto tiende a limitar los recursos de ancho de banda consumidos por el tráfico de enviar paquetes al siguiente AS. En el lado opuesto está la patata fría, en la cual el encaminador transporta los paquetes en la medida de lo posible, en su propia red antes de entregarlos a otro AS. La política anterior minimiza la

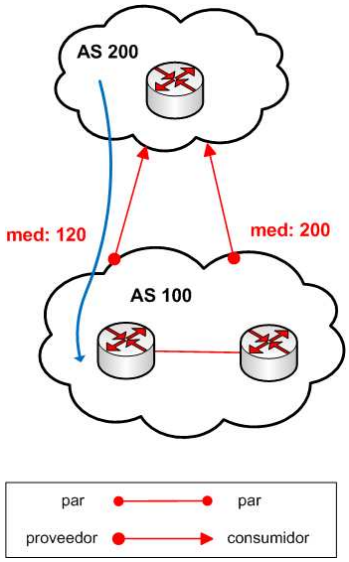


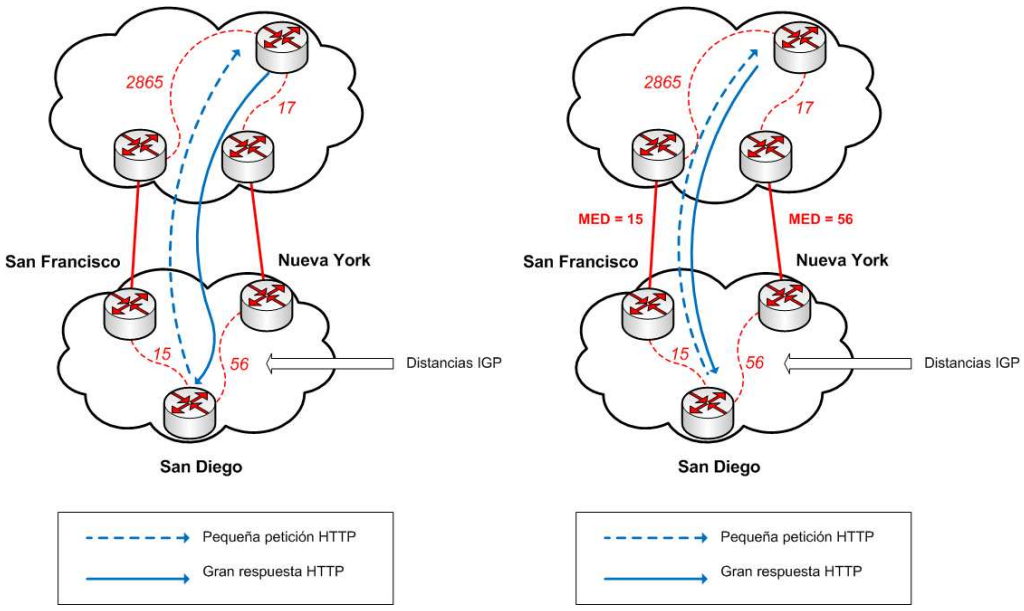

Figura 2.15: Atributo MULTI-EXIT-DISC

carga en la red del ISP mientras que la primera le da al proveedor de Internet un mayor control sobre la calidad extremo a extremo. Como se observa en la Figura 2.16b, el atributo `MED` se considera antes que la distancia IGP.

**LOCAL_PREFERENCE** Atributo específico y discreto que informa sobre la preferencia del punto de salida del AS, ver Figura 2.17. Sólo afecta al encaminamiento iBGP. Con este atributo se pueden implementar enlaces de backup dando preferencia a unas rutas sobre otras. Es útil para seleccionar el tráfico de salida cuando un AS tiene conectividad con varios ASs o múltiples rutas BGP, incluso con el mismo `NEXT_HOP`. Preferencia sobre el valor más alto.

**COMMUNITY** Atributo opcional y transitivo que identifica a un grupo de destinos que comparten una propiedad común, ver Figura 2.18. El administrador de cada AS puede definir a qué comunidades pertenece un destino. Por defecto, todos los destinos pertenecen a la comunidad de Internet en general (comunidad de valor 0). Cada destino puede ser miembro de varias comunidades.

El atributo está formado por cuatro octetos, codificando el número de AS en los dos primeros octetos, es decir, con el formato <AS:Valor>. Por tanto,




(a) Patata caliente (b) Patata fría

Figura 2.16: Problema de la patata caliente y fría

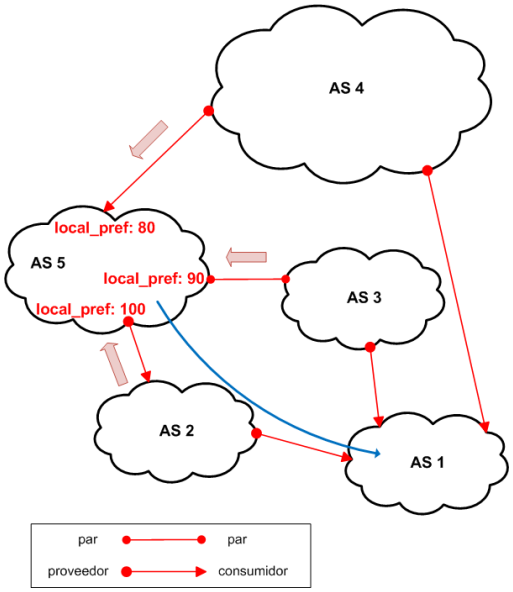



Figura 2.17: Atributo LOCAL_PREFERENCE

las comunidades son tratadas como 32 bit, sin embargo por razones administrativas de asignación [1], los valores que van desde 0x0000000 a 0x0000FFFF y desde 0xFFFF0000 a 0xFFFFFFFF quedan reservados.

Las comunidades más conocidas son:

- `NO-EXPORT (0xFFFFFF01)`: No se anuncian fuera del AS o confederación BGP.
- `NO-ADVERTISE (0xFFFFFF02)`: No se anuncian a ningún otro par BGP.
- `LOCAL-AS` o `NO-EXPORT-SUBCONFED (0xFFFFFF03)`: No se anuncian a otros pares eBGP.

## 2.5. Ingeniería de tráfico

La Ingeniería de Tráfico o Traffic Engineering (TE) [50] es el arte de conseguir encaminar el tráfico por la red de la forma que se quiera. Los

[1] http://www.iana.org/assignments/bgp-well-known-communities

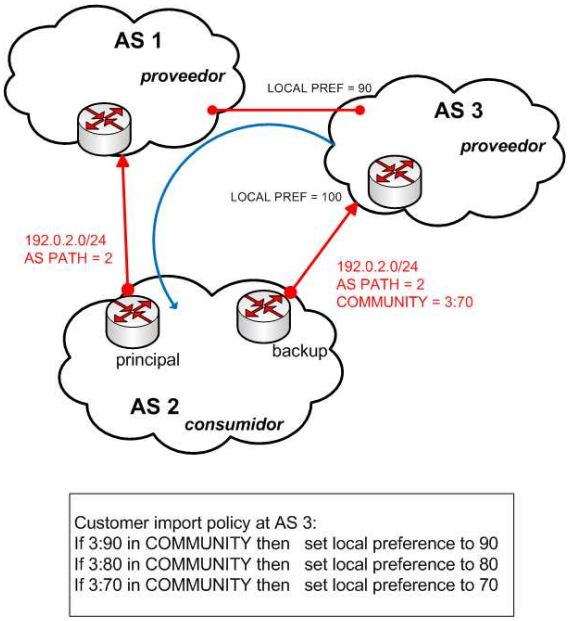



Figura 2.18: Atributo COMMUNITY

objetivos de la TE se pueden resumir en evitar la congestión, distribuir el tráfico dentro de la red con el fin de aumentar la cantidad de tráfico que puede ser transportado por la red, reacción rápida ante fallos alejando el tráfico de los enlaces caídos, proporcionar capacidad de recuperación y apoyo eficiente a los requerimientos de la Quality of Service (QoS) (Calidad de Servicio).

El desarrollo de técnicas eficaces para adaptarse a las rutas de tráfico reinantes y a la topología ha sido un área activa de investigación en los últimos años [10] [11] [12] [30]. El trabajo previo de la TE se ha centrado principalmente en los protocolos IGP, tales como OSPF, ISIS, y Multiprotocol Label Switching (MPLS), que controlan el flujo de tráfico dentro de un único AS. La TE intradominio es un problema entendido y con soluciones [32] [33].

Por el contrario, el estado del arte de la TE entre dominios es extremadamente primitivo. El Traffic Engineering Working Group del Internet Engineering Task Force (IETF), que se ha centrado casi exclusivamente en la TE entre dominios, señaló recientemente: *"se aplica generalmente en forma de ensayo y error. Aún no se ha concebido un enfoque sistemático para la ingeniería de tráfico entre-dominio"* [10]. De todos modos, actualmente algu-

nos mecanismos primitivos de control de encaminamiento basados en BGP son utilizados por los ASs para la TE entre dominios. Estos mecanismos se apoyan en el ajuste de atributos de las rutas BGP.

Un paso importante hacia la comprensión de las necesidades de la TE es conseguir una caracterización del tráfico de Internet. Se pueden utilizar diferentes enfoques para encontrar una taxonomía del tráfico de Internet, por ejemplo, se podría utilizar valores técnicos como prefijos IP. Sin embargo, un método mucho mejor puede ser desde un punto de vista **económico**, más cercano a la relación de negocios de cada una de las partes, del consenso sobre un acuerdo de interconexión. En primer lugar está la relación del **proveedor-consumidor**, ver Figura 2.19. Esto a menudo afecta a un pequeño AS, que actúa como un consumidor en este modelo, el cual firma contratos con uno o varios proveedores para acceder a Internet. El proveedor está de acuerdo en encaminar el total del tráfico anunciado por el AS y es pagado por ese servicio. Generalmente, el cliente suele ser más pequeño que el proveedor. Los grandes proveedores tienen un gran poder de mercado.

Cuando se tienen dos dominios del mismo tamaño o del mismo poder de mercado se establece una relación de **igual a igual** (*peer to peer*) entre ellos, Figura 2.19. En este modelo, ambas partes están de acuerdo en encaminar el tráfico procedente del otro, compartiendo sus tarifas de tráfico. [13] Sin embargo, los detalles del contrato no están disponibles públicamente en la mayoría de los casos.

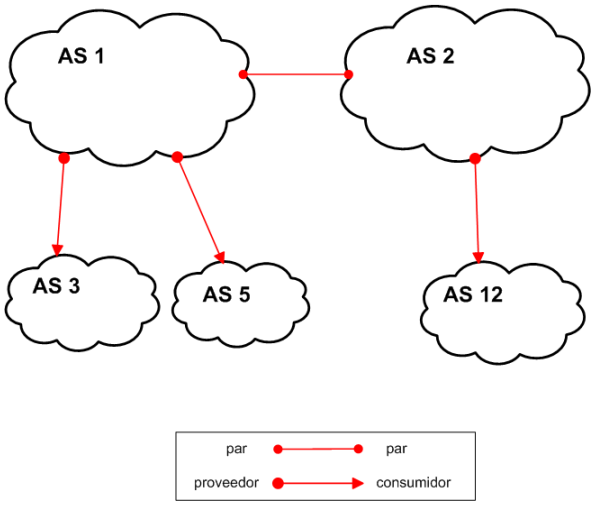



Figura 2.19: Relaciones de conexión

Un punto de intercambio de Internet (Internet Exchange Point) proporciona una infraestructura compartida a los ISP para intercambiar tráfico. El ámbito de aplicación es sobre todo geográfico. Los principales objetivos son la racionalización de los flujos de tráfico y disminuir el coste de las conexiones de larga distancia. Ejemplo de ellos son España Internet Exchange (EspaNIX), London Internet Exchange (LINX), Cataluña Internet Exchange (CatNIX).

### 2.5.1. Ingeniería de tráfico intradominio

Dentro de un único dominio, la topología de la red es conocida por todos los encaminadores debido a la utilización de protocolos de encaminamiento de estado de enlace, ver Sección 2.3.3. Además, el protocolo de encaminamiento intradominio suele optimizar un único objetivo global. Por lo tanto, varias técnicas pueden ser usadas para controlar el flujo de paquetes IP. En concreto, el protocolo IGP calculará el mejor camino para alcanzar cada destino.

### 2.5.2. Ingeniería de tráfico interdominio

Los requerimientos de la TE interdominio son diversos. Dependen de la conectividad con otros ASs pero también del tipo de negocio manejado por el AS. Típicamente, los proveedores que alojan una gran cantidad de servidores web y por lo general tienen varias relaciones consumidor-proveedor, tratarán de optimizar la forma que tiene el tráfico de salir de sus redes. Por el contrario, el acceso a los proveedores que sirven a las pequeñas y medianas empresas, de acceso telefónico por ejemplo, desearán optimizar cómo el tráfico entra en sus redes. Y, por último, un AS de tránsito tratará de equilibrar el tráfico entre los múltiples enlaces que tiene con sus vecinos.

Por tanto, el principal objetivo de la TE interdominio es controlar a través de qué enlace el tráfico entra o sale de una red. Lo que significa favorecer un enlace sobre otro para llegar a un destino determinado, o para recibir el tráfico procedente de una fuente determinada. Este tipo de ingeniería de tráfico entre dominios puede ser realizada ajustando la configuración de los encaminadores BGP-4 del AS. Desafortunadamente, BGP-4 no tiene ningún tipo de forma obvia para especificar los requisitos de TE de un AS. Es por eso que los operadores de red deben tener una mirada más cercana al proceso de decisión BGP-4, Sección 2.4.3, para ser capaces de ajustar mejor el protocolo a sus necesidades.

En las siguientes secciones, se distingue entre medios de TE entrante y saliente. Sólo la combinación de ambos lleva a unos resultados adecuados de TE.

### Control de tráfico saliente

Para controlar el tráfico que sale de la red, el AS debe ser capaz de elegir la ruta que se utilizará para alcanzar un destino concreto a través de sus vecinos. Como un AS controla el proceso de decisión en sus rutas, basta dar preferencia a la ruta de salida deseada. El atributo más utilizado con el propósito de controlar el tráfico saliente es `LOCAL_PREF`, estudiado en [91].

Una utilización común de este atributo es preferir rutas aprendidas de clientes sobre las aprendidas de proveedores [37]. Otras situaciones similares son preferir un enlace con mayor ancho de banda en un AS con dos enlaces hacia un mismo proveedor superior, o preferir un enlace hacia el proveedor más barato en el caso de un AS conectado a dos proveedores.

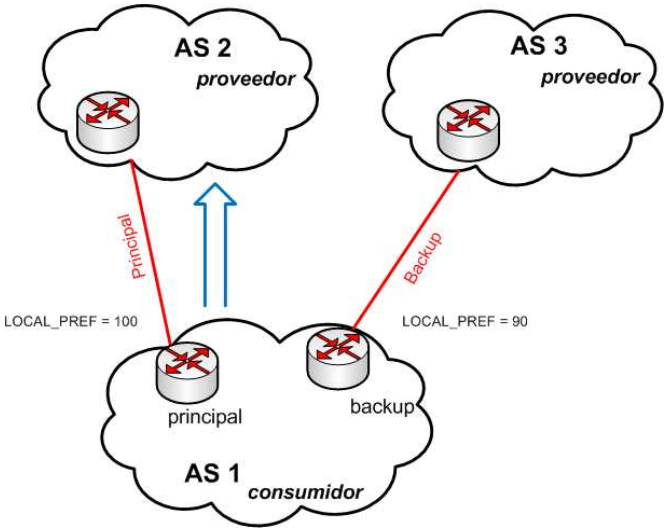



Figura 2.20: Control del tráfico saliente con `LOCAL_PREF`

Un ejemplo práctico puede ser un dominio AS 1 que tiene dos proveedores superiores AS 2 y AS 3 como se indica en la Figura 2.20. AS 1 quiere usar AS 2 como enlace primario y AS 3 sólo por motivos de backup. El encaminador de borde en el AS 1 tiene una conexión directa con los encaminadores de borde de AS 2 y AS 3, así que si AS 1 recibe una notificación de actualización de

AS 3 para un determinado prefijo IP, el filtro de entrada modifica el atributo `LOCAL_PREF` a un valor bajo. Si llega otro anuncio para el mismo prefijo IP desde el encaminador de borde de AS 2, el correspondiente encaminador de AS 1 establece una mayor preferencia local en las reglas de filtro de entrada. Por lo tanto el proceso de selección de rutas siempre ha de preferir el envío de paquetes a AS 2, según lo solicitado.

### Control del tráfico entrante

Controlar el flujo de tráfico de red entrante es mucho más difícil que el saliente. Para el tráfico saliente, se está en posesión directa de los paquetes en cuestión. Para el tráfico entrante, se tienen que prever las reglas de los filtros de entrada implementadas en el encaminador de borde vecino. En otras palabras, se ha de competir con las decisiones de TE saliente.

Una forma de definir las preferencias de un enlace entre dominios es dividir un anuncio de una ruta de un prefijo IP en prefijos más pequeños. Esto se basa en el hecho de que un encaminador siempre seleccionará la ruta más específica. Por ejemplo, AS 1 envía un anuncio para 192.168.0.0/23, prefiriendo recibir el tráfico a través de AS 2. Se podría dividir el prefijo IP en dos, 192.168.0.0/24 y 192.168.1.0/24, y anunciar los tres a AS 2. AS 3, por el contrario, sólo enviará el anuncio original. Los tres mensajes de actualización se propagarán a través de Internet, de modo que AS 3 aprenderá en algún momento la existencia de rutas más específicas, y utilizará éstas en su lugar. Las desventajas de esta solución es el crecimiento desmesurado de las tablas de encaminamiento en Internet, Figura 2.21 [38], dando lugar a una pérdida de prestaciones en todo el mundo. [17] informa que las rutas más específicas constituyen más de la mitad de las entradas en la tabla BGP. Ante este incremento, algunos grandes ISPs han comenzado a instalar filtros para eliminar anuncios de prefijos pequeños, a fin de evitar un crecimiento innecesario de las tablas de rutas BGP.

Otro método, basado en el atributo `AS_PATH`, es incrementar artificialmente su longitud [75]. Esta técnica llamada *AS Path prepending* explota el hecho de que el proceso de decisión de BGP-4 usa la longitud del `AS_PATH` para estimar la calidad de una ruta, considerando menos preferidas a las rutas de mayor longitud.

Para aumentar artificialmente la longitud de este atributo, un operador de red puede anteponer su propio número de AS más de una vez, como se observa en la Figura 2.22. Esto hace la ruta menos atractiva para ser

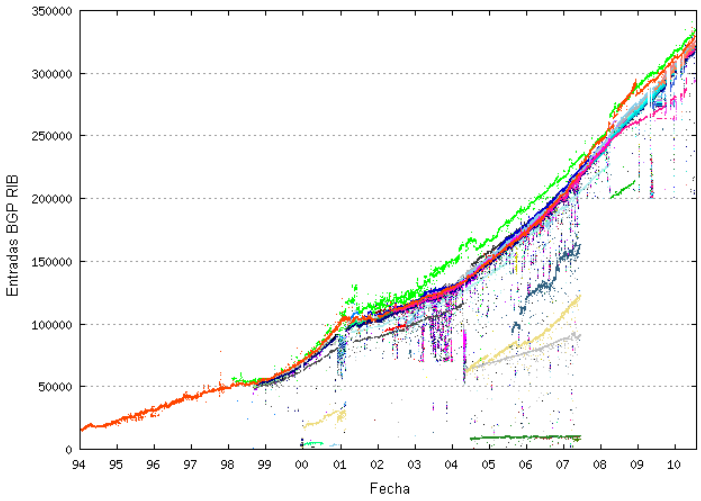



Figura 2.21: Crecimiento de las tablas de rutas BGP-4

considerada como enlace primario. Sin embargo, esta solución da lugar a múltiples problemas. Después de haber anunciado una ruta con menor valor a un vecino, no se tiene ninguna influencia en la forma en la que éste la manejará. Por una parte, la ruta puede ser insertada en las tablas de rutas de los vecinos, de modo que si el enlace primario falla, las rutas del enlace secundario puede ser utilizadas. Por otra parte, el vecino puede descartar la ruta por varias razones en su filtro de entrada del proceso de decisión, por ejemplo, que la longitud del `AS_PATH` sea demasiado larga para considerarla válida. El resultado del fallo del enlace primario en este caso, es una pérdida total de conectividad.

Con esta técnica no sólo se puede manipular la decisión de encaminamiento de vecinos directos, sino también de otros vecinos a varios saltos de distancia. Para los enlaces de *back-up*, el número de AS antepuestos es muy grande. Con una longitud correcta se puede conseguir un efecto de equilibrio de carga.

Existen datos relacionados con esta práctica de la red AT&T. Alrededor del 32 % de las rutas en las tablas BGP-4 analizadas habían incrementado la longitud del atributo `AS_PATH`. La Figura 2.23 [29] muestra la distribución de la cantidad de ASs antepuestos en los caminos. Como se puede observar, la mayoría de los caminos se extendieron en uno o dos saltos.

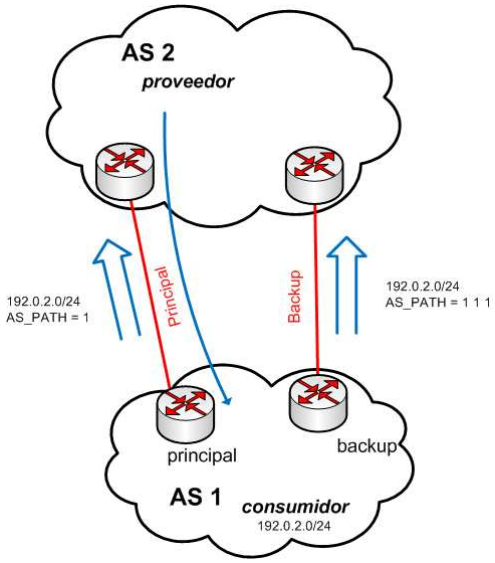



Figura 2.22: Control del tráfico entrante con `AS_PATH`

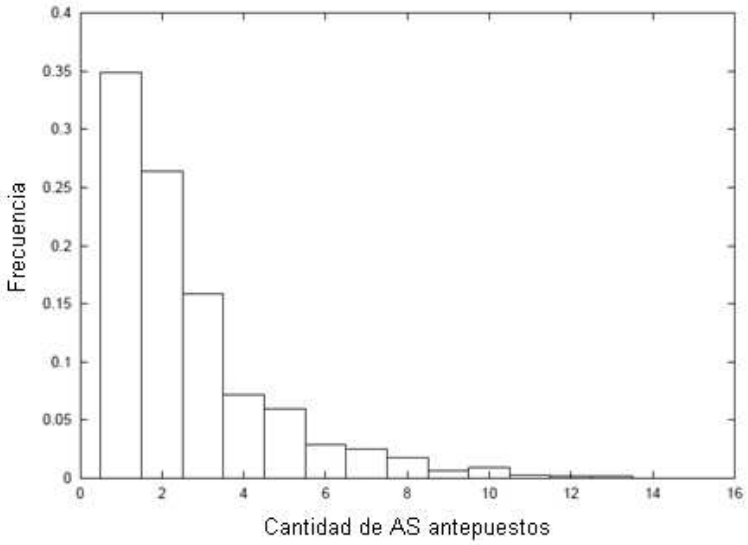



Figura 2.23: Incremento del `AS_PATH` del 32 % de las rutas de AT&T

El último método que permite a un AS controlar su tráfico de entrada es el atributo `Multi-Exit Discriminator (MED)`. Este atributo opcional

sólo puede ser utilizado por un AS multi-conectado a otro, para influir en el enlace que debe ser usado por el otro AS para enviar paquetes a un destino específico. No obstante, cabe señalar que la utilización del atributo `MED` está normalmente sujeta a la negociación entre ambos ASs y puede que alguno no acepte tener en cuenta el atributo `MED` en su proceso de decisión. Este atributo solo proporciona un control local del tráfico de entrada.

Hay demasiadas consideraciones que deben tenerse en cuenta. La mayoría de ellas en manos de operadores de red diferentes, que no se conocen entre sí y tienen objetivos diferentes sobre los efectos deseados, por lo que muchas de estas prácticas se llevan a cabo en forma de ensayo y error.

### 2.5.3. Conclusiones

Actualmente la TE en Internet, es reconocida generalmente como un medio válido para influir en los paquetes salientes y entrantes de un AS. Dado que las versiones originales de BGP-4 no permiten especificar directamente los parámetros de TE, estas prácticas son más o menos realizadas sobre la base de ensayo y error [8]. Nada beneficioso para los ingenieros, que no tienen una manera confiable de manipular su flujo de tráfico.

Hay incertidumbre acerca de la razón por la cual el flujo de tráfico se ve alterado en algunos casos. ¿Era un cierto flujo previsto, o era sólo un encaminador mal configurado? Debido a la falta de ese conocimiento, se inició un intento de uso más transparente de las comunidades [16].

Existen también herramientas [92] que automáticamente modifican los parámetros de la ingeniería de tráfico para un AS con el propósito de tener una configuración simple de complejas políticas de encaminamiento interdominio para el administrador de la red.

En la Figura 2.24 [74] se puede observar una comparativa de los diferentes métodos empleados en la TE para el control del tráfico, tanto entrante como saliente.

## 2.6. Inestabilidades

Los avances y mejoras en la infraestructura hardware y software de Internet han prevenido los problemas más graves de escasez de ancho de banda y falta de capacidad de los encaminadores. Sin embargo, a día de hoy, uno de

| | Traffic | Scope | Predictability | Scalability | Robustness | Efficiency |
|---|---|---|---|---|---|---|
| **BGP-based approaches** | | | | | | |
| **Local-Pref** | Out | Domain | ✓ | ✓ | ✓ | |
| **IGP weights** | Out | Domain | ✓ | (✓) | ✓ | |
| **Sel. announcements** | In | Internet | ✓ | | | Not robust to access link failure. |
| **More spec. prefixes** | In | Internet | | | ✓ | Sensitive to filtering |
| **MED** | In | Neighbor(s) | ✓ | ✓ | (✓) | Requires bilateral agreement(s) |
| **AS-Path prepending** | In | Internet | | ✓ | ✓ | Limited granularity (given the diameter of the Internet). Impact difficult to predict. |
| **Communities** | In | Internet | | ✓ | ✓ | Impact difficult to predict. Large search space. |
| **Non BGP-based approaches** | | | | | | |
| **RON, Detours** | In/Out | Internet | ✓ | | ✓ | Require modifications to end-systems. Rely on a large number of IP tunnels. |
| **NAT** | In | Internet | ✓ | | | Target multi-homed enterprise networks. Poses problem when one access link fails. |
| **New architectures** | In/Out | Internet | ✓ | ✓ | ✓ | Difficult to deploy in the current Internet. |

Figura 2.24: Técnicas de TE

los principales problemas es el que concierne a la **inestabilidad de encaminamiento**.

Definida informalmente como el cambio rápido de la información de accesibilidad y topología de la red, o lo que es lo mismo, falta de estabilidad, *"The routing system experiences transient changes in routes, caused by router and link failures or router misconfiguration."* [42]. Tiene numerosos orígenes, incluidos errores de configuración del encaminador [63], problemas físicos transitorios y de enlace de datos, y errores de software. Todas estas fuentes de inestabilidad de la red dan como resultado un gran número de actualizaciones que se transmiten a los encaminadores de Internet, además de forma colateral, se produce un aumento de pérdida de paquetes, retrasos en la convergencia de redes y consumo adicional de recursos dentro de la infraestructura de Internet.

La inestabilidad se puede propagar de encaminador a encaminador y di-

fundirse por la red. *In extremis*, puede llevar a la pérdida de la conectividad de gran parte de Internet, como el caso del operador pakistaní que desvió todo el tráfico de datos de YouTube [6].

No obstante, aunque las propiedades teóricas de los algoritmos de encaminamiento se han estudiado bien, el comportamiento real de los protocolos de encaminamiento no ha tenido un análisis formal. Estudios recientes han demostrado que el comportamiento de la implementación de los protocolos puede variar drásticamente de lo esperado [59].

BGP-4 es un protocolo basado en políticas de encaminamiento y la interacción de dichas políticas puede conducir a anomalías globales inesperadas, como a un encaminamiento no determinista y a un protocolo divergente [46] [93]. Estos efectos no deseados pueden ser clasificados generalmente en dos tipos de anomalías: **oscilación persistente** [68], en la que el protocolo nunca llega a establecer un conjunto estable de rutas; y **encaminamiento no determinista**, en el que el protocolo puede llegar a establecer un conjunto estable de rutas, pero la convergencia no está garantizada y el conjunto de rutas no es predecible. Como las políticas que causan estas anomalías se definen en redes administradas de forma autónoma, estos problemas pueden llegar a ser muy difíciles de depurar y corregir.

Se observa en la Figura 2.25 que ciertas configuraciones tienen más de una 'solución' o convergen en varias soluciones estables. En esta situación el proceso de convergencia de BGP puede operar de una manera no determinista, como en el caso de **BGP-Wedgies**.

Se han comenzado a desarrollar herramientas que enumeran el conjunto de todas las rutas estables [2].

### 2.6.1. BGP-Wedgies

*It has commonly been assumed that the Border Gateway Protocol (BGP) is a tool for distributing reachability information in a manner that creates forwarding paths in a deterministic manner. In this memo we will describe a class of BGP configurations for which there is more than one potential outcome, and where forwarding states other than the intended state are equally stable. Also, the stable state where BGP converges may be selected by BGP in a non-deterministic manner. These stable, but unintended, BGP states are termed here "BGP Wedgies".* [43]

[2] http://nms.lcs.mit.edu/rcc/

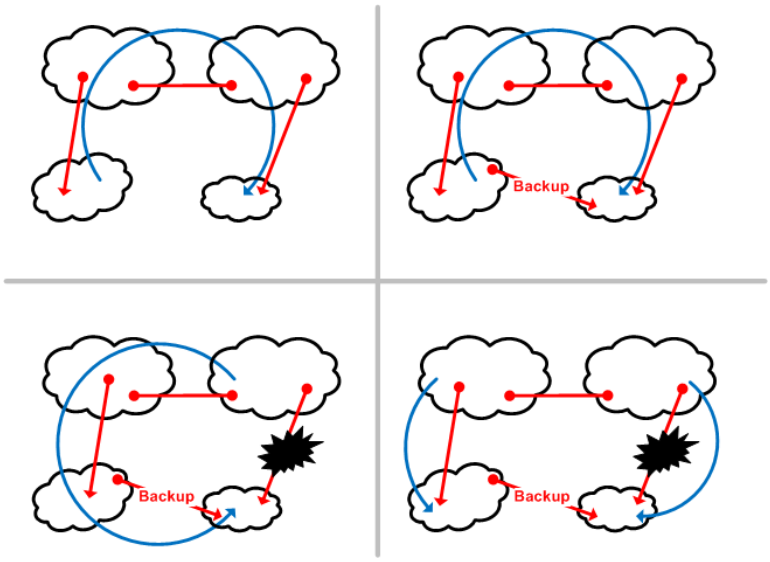
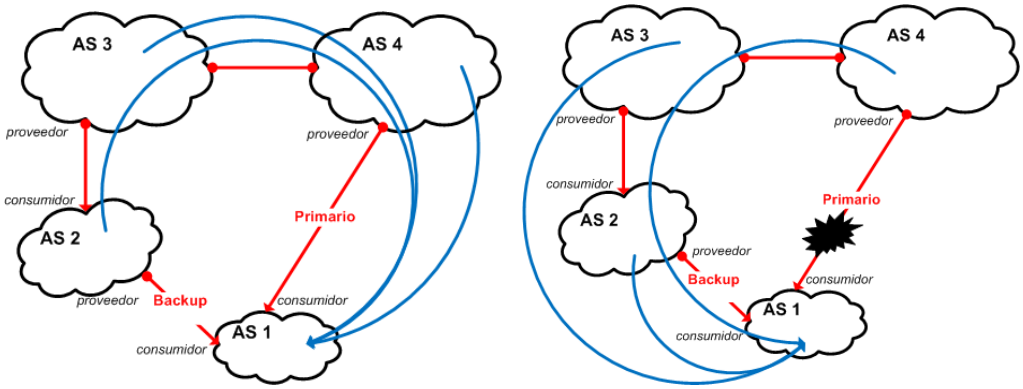



Figura 2.25: Ejemplo de impacto no predecible de BGP-4

La interación de políticas locales permite múltiples estados de encaminamiento estables. Algunas rutas son coherentes con las políticas propuestas, pero otras no. Si una ruta no deseada es instalada (*BGP is wedged*) entonces se necesita una intervención manual para cambiar al camino deseado. En concreto, es la falta de un mecanismo global explícito para expresar menos preferencia para los anuncios vía *backup*. Este comportamiento del protocolo se puede clasificar en dos tipos.

**3/4 Wedgies** Este tipo de wedgies es el más sencillo. Se puede solucionar el problema con esfuerzo y cooperación entre los diferentes proveedores. Un ejemplo de esta situación es indicado en la figura Figura 2.26a.

En este caso, AS1 ha marcado su anuncio de prefijos para AS2 como secundario, y su anuncio de prefijos para AS4 como primario. AS4 anunciará los prefijos AS1 a AS3. AS3 escuchará los anuncios de AS4 a través del enlace de interconexión, y seleccionará los prefijos de AS1 con la ruta 'AS4, AS1'. AS3 anunciará estos prefijos a AS2. AS2, recibirá dos rutas hacia AS1, la primera es a través de la conexión directa a AS1, y la segunda es a través de la ruta 'AS3, AS4, AS1'. AS2 preferirá el camino más largo a la ruta directa de backup, debido al atributo `LOCAL_PREF`. Bajo condiciones normales, el enlace primario conduce todo el tráfico y el enlace secundario está en modo




(a) Tráfico en circunstancias normales (b) Fallo enlace primario

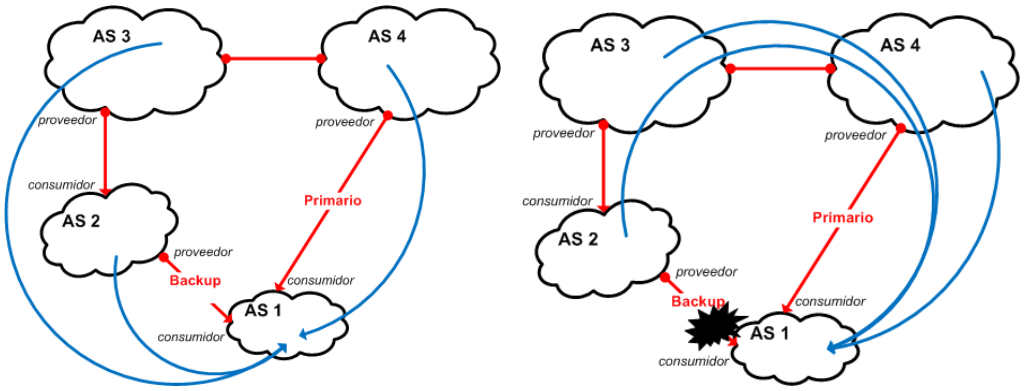
Figura 2.26: Ejemplo *3/4 Wedgie*

de espera.

Si el enlace primario se rompe, Figura 2.26b, causa un fallo de sesión BGP-4 entre AS1 y AS4. Entonces AS4 retira sus anuncios de AS1 a AS3, que, a su vez, enviará una retirada a AS2. En consecuencia, AS2 selecciona la ruta de backup hacia AS1. AS2 anunciará esta ruta a AS3, y AS3 anunciará este camino para AS4. De nuevo, este comportamiento es parte de la operación prevista de política de enlaces primarios y secundarios, y todo el tráfico de AS1 utiliza ahora la ruta de backup.

Cuando la conectividad entre AS4 y AS1 se restaura, el estado del BGP no volverá a su estado original, ver Figura 2.27a. AS4 aprende el camino principal a AS1 y volverá a anunciar esto a AS3 con la ruta 'AS4, AS1'. AS3, con una preferencia por defecto de las rutas anunciadas en el cliente a las rutas entre pares, sigue prefiriendo 'AS2, AS1' como ruta. AS3 no pasará ninguna actualización a AS2. Después de la restauración del circuito AS4 a AS1, el tráfico de AS3 para AS1 y de AS2 a AS1 se presentará a AS1 por la vía alternativa, aunque por el camino principal a través de AS4 está de nuevo en servicio. El comportamiento deseado sólo puede ser restaurado rompiendo a propósito la sesión BGP entre AS1 y AS2, aunque circule tráfico, Figura 2.27b.

**Full Wedgies** Un ejemplo de esta situación está representado en la figura Figura 2.28. En este ejemplo, el estado deseado es que AS2 y AS5 sean proveedores secundarios para AS1, y AS4 el proveedor primario.




(a) Recuperación enlace primario (b) Tráfico deseado

Figura 2.27: Ejemplo *3/4 Wedgie*

Cuando el enlace entre AS1 y AS4 se rompe y posteriormente se restablece, AS3 continua enviando el tráfico directamente a AS1 a través de AS2 o AS5, ver Figura 2.29. En este caso, un sólo reseteo del enlace entre AS2 y AS1 no restaurará el estado BGP original deseado, pues AS1 seleccionará como mejor ruta AS5, y AS2 y AS3 aprenderán el camino a AS1 a través de AS5.

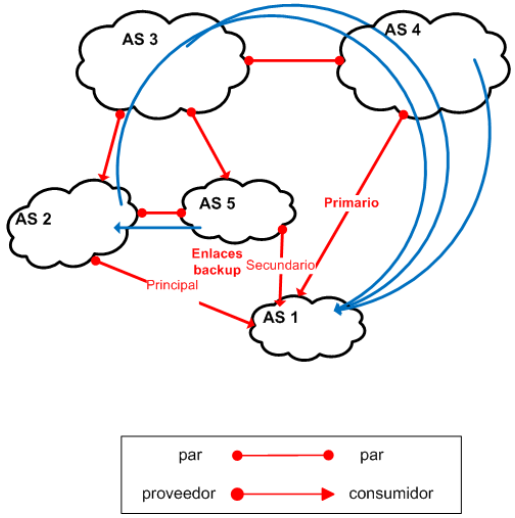
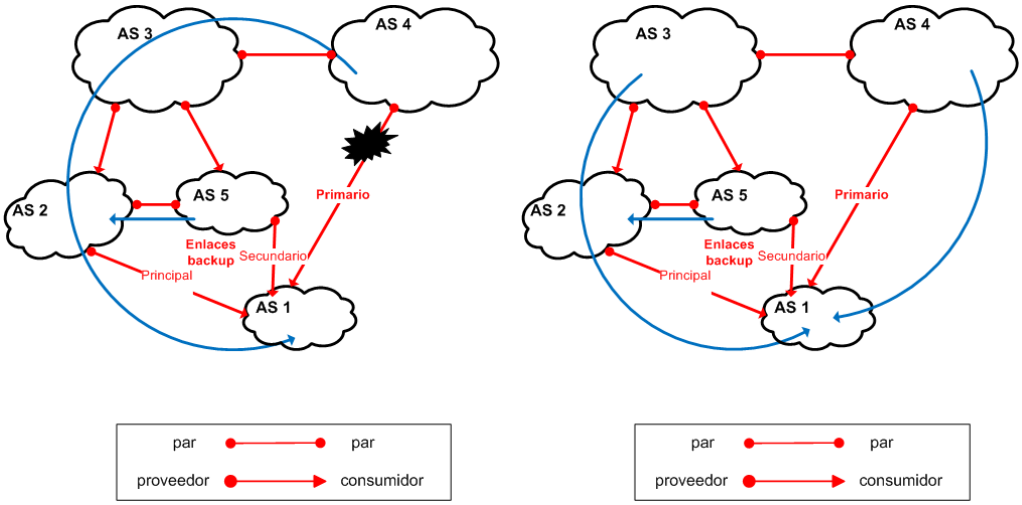



Figura 2.28: Ejemplo *Full Wedgie*




(a) Enlace entre AS1 y AS4 caido (b) Restauración enlace entre AS1 y AS4

Figura 2.29: Ejemplo *Full Wedgie*

Lo que AS1 está observando es tráfico entrante a través de su enlace secundario con AS2. Reseteando esta conexión no se consigue el tráfico de vuelta al enlace primario, pero en su lugar, el tráfico se encamina a través de AS5. La acción requerida para solucionar la situación es resetear simultáneamente los enlaces de AS1 con AS2 y AS5, Figura 2.30b, lo cual no es intuitivamente una solución obvia.

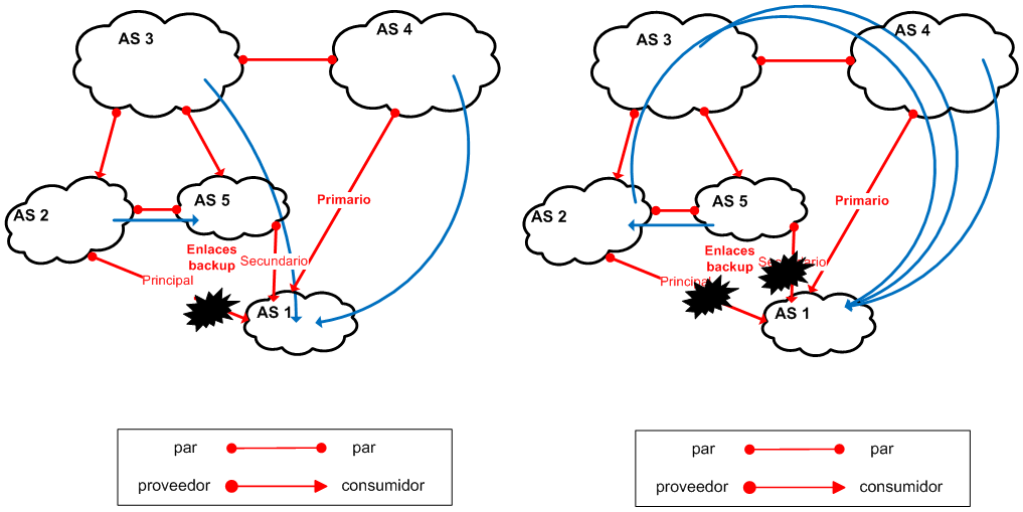
En la Figura 2.31 se puede observar lo realista de este tipo de configuraciones, cómo la distribución regional de los grandes ISP es propensa a interconexiones con Wedgies.

En definitiva, no hay garantías de que una configuración BGP-4 tenga una única solución de encaminamiento y tampoco de que tenga una solución. Las políticas complejas incrementan las posibilidades de que existan anomalías de encaminamiento.

### 2.6.2. Stable Path Problem

Timothy G. Griffin, F. Bruce Shepherd, y G. Wilfong, introducen en [46] *Stable Paths Problem (SPP)* como el problema de fondo teórico que BGP trata de resolver.

Se trata de una instancia para representar las interacciones de políticas




(a) Desactivar enlace backup (b) Recuperación del tráfico deseado

Figura 2.30: Ejemplo *Full Wedgie*

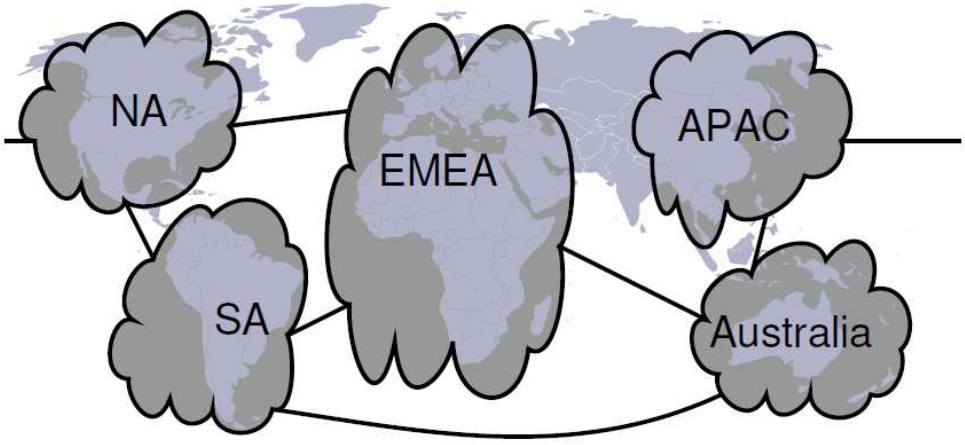



Figura 2.31: Lo realista de *BGP-Wedgies*

en una red. Informalmente, consta de un **grafo** no dirigido, en el que cada nodo representa un AS, siendo el nodo 0 el origen, y las aristas representan enlaces entre ASs, ver Figura 2.32. Cada nodo, excepto el origen, tiene un conjunto de caminos permitidos hacia el origen y además, a cada uno de los

cuales se les asigna un rango positivo que indica el nivel de preferencia del camino. Es decir, cada AS implementa una política en forma de orden lineal de preferencia de caminos a **un** destino.

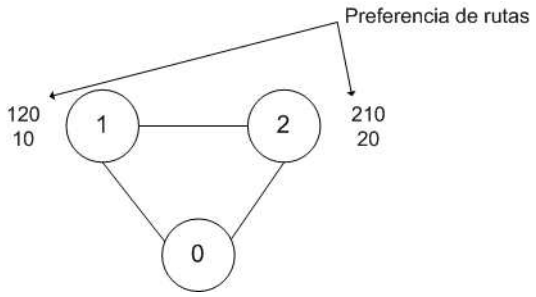
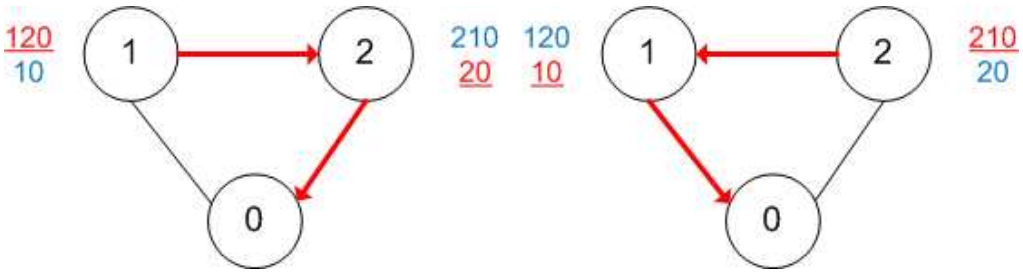



Figura 2.32: Representación gráfica *Stable Paths Problem*

Es un problema **NP-complejo**. *NP-hard o NP-complejo es el conjunto de los problemas de decisión que contiene los problemas H tales que todo problema L en NP puede ser transformado polinomialmente en H* [5].

Una solución es una asignación de rutas permitidas a los nodos, de forma que la ruta asignada a cada nodo es su ruta de mayor valor incluidas las rutas asignadas a sus vecinos. Una solución puede no ser única en la red. La configuración que se muestra en la figura Figura 2.33 originalmente en [46], llamada *DISAGREE* tiene dos posibles soluciones, representadas en la Figura 2.33a y Figura 2.33b.

Cualquiera de las dos rutas son estables porque ningún nodo puede aprender una con mayor preferencia. Desafortunadamente, el protocolo no tiene




(a) Primera solución (b) Segunda solución

Figura 2.33: DISAGREE

que converger a cualquiera de las dos. Por tanto, encontrar una solución no garantiza la convergencia.

La siguiente Figura 2.34 es una versión del SPP presentado en [93], llamado *BAD GADGET*. Si ambos nodos comienzan eligiendo la ruta directa 10 y 20 y las anuncian al resto de los encaminadores, el protocolo puede oscilar de forma indefinida.

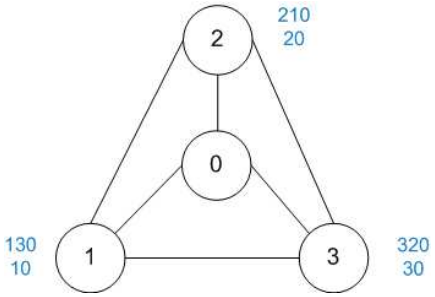



Figura 2.34: BAD GADGET

# Capítulo 3

# Software de emulación Netkit

La creación de un laboratorio de redes puede ser muy costosa, tanto en términos económicos como de espacio. Con el objetivo de observar y entender el comportamiento y configuración de BGP-Wedgies por un lado y multihoming por otro, se van a emular dichos conceptos en el entorno **Netkit**.

## 3.1. Introducción

Hasta hace relativamente poco, para experimentos prácticos se utilizaban laboratorios físicos juntando ordenadores de bajo coste y equipamiento de red. Esto suponía unos costes y espacio innecesarios, por suerte, la era de la **virtualización** acudió al rescate con lo que hoy en día es posible montar complejos laboratorios virtuales dentro de nuestro ordenador.

Figura 3.1: Laboratorio de redes

Actualmente, existe infinidad de software que permite realizar este tipo de experimentos de entornos de red, véase una comparativa en la Tabla 3.1 [67]. Este software se suele dividir en dos tipos: emuladores y simuladores. Los entornos de **simulación** permiten al usuario predecir el resultado de ejecutar un conjunto de dispositivos de red en una red compleja mediante el uso de un modelo interno especificado para el simulador. Los simuladores no reproducen necesariamente la misma secuencia de eventos que tienen lu-

gar en el sistema real, sino más bien aplican un conjunto interno de rutinas de transformación que conducen a la red a un estado final lo más cercano posible al que evolucionaría el sistema real. Sin embargo, los entornos de **emulación** tienen como objetivo reproducir las características y comportamiento de los dispositivos del mundo real. Por esta razón, consisten en una plataforma software o hardware que permite ejecutar las mismas piezas de software que se usarían en dispositivos reales. A diferencia de los sistemas de simulación, en un emulador, la red se pone a prueba bajo los mismos cambios de estado que se producirían en la realidad. En consecuencia, los emuladores, sobre todo si se implementan en software, son muy útiles para llevar a cabo experimentos que puedan poner en peligro el funcionamiento del sistema de destino o, simplemente, cuando el sistema real en sí no está disponible. Esto es verdad en particular para las redes de ordenadores, donde las configuraciones de dispositivos de red a menudo necesitan ser probadas antes de ser desplegadas.

La opción más sencilla son las máquinas virtuales individuales del tipo *VMWare* o *VirtualBox*, Figura 3.2, de modo que iniciando varias de estas máquinas virtuales se pueden hacer pruebas de red simulando una LAN. Sin embargo, llevar hasta ese punto tales herramientas puede exigir consumir demasiados recursos al ordenador. Por el contrario, Netkit está enfocado no tanto a la emulación de equipos concretos sino de redes completas, permitiendo definir una topología de red y hacer pruebas con ella.


Figura 3.2: Ejemplo VirtualBox

Desarrollado por la Universidad de Roma Tre [41], Netkit fue creado pa-

| | **Scale** | **Emulation type** | **Emulated device** | **License** |
|---|---|---|---|---|
| *Bochs* | Small | Full emulation | IA-32 x86 | GPL |
| *Cooperative Linux* | Medium | Kernel port | Linux box | Free |
| *CrossOver* | Medium | Compatibility layer | Windows APIs | Commercial |
| *DosBox* | Small | Full emulation | x86 DOS box | GPL |
| *DosEMU* | Small | Compatibility layer | DOS box | GPL |
| *Einar* | Large | Router emulation | Quagga based router | GPL |
| *Emulab* | Large | Testbed | — | Project based |
| *FAUmachine* | Medium | User-mode kernel | x86 box | GPL |
| *IMUNES* | Medium | Virtual image | Linux box | Free |
| *KVM* | Medium | Native virtualization | x86 box | GPL |
| *MLN* | Medium | Paravirtualization/User-mode kernel | Linux box | Free |
| *Modelnet* | Large | Testbed | Linux box | GPL/BSD |
| *NCTUns* | Medium | Simulation | Host/Router | Free |
| *Netkit* | Medium | User-mode kernel | Linux box | GPL |
| *Parallels* | Medium | Full virtualization | x86 box | Commercial |
| *PearPC* | Small | Full emulation | PowerPC box | GPL |
| *Planetlab* | Large | Overlay network | — | Membership |
| *Plex86* | Medium | User-mode kernel | Linux box | Free |
| *Q* | Small | Full virtualization | x86 box | Free |
| *QEMU* | Small | Full virtualization | x86 box | GPL |
| *SVISTA* | Small | Full virtualization | x86 box | Commercial |
| *UML* | Medium | User-mode kernel | Linux box | GPL |
| *UMLMON* | Medium | User-mode kernel | Linux box | GPL |
| *vBET* | Medium | User-mode kernel | Linux box | N/A |
| *VDE* | Large | Overlay network | — | GPL |
| *VINI* | Large | User-mode kernel | Linux box | Membership |
| *VirtualBox* | Small | Full virtualization | x86 box | GPL/Commercial |
| *Virtual PC* | Small | Full virtualization | x86 box | Free |
| *Virtuozzo* | Small | Full virtualization | x86 box | Commercial |
| *VMware* | Small | Full virtualization | x86 box | Commercial |
| *VNUML* | Medium | User-mode kernel | Linux box | GPL |
| *Win4Lin* | Medium | Full virtualization | x86 box | Commercial |
| *Wine* | Medium | Compatibility layer | Windows APIs | GLPL |
| *Xen* | Medium | Paravirtualization | x86 box | GPL/Commercial |

Tabla 3.1: Comparativa emuladores

ra disponer de un entorno virtual donde poder configurar y realizar pruebas experimentales de topologías de red, ver la Figura 3.3. La creación del entorno virtual es realizada por medio de un gran conjunto de *scripts* de Shell para GNU/Linux. Estos *scripts* permiten crear nodos virtuales de máquinas GNU/Linux. Aunque los equipos de conexión en red son virtuales, tienen muchas de las características de los reales incluyendo la interfaz de configuración.

Netkit explota el software de código abierto, en su mayoría bajo la licencia General Public License (GPL) [34].

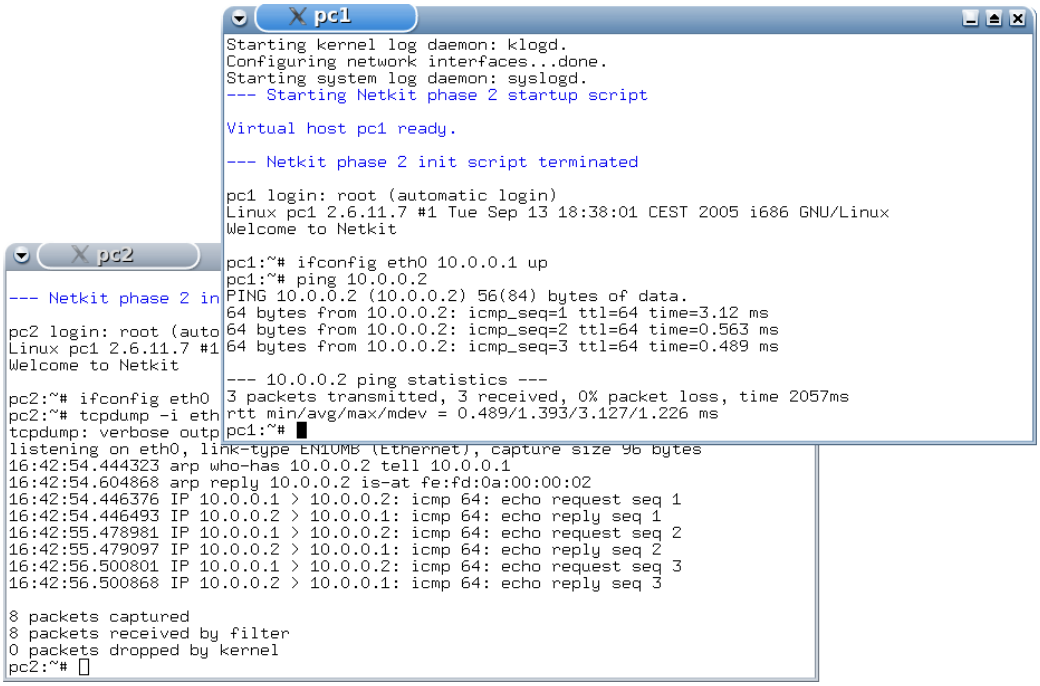



Figura 3.3: Comando *ping* entre dos máquinas virtuales con Netkit

## 3.2. Características

El enfoque de emulación adoptado en Netkit es simple. Básicamente, todo dispositivo que compone una red es implementado dentro de Netkit como una **máquina virtual**. Cada máquina virtual tiene un conjunto de recursos

virtuales que se asignan a las porciones de los recursos correspondientes en la máquina real. La Figura 3.4 muestra cómo se lleva a cabo esta asignación.

Las máquinas virtuales están equipadas con un disco, cuya imagen es un archivo en la máquina destino; dichas máquinas tienen su propia región de memoria, cuyo tamaño puede establecerse en el inicio, y pueden ser configuradas con un número arbitrario de interfaces de red virtuales que están conectadas al concentrador o *hub* virtual. El recuadro de trazo discontinuo en la Figura 3.4 rodea los recursos virtualizados, mientras que todo componente fuera de ese recuadro corresponde a un dispositivo o proceso en la máquina real. Es posible observar que el concentrador virtual está alojado en la máquina real, de hecho, es un proceso especial que replica los paquetes de todas las interfaces conectadas. Si se desea, el concentrador virtual puede ser conectado a una interfaz de red de la máquina real, de modo que una máquina virtual puede llegar a una red externa, como Internet.

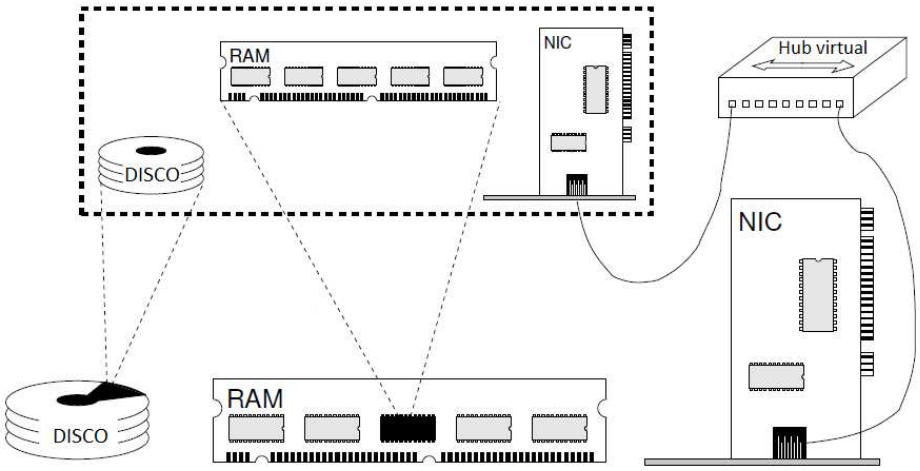



Figura 3.4: Recursos de una máquina virtual en Netkit

Las máquinas virtuales en Netkit se basan en User Mode Linux (UML), que se describe en la Sección 3.2.1. Arrancar una máquina virtual significa poner en marcha una instancia de UML, que a menudo requiere algunos argumentos complejos de línea de comandos. Por esta razón Netkit admite una configuración sencilla y gestión de máquinas virtuales a través de una intuitiva interfaz que consiste en varios *scripts*.

La Figura 3.5 describe la arquitectura de Netkit, compuesta por los bloques dentro del cuadro de líneas discontinuas. Cada bloque representa una pieza de software que corre por encima y es controlado por las herramientas de su izquierda. Las máquinas virtuales son instancias UML que directamente se ejecutan en el núcleo de la máquina y son manejadas por un conjunto de comandos, cuyos nombres comienzan con los prefijos l (*ltools*) y v (*vtools*). Además, las máquinas virtuales pueden ejecutar software de encaminamiento así como otras utilidades y herramientas.

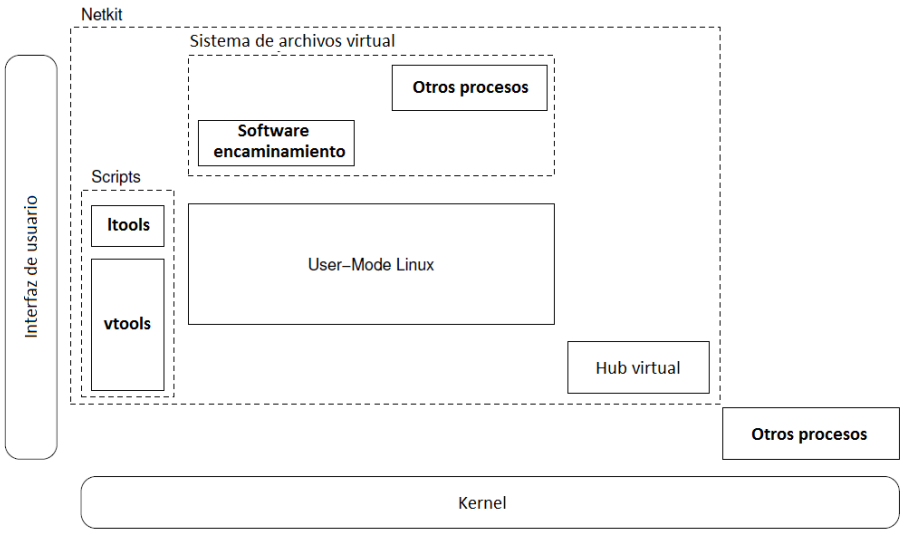



Figura 3.5: Arquitectura de Netkit

Los bloques con el texto en negrita representan los únicos componentes que se muestran al usuario final. Es posible observar que no hay necesidad de que el usuario final interactúe directamente con los núcleos de UML o concentradores virtuales. Por otra parte, tanto las herramientas *ltools* como *vtools* son accesibles para el usuario. La diferencia entre ambos es que las *ltools* proporcionan una interfaz de nivel superior a las máquinas virtuales y por lo tanto aprovechan *vtools* para implementar sus funcionalidades.

Las ventajas de Netkit respecto a otros emuladores de redes disponibles son entre otras: ser ligero, fácil de instalar y ejecutar, se ejecuta totalmente en espacio de usuario y no tiene dependencias con otras piezas de software. Además, viene con la mayoría de las herramientas de redes de uso general,

aunque en caso de necesidad, es posible también instalar paquetes adicionales dentro de las máquinas virtuales.

Por otro lado, Netkit sólo funciona sobre Linux, aunque existen versiones preliminares para el sistema operativo Windows, y proporciona máquinas virtuales Linux.

### 3.2.1. UML

En un entorno de emulación las máquinas virtuales tienen casi las mismas características de una máquina real, incluyendo su propio núcleo. Netkit explota UML como núcleo para las máquinas virtuales.

Inicialmente UML se creó para que los desarrolladores del núcleo de Linux pudieran probar versiones inestables del núcleo sobre un sistema en funcionamiento. Como el núcleo en prueba es sólo un proceso de usuario, si se cuelga, no compromete al sistema que lo aloja. Los fundamentos de UML se ilustran por su diseñador Jeff Dike en algunas publicaciones [26] [27].

UML es un puerto del núcleo de Linux que está diseñado para ejecutarse como un proceso en espacio de usuario. Los dispositivos de red son emulados en Netkit como máquinas virtuales UML que se ejecutan en una distribución completa GNU/Linux. UML permite arrancar una máquina Linux como un proceso de usuario corriendo dentro de una máquina anfitriona. Desde el punto de vista de la máquina anfitriona, UML es un proceso normal de usuario, véase Figura 3.6. Desde el punto de vista de un proceso que corre dentro de UML, UML es un núcleo, proporcionando memoria virtual y acceso a dispositivos, lo que denominamos máquina virtual.

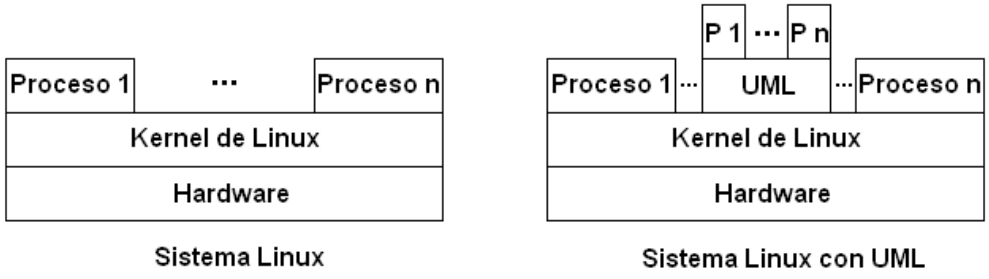



Figura 3.6: UML

Básicamente lo que hace UML es proveer virtualización para las llamadas al sistema. Normalmente, una llamada al sistema de un proceso de una

máquina virtual es gestionada directamente en el núcleo de la máquina. En su lugar, UML intercepta dichas llamadas al sistema. Por tanto, el núcleo UML no se comunica directamente con el hardware, lo hace a través del núcleo de Linux de la máquina anfitriona. Los procesos que corren dentro de UML funcionan igual que dentro de una máquina real Linux, ya que UML proporciona su propio espacio de direccionamiento del núcleo y de proceso y su sistema de gestión de memoria, véase figura Figura 3.7.

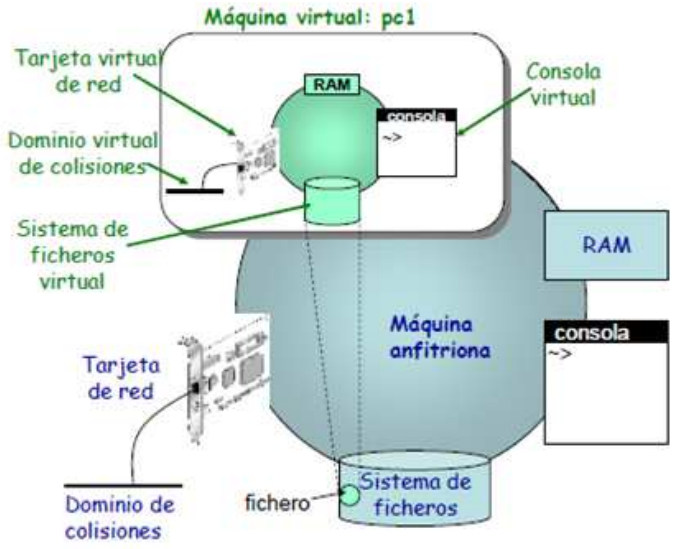



Figura 3.7: Máquina virtual y máquina anfitriona

Gracias a UML se puede ejecutar un conjunto de máquinas virtuales dentro de una máquina real, la máquina anfitriona. Las máquinas virtuales se conectan a través de dominios de colisión virtuales, véase Figura 3.8. Una máquina virtual puede funcionar como un equipo terminal, un encaminador o un conmutador, por ejemplo.

### 3.2.2. Interfaz de usuario

Aunque en este proyecto se utilizará la propia consola de los terminarles, la configuración de un entorno de red puede ser algo complicado para cualquier usuario. Muchas veces el terminal de Linux puede repeler a un usuario final

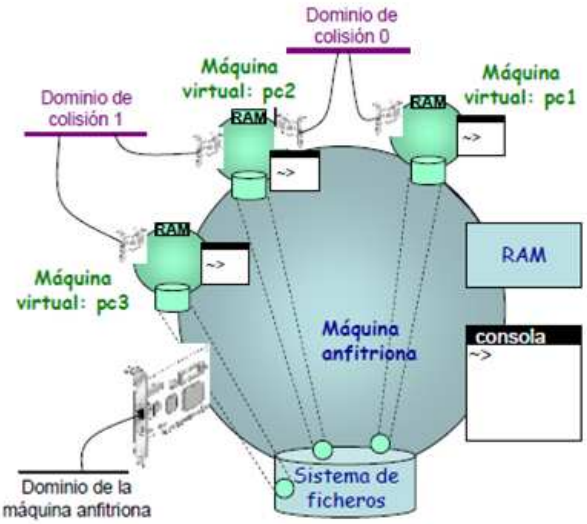



Figura 3.8: Conexión de máquinas virtuales a través de dominios de colisión

el usar una herramienta tan potente como puede ser Netkit. En el Apéndice B se pueden consultar los principales comandos utilizados en la creación de un laboratorio virtual.

Para facilitar el uso de esta herramienta se han desarrollado varios proyectos software que han proporcionado a Netkit un interfaz que permite al usuario centrarse en el uso únicamente de la aplicación y no de la instalación ni la configuración, algunos de los cuales se exponen a continuación.

**NetEdit** Desarrollado por la Universidad de Roma, Netedit permite el diseño de redes por medio de una aplicación gráfica, Figura 3.9. Esta aplicación, realizada en Java, permite al usuario realizar los diseños de redes en un entorno de desarrollo amigable. El usuario únicamente se encargaría de añadir los componentes (computadores, encaminadores...) y posteriormente realizar los enlaces oportunos. Una vez que el diseño estuviera completo, se ejecutaría. Cada nodo creado tendría su propio terminal para poder interactuar con él, igualmente como si se lanzará los nodos usando los comandos básicos de Netkit.

Como ventajas en vez del uso de comandos, se encontrarían disponer de

un interfaz gráfico para realizar los diseños y fácilmente redistribuible, con la posibilidad de guardar las configuraciones de red implementadas para usarlas en posteriores sesiones.

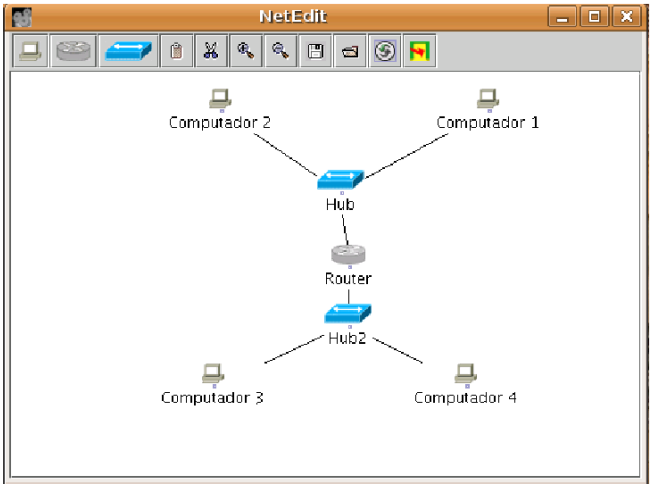



Figura 3.9: Interfaz de NetEdit

**VisualNetkit** VisualNetkit es un entorno gráfico que permite configurar y administrar un laboratorio de Netkit de una forma muy simple e intuitiva. Tiene una arquitectura de *plugin* que permite habilitar selectivamente funcionalidades o servicios en los elementos de red (máquinas virtuales, dominios de colisión, enlaces, interfaces...) de acuerdo a las necesidades del usuario. En la actualidad, VisualNetkit solo está disponible para abrir laboratorios que han sido configurados con la misma herramienta.

**NetGUI** Desarrollado por la Universidad Rey Juan Carlos, NetGUI ofrece un interfaz de usuario a Netkit, similar a las aplicaciones comentadas anteriormente. NetGUI, ha sido usada como método docente en asignaturas de la Titulación de Ingeniería de Telecomunicaciones de la Universidad Rey Juan Carlos, en concreto la asignatura de Arquitectura de Redes de Ordenadores.

### 3.2.3. Otras características

▷ Genéricas de red

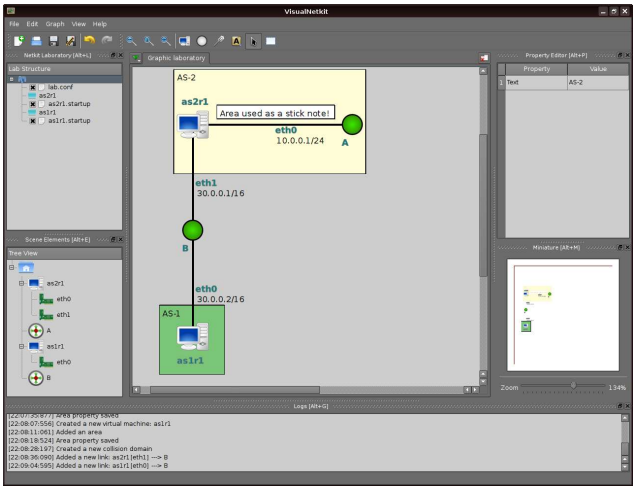



Figura 3.10: Interfaz de VisualNetkit

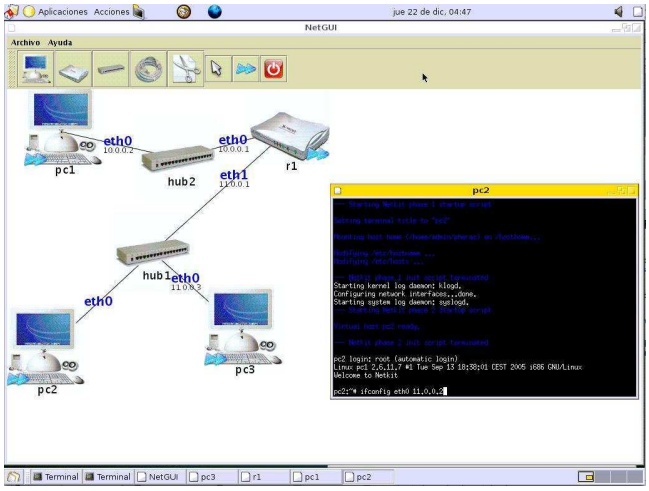



Figura 3.11: Interfaz de NetGUI

- Capa física
  - Capa física Ethernet (soporte a nivel del núcleo)
- Capa de enlace de datos
  - 802.1D Spanning Tree Protocol (STP) (brctl)
  - 802.1Q Etiquetado VLAN (vconfig)

- Point to Point Protocol (PPP)

◦ Conmutación de etiquetas

- Reenvío basado en MPLS
- Manipulación de pilas de etiquetas
- Distribución de etiquetas a través de Label Distribution Protocol (LDP)

◦ Capa de red

- Resolución de direcciones ARP y RARP (soporte a nivel de núcleo)
- Control de mensajes ICMP
- Encaminamiento IPv4 e IPv6 (soporte a nivel de núcleo)

◦ Capa de transporte

- TCP (soporte a nivel de núcleo)
- User Datagram Protocol (UDP) (soporte a nivel de núcleo)

◦ Capa de aplicación

- Configuración automática de Dynamic Host Configuration Protocol (DHCP)
- Domain Name System (DNS) tanto del lado del servidor como del lado del cliente
- Transferencia de correo electrónico vía Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP), Internet Message Access Protocol (IMAP)
- File Transfer Protocol (FTP) y Trivial file transfer Protocol (TFTP) tanto del lado del servidor como del lado del cliente
- Hypertext Transfer Protocol (HTTP) y Hypertext Transfer Protocol Secure (HTTPS)
- Network File System (NFS)
- Telnet
- Samba
- Secure SHell (SSH)
- Web proxying

▷ Encaminamiento

◦ Conmutación de etiquetas MPLS

- Protocolos de encaminamiento
  - BGP-4 (quagga/XORP)
  - OSPF (quagga/XORP)
  - RIP (quagga/XORP)
  - Balanceo de carga múltiple de igual costo
- Multidifusión
  - Multidifusión, Protocol Independent Multicast - Sparse Mode (PIM-SM)

▷ Herramientas de Seguridad

- Internet Protocol Security (IPSec)
- Internet Key Exchange (IKE)
- Sistemas de detección de intrusos
- Remote Authentication Dial-In User Server (RADIUS)

▷ Manipulación de paquetes

- Encapsulación
  - Túneles Generic Routing Encapsulation (GRE)
  - Túneles MPLS
- Captura de paquetes y disección
  - Tethereal
  - Tcpreen
  - Tcpdump
  - Ssldump
  - Ettercap
- Filtrado de paquetes
  - Filtrado de paquetes con el framework de netfilter (incluyendo Network Address Translation (NAT))
- Falsificado de paquetes
  - Dsniff
  - Hping

        - Sendip
        - Tcpreplay

- ▷ Varios
    - ○ Lenguajes de scripting
        - Awk
        - Bash
        - Expect
        - Python

## 3.3. Instalación

Ver Apéndice A.

# Capítulo 4

# Software de encaminamiento Quagga

## 4.1. Introducción

El encaminamiento puede ser manejado utilizando diferentes métodos como son, con equipos especializados como los encaminadores que tienen sus sistemas operativos propios, o con sistemas operativos estándar que tienen la funcionalidad de encaminamiento como es el caso de Windows y Linux. Sin embargo existen programas especializados para cumplir el papel de encaminamiento que pueden ser instalados bajo sistemas operativos estándar.

En el caso particular de Linux existen paquetes gratuitos y también no gratuitos, que están diseñados para el propósito de encaminar, muchos de ellos con una funcionalidad básica para situaciones de encaminamiento no muy complicadas, y otros con una programación mucho más extendida donde se pueden observar la implementación de los diferentes protocolos de encaminamiento. Entre los paquetes software de encaminamiento más usados cabe destacar soluciones de libre distribución como *Zebra*, *Quagga* o *Radvd*, así como aplicaciones propietarias tales como *ZebOs* y *Gated*.

La solución utilizada ha sido **Quagga**, una suite de software libre para poder usar la familia de sistemas operativos Unix, en particular FreeBSD, Linux, Solaris y NetBSD, como **encaminadores**. Proporciona servicios de encaminamiento TCP/IP con implementaciones de protocolos como OSPFv2[69], OSPFv3[22], RIPv1[49], RIPv2[64], RIPng[65] y BGP-4[66]. Además, da soporte a Reflectores de Rutas BGP.

Quagga actúa como conmutador del *GNU Zebra*, el cual a su vez es un demonio que se encarga de manejar las tablas de rutas del núcleo. Además de los tradicionales protocolos de encaminamiento IPv4, Quagga también soporta los protocolos de encaminamiento IPv6. Se distribuye bajo la bajo la licencia GPL [34].

## 4.2. Arquitectura

El software de encaminamiento tradicional está hecho como un programa o proceso que proporciona todas las funcionalidades de protocolo de encaminamiento. Sin embargo, Quagga tiene un enfoque diferente, está compuesto por una **colección de varios demonios** que trabajan juntos para construir la tabla de encaminamiento, ver Figura 4.1.

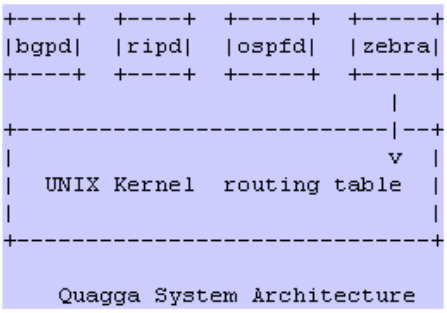



Figura 4.1: Arquitectura de Quagga

Como se observa, se tienen los demonios *ripd*, *ospfd* y *bgpd*, que soportan los protocolos RIP, OSPF versión 2 y BGP-4, respectivamente. A su vez, todos se comunican con el demonio gerente, *zebra*, el cual interacciona con el sistema operativo para la configuración general de las interfaces y para actualizar las tablas de encaminamiento del núcleo. Esta arquitectura **multiproceso modular** permite que el sistema Zebra/Quagga sea más fácilmente extensible y gestionable. Así, es fácil añadir un nuevo protocolo de encaminamiento sin afectar a ningún otro software; basta ejecutar sólo el demonio asociado con el protocolo de encaminamiento en uso.

### 4.2.1. Zebra

El protocolo Zebra [54] es utilizado por los demonios de los diferentes protocolos para comunicarse con el demonio de *zebra*. Cada demonio de protocolo podrá solicitar y enviar información desde y hacia el demonio de *zebra* como los estados de interfaz, estado de encaminamiento, atributo `NEXT_HOP`, etc. Los demonios pueden instalar rutas a través de *zebra*. El demonio *zebra* gestionará qué ruta se instala en la tabla de rutas del núcleo.

*Zebra* se puede configurar, por un lado, mediante una serie de ficheros de configuración y, por otro, pasándole comandos a los demonios en una conexión `telnet` al puerto en el que escucha cada demonio, el cual puede encontrarse en la misma máquina (`localhost`) o en una máquina remota.

Para la configuración de las interfaces de *Zebra* se dispone de los comandos `interface nombre_interfaz` e `ip address ip_de_la_interfaz`. Este comando es uno de los más importantes para realizar una configuración de *zebra* efectiva, ya que en el se encuentran todos las interfaces que utilizará para realizar el encaminamiento. Se deben especificar todas las que posea el encaminador.

Para la configuración del encaminamiento estático en *zebra* se utiliza el comando `ip route red puerta_de_enlace`. Este comando sirve para añadir una nueva ruta a la tabla de encaminamiento, lo cual consiste en indicar al encaminador que si le llega un paquete con una dirección IP de destino que no pertenece a las redes conectadas directamente, éste debe redirigirlo hacia otro encaminador por una de las interfaces. El parámetro `puerta_de_enlace` indica el encaminador al cual enviar el paquete cuya dirección IP destino coincida con el parámetro red anterior.

En la Figura 4.2 se puede ver un ejemplo de archivo de configuración de *zebra*, en el que aparecen además los siguientes parámetros.

`hostname encaminador`

Se utiliza para nombrar el encaminador con el que se está trabajando.

`password zebra`

Se indica la contraseña que se utilizará cuando se realice alguna conexión al encaminador.

`enable password zebra`

Se utiliza para establecer la contraseña para poder acceder al modo pri-

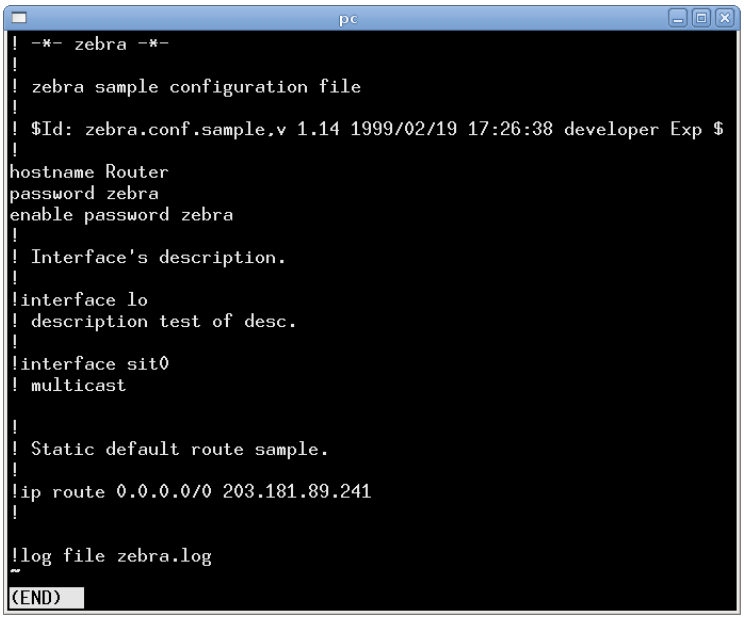



Figura 4.2: Ejemplo *zebra-conf*

vilegiado del encaminador.

### 4.2.2. Demonio bgpd

En concreto, el demonio *bgpd* implementa el protocolo BGP-4 (incluye soporte para direcciones multidifusión e IPv6). Cada demonio tiene su propio **fichero de configuración**. Así, para configurar un protocolo de encaminamiento en concreto se debe configurar el fichero correspondiente al demonio asociado. Por ejemplo, en el caso de *bgpd* el nombre del fichero de configuración es `bgpd.conf`. Este archivo se encuentra por defecto en la ubicación `/usr/local/etc/bgpd.conf.sample`, al que se debe cambiar el nombre por el mencionado anteriormente y después configurar.

#### Configuración

Para activar el proceso *bgpd* en el encaminador se utiliza el comando `router bgp asn`. En este comando se indica el número del AS al que pertenece el encaminador, lo que permite al proceso *bgpd* detectar si una sesión BGP-4

es interna o externa. El comando `bgp router-id id_router` sirve para especificar el identificador del encaminador en BGP a mano. El `id_router` se obtiene por defecto a partir de la información de las interfaces del demonio *zebra*, tomando como identificador la dirección de mayor numeración de todas las interfaces. Sin embargo, cuando el demonio *zebra* no está habilitado, *bgpd* no puede obtener la información de las interfaces y el identificador se fija a `0.0.0.0`, siendo necesario configurarlo a mano.

Una vez activado el demonio se introducen los comandos para la configuración básica de BGP: `neighbor` y `network`. Las sesiones BGP se establecen utilizando el comando `neighbor peer remote-as asn`. El valor de `peer` puede ser una dirección IPv4 ó IPv6 e indica la dirección del vecino con el que se establece la sesión BGP. Este vecino pertenecerá al AS cuyo número sea `asn`, de tal modo que si los números de AS de los dos encaminadores son iguales se establece una sesión iBGP, mientras que en caso contrario se establecerá una sesión eBGP.

Por otro lado, el comando `network prefijo [mask] [mascara]` activa el anuncio de la red indicada. El prefijo se puede introducir en notación Classless Inter-Domain Routing (CIDR)[35] o, por el contrario, mediante una máscara en decimal con la opción `mask`, si la red introducida no es de clases.

De este modo, en el fichero de configuración *bgpd.conf* se configuran el número del AS cliente, se activan las redes a anunciar con el comando `network` y se establece una sesión BGP con otro encaminador con el comando `neighbor`. La Figura 4.3 es un ejemplo de fichero de configuración de *bgpd* en el que aparecen además los siguientes comandos.

`hostname bgpd`

Se utiliza para nombrar el encaminador con el que se está trabajando.

`password zebra`

Se indica la contraseña que se utilizará cuando se realice alguna conexión al encaminador.

`log stdout`

Se utiliza para indicarle al encaminador que toda la información que se solicite al encaminador debe ser enviada al dispositivo de salida principal, es decir, la pantalla.

`enable secret password`

Se utiliza para establecer la contraseña para poder acceder al modo privilegiado del encaminador.

`line vty`

Indica que se utilizará una conexión vía `telnet` para poder realizar configuraciones al encaminador.

`log file`

Indica la ubicación donde es almacenará el fichero de log, normalmente `/var/log/zebra/bgpd.log`. Incluye actualizaciones de encaminamiento, cambios de estado de los vecinos, fecha, hora, tipo de paquete, dirección IP del vecino y otra información de encaminamiento.

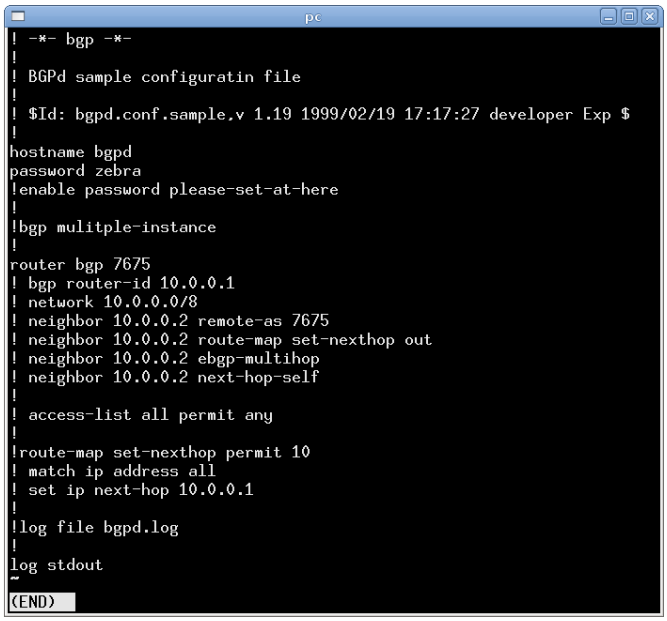



Figura 4.3: Ejemplo *bgpd-conf*

Para más información sobre estos y otros comandos ver Apéndice D.

## 4.3. Características

Actualmente Quagga soporta GNU/Linux, BSD y Solaris. Portar Quagga a otras plataformas no es demasiado difícil, ya que el código dependiente de la

plataforma debería ser limitado al demonio *zebra*. Los demonios de protocolos son en su mayoría independientes de la plataforma. La lista de las plataformas oficialmente soportadas se enumera a continuación. Hay que tener en cuenta que Quagga se puede ejecutar correctamente en otras plataformas, y puede ejecutarse con una funcionalidad parcial en plataformas adicionales.

- gnu/Linux 2.4.x y superior
- FreeBSD 4.x y superior
- NetBSD 1.6 y superior
- OpenBSD 2.5 y superior
- Solaris 8 y superior

## 4.4. Instalación

Véase Apéndice C.

## 4.5. Alternativas

Dentro de las herramientas de software con el propósito de encaminar, existen otras muchas alternativas a Quagga, algunas de las más destacadas e importantes se listan y detallan a continuación, debido a su funcionalidad y características específicas así como a su popularidad.

### 4.5.1. XORP

XORP viene del acrónimo *Extensible Open Router Platform*, y es un programa de código abierto que nace como un proyecto fundado por Mark Handley en el año 2000, aunque su primera versión se liberó en el 2004.

Este paquete está totalmente escrito en C++ y se considera como uno de los proyectos de encaminamiento que plantea mayor expectativa por su flexibilidad, ya que puede ser implementado tanto en ambientes Linux como Windows Server 2003, aunque en este último sólo para IPv4. XORP maneja protocolos de encaminamiento como RIP, OSPF, BGP-4, Protocol Independent Multicast (PIM)[31], Internet Group Management Protocol (IGMP)[18],

Multicast Listener Discovery (MLD)[94]. Se puede implementar encaminamiento tanto con direccionamiento IPv4 como con direccionamiento IPv6. Proporciona una interfaz de línea de comandos para la configuración interactiva, llamada *xorpsh*, que puede ser invocada por múltiples usuarios simultáneamente. El lenguaje de la línea de comandos se basa en el de la plataforma Juniper, JunOS.

Se distribuye bajo la licencia BSD[96] y cuenta con el apoyo económico de Intel, Microsoft, la National Science Foundation estadounidense y Vyatta.

### 4.5.2. BIRD

El proyecto BIRD, Internet Routing Daemon, fue desarrollado como un proyecto universitario en la Facultad de Matemáticas y Físicas de la Charles University de Praga. Tiene como objetivo desarrollar una plataforma de encaminamiento IP dinámico completamente funcional, dirigida principalmente, aunque no limitada, a sistemas de tipo UNIX. Se distribuye bajo la licencia GPL.

### 4.5.3. OpenBGPD

Es una implementación libre del protocolo BGP-4. Permite usar máquinas normales como encaminadores que hablan BGP con otros sistemas. Los objetivos de diseño para OpenBGPD incluyen ser seguro, fiable y lo suficiente ligero para la mayoría de usuarios tanto en el tamaño como uso de memoria. El lenguaje de configuración debería ser potente y fácil de usar. También debe ser capaz de manejar rápidamente cientos de miles de entradas en tabla de una manera eficiente en términos de memoria.

OpenBGPD está desarrollado por Henning Brauer y Claudio Jeker como parte del proyecto OpenBSD.

# Capítulo 5

# Escenario BGP-Wedgies

Se pretende utilizar el entorno de emulación **Netkit**, que permite crear máquinas virtuales y utilizarlas como si de encaminadores reales se tratase, junto con una modificación del software de encaminamiento **Quagga**, que almacena rutas de tráfico en formato comprimido (`gzip`), para reproducir y estudiar lo que se conoce como **BGP-Wedgies**.

## 5.1. Introducción

Con objeto de equilibrar la carga de tráfico, favorecer unos enlaces sobre otros, u optimizar la forma en que el tráfico entra o sale de la red, en la actualidad todo AS realiza su propia ingeniería de tráfico, aplicando las políticas locales que le parecen más adecuadas a los objetivos que desea conseguir. En consecuencia, se pueden producir anomalías en el comportamiento del protocolo de encaminamiento BGP como es el caso de BGP-Wedgies, objeto de estudio del presente capítulo.

### 5.1.1. Objetivos

Conseguir mediante una cierta configuración, un escenario en el que se obtenga un estado estable BGP no intencionado, y observar los posibles efectos que puede causar, tanto en las tablas de rutas como en el comportamiento del protocolo. Para ello se utilizará una modificación de Quagga que permita almacenar los datos de encaminamiento en formato comprimido para su posterior estudio.

## 5.2. Conceptos básicos

Como se explica en la Sección 2.6.1, BGP-Wedgies es un comportamiento no deseado de BGP que se produce como consecuencia de la interacción de políticas locales, y que da lugar a comportamientos no predecibles del protocolo.

Investigadores e ingenieros a menudo desean analizar el comportamiento de la red mediante el estudio de las transiciones de los protocolos de encaminamiento. Para facilitar este estudio, además de ayuda en temas de depuración, la mayoría del software de encaminamiento permite el registro de mensajes por parte de los protocolos. Vinculado a ello se crearon un formato para el almacenamiento de las rutas de tráfico y librerías que permiten su análisis.

### 5.2.1. Formato rutas de tráfico

El formato MRT [55] se desarrolló para encapsular, exportar y almacenar en una representación estandarizada la información de encaminamiento. Este formato representa una manera efectiva de almacenar la información de encaminamiento de BGP-4 en archivos de volcado binarios. Fue inicialmente definido en [56].

Con el fin de permitir el análisis de tráfico BGP, encaminadores y herramientas de supervisión habilitan el registro de mensajes de actualización BGP-4 y la creación de volcados de tablas de encaminamiento. Aunque la mayoría de las herramientas de encaminamiento usan su propio formato para los registros de mensajes de actualización de BGP-4, muchas de ellas permiten el uso del formato binario MRT. El software de encaminamiento Quagga es capaz de producir archivos de registro de texto legible para un análisis más detallado del comportamiento de las actualizaciones de BGP-4.

Mientras que los archivos de registro son normalmente muy grandes pues están destinados para fines de depuración y pueden contener muchas entradas innecesarias para el análisis de tráfico BGP-4, los archivos de volcado MRT contiene sólo los mensajes BGP-4, conservando su información completa en forma binaria y también consumiendo menos espacio de disco.

Quagga puede producir dos tipos diferentes de archivos de volcado MRT para BGP:

- **BGP4MP** que contienen los tipos de mensajes `UPDATE`, `NOTIFICA-`

`TION`, `KEEPALIVE` y `OPEN`, recolectándolos en el orden en que llegan al encaminador colector.

- **TableDumpV2** que contienen la tabla de rutas completa del encaminador, volcada cada cierto intervalo de tiempo para mantener actualizado el estado de la tabla de encaminamiento.

En el Apéndice F se puede ver el formato básico MRT según [55].

### 5.2.2. Colección de datos en Quagga

Esta sección describe como se puede habilitar la colección de datos en Quagga a través de los archivos de configuración *bgpd* o a través de la interfaz de la línea de comandos. Los comandos son los mismos en ambos casos.

En primer lugar, es necesario que los archivos donde los datos van a quedar registrados existan. También es necesario, que el usuario que ejecuta el proceso Quagga (normalmente el usuario *quagga*) tenga permiso de escritura en estos ficheros. La Tabla 5.1 muestra los comandos para activar mensajes de registro en Quagga. La Tabla 5.2 muestra cómo volcar los paquetes BGP en formato MRT, tanto de tipo `BG4MP` como `TableDumpV2`. Por último, en la Figura 5.1 se puede observar cómo se ha configurado la recopilación de datos a través del archivo de configuración *bgpd.conf* y cómo Quagga ha almacenado las rutas de tráfico en los archivos especificados.

### 5.2.3. Manipulación de las rutas de tráfico

Como los paquetes BGP y MRT pueden llegar a ser bastante complejos, es necesario el uso de librerías existentes para la manipulación de dichos paquetes. Existen un conjunto de librerías, para diversos lenguajes de programación, para extraer la información de los archivos de volcado MRT. Algunas de ellas son bastante completas, otras sólo proporcionan soporte básico.

#### libbgpdump

*libbgpdump*[3] es una librería escrita en el lenguaje de programación C diseñada para el análisis de los archivos de volcado producidos por Zebra/-Quagga o archivos en formato MRT. Actualmente es mantenida por RIPE dentro del proyecto RIPE's Routing Information Service (RIPE RIS) [78], proyecto para colectar y almacenar datos de encaminamiento de Internet a

| | |
|---|---|
| `log file bgpd.log` | Establece el archivo de registro de texto en el archivo `bgpd.log` en el directorio actual. Es necesario que el archivo exista y sea editable por Quagga |
| `debug bgp` | Habilita registro |
| `debug bgp events` | Habilita registro para eventos BGP-4 |
| `debug bgp updates` | Habilita registro para anuncios BGP-4 |
| `debug bgp fsm` | Habilita registro para eventos de máquina de estados |
| `debug bgp filters` | Habilita registro para filtrado de eventos |
| `debug bgp keepalives` | Habilita registro para mensajes `keepalive` |
| `debug bgp as4` | Habilita el registro de información para el tratamiento de los anuncios que contienen números de AS de 4 bytes |
| `debug bgp as4 segment` | Habilita el registro de información para segmentos de `AS_PATH` de anuncios que contienen `AS_PATH` de 4 bytes, además de los de 2 bytes |
| `debug bgp zebra` | Habilita el registro para eventos de Forwarding Information Base (FIB) |

Tabla 5.1: Comandos registro de texto Quagga

| | |
|---|---|
| `dump bgp all bgpd.dump` | Este comando vuelca todos los mensajes en formato de MRT en el archivo `bgpd.dump`. El archivo debe tener permisos de escritura por Quagga |
| `dump bgp routes-mrt rib/dump 24h` | Este comando vuelca la tabla de encaminamiento en el archivo de volcado dentro de la carpeta `rib`. El volcado se repite cada 24 horas, y la primera descarga se crea al principio de la hora siguiente cuando el comando es ejecutado. El formato de volcado es `TableDumpV2` |

Tabla 5.2: Comandos registro de texto Quagga

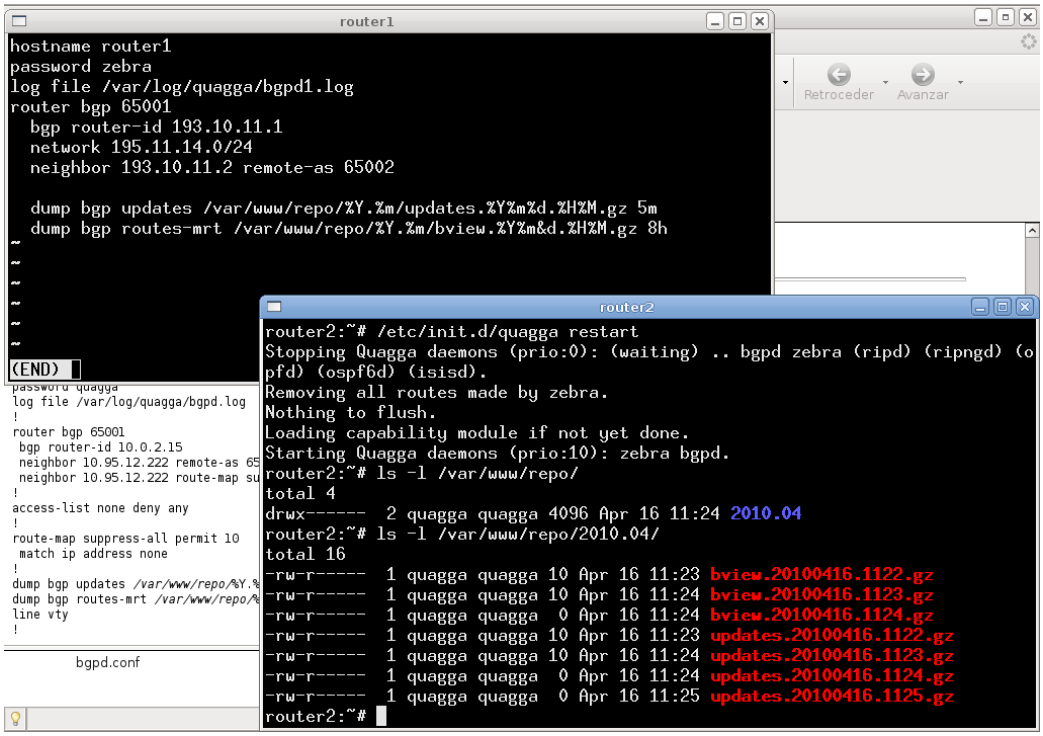



Figura 5.1: Recolección de datos en Quagga

través de Remote Route Collectors (RRCs) desplegados en diferentes Internet Exchange Point (IXP). Posteriormente dichos datos están disponibles para la comunidad de Internet para la solución de problemas e investigación. Hay dos grupos de ficheros disponibles:

- Todos los paquetes BGP creados con Zebra mediante el comando `dump bgp all`. Los nombres de los archivos empiezan por *updates* y son creados cada cinco minutos.
- La tabla de encaminamiento BGP completa creada con Zebra mediante el comando `dump bgp routes-mrt`. Los nombres de los archivos empiezan por *bview* y son creados cada ocho horas.

Las actualizaciones, retiros y anuncios respectivamente, presentan la siguiente sintaxis una vez decodificadas por esta herramienta:

```
TIME|TYPE|FROM|TO|WITHDRAW
```

```
TIME|TYPE|FROM|TO|ORIGIN|ASPATH|NEXT_HOP|MULTI_EXIT_DISC| COM-
MUNITY|ANNOUNCE
```

## 5.3. Modificación Quagga

La modificación realizada sobre el software de encaminamiento Quagga consiste en almacenar las rutas de tráfico en **formato comprimido** `gzip` debido al límite de almacenamiento de las máquinas virtuales. `gzip` produce archivos con extensión `.gz` y se basa en el algoritmo *deflate*. Para hacer más fácil el desarrollo del software que usa compresión, se creó la biblioteca `zlib`. Soporta el formato de ficheros `gzip` y la compresión *deflate*. El formato de compresión `zlib`, el algoritmo *deflate* y el formato `gzip` fueron estandarizados como [25], [23] y [24] respectivamente.

No se debe confundir `gzip` con *ZIP*, el cual no es compatible. El primero sólo comprime archivos, pero no los archiva. Debido a esto a menudo se usa junto con alguna herramienta para archivar (popularmente `tar`).

En el Apéndice E se puede encontrar un archivo *diff* [62] con las diferencias entre los archivos de Quagga originales y la modificación realizada.

## 5.4. Entorno de ejecución

Como se indica en la Sección 4.3, Quagga soporta las plataformas GNU/-Linux, BSD y Solaris. Para el desarrollo del presente escenario se ha utilizado por completo Linux, en concreto Ubuntu 8.04, la versión 2.4 del software de emulación Netkit y la versión 0.99.10 de Quagga.

Para la instalación, configuración y ejecución del escenario se han seguido los pasos que se indican en los siguientes apartados, suponiendo instalado el software de emulación Netkit, cuya instalación puede consultarse en el Apéndice A.

### 5.4.1. Creación del paquete Quagga

Una vez realizadas las modificaciones oportunas en el código Quagga y compilado éste sin errores, se ha de crear un paquete de software Debian [1], `.deb`, para instalarlo posteriormente en las máquinas virtuales de Netkit. Se ha de tener el entorno correctamente configurado con los paquetes necesarios

(`dh-make`, `fakeroot`, `build-essential`, `dpkg-dev`) para evitar errores en la creación del paquete. Para este propósito se ha utilizado el comando

```
fakeroot debian/rules clean binary
```

Este comando permite mediante `fakeroot` crear archivos (tar, deb) con ficheros con permisos de superusuario. Mediante `debian/rules clean` se limpia el árbol del código, y finalmente mediante la orden `binary` se construye el paquete binario con el nombre *quagga_versión_i386.deb* en nuestro caso. En la Figura 5.2 podemos ver la salida en el terminal de la ejecución del comando.

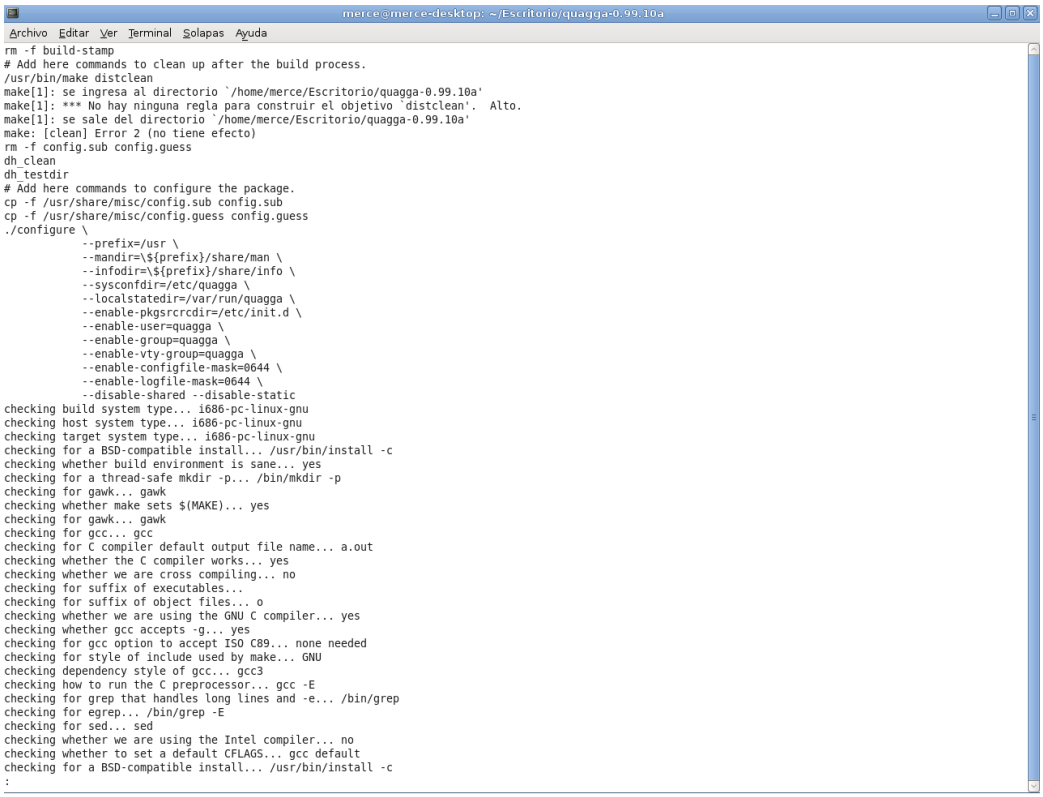



Figura 5.2: Salida por pantalla de la creación de un paquete Debian

### 5.4.2. Instalación de Quagga en Netkit

Con el fin de experimentar con los protocolos de encaminamiento, Netkit viene con una versión instalada del software de encaminamiento Zebra [54].

Sin embargo, nuestro propósito es que las máquinas virtuales funcionen con el software de Quagga que se ha modificado, para lo cual se ha de instalar dentro de Netkit el paquete Debian creado.

Para comenzar, se ha de arrancar una máquina virtual para **escritura**, con la opción `-W`, de modo que se cambia el sistema de archivos de forma permanente y, en consecuencia, se aplican los cambios a todas las máquinas virtuales que se arranquen posteriormente. También es recomendable arrancar la máquina con un aumento de memoria, con la opción `-M 256` por ejemplo, por si fuera necesario durante la instalación del paquete. Una solución alternativa es montar el sistema de ficheros en la máquina real y realizar la instalación sin iniciar ninguna máquina virtual. Para proceder de esta manera es necesario tener privilegios de superusuario en la máquina real.

Una vez llegado a este punto, primero, por seguridad, eliminamos el software Quagga que haya instalado con el gestor de paquetes `apt-get remove` y todos los ficheros de configuración, opción `-purge`. A continuación, simplemente instalamos el paquete Debian con el sistema de instalación de paquetes de Debian, `dpkg -i`.

Después de instalar el paquete, se ha detener la máquina virtual de forma segura usando `halt`, o en su defecto con el comando `shutdown`, que también permite apagar el sistema de modo seguro. De lo contrario, daría lugar a un sistema de ficheros corrupto, que haría que todas las máquinas virtuales iniciadas a partir de ese momento realizasen una comprobación en su inicio.

### 5.4.3. Ejecución del escenario

Con la finalidad de conseguir un escenario fácilmente portable y de sencilla ejecución, se ha desarrollado un laboratorio de Netkit, que se explica detalladamente en la Sección 5.5, de forma que para iniciarlo basta con utilizar dentro del directorio raíz el comando `lstart`, Figura 5.3. Los *scripts* de inicio instalados ponen en marcha los demonios *zebra* y *bgpd* con los argumentos esperados, y los archivos de inicio de cada encaminador se han modificado de forma que cuando arranquen lean la configuración y no sea necesario hacerlo manualmente.

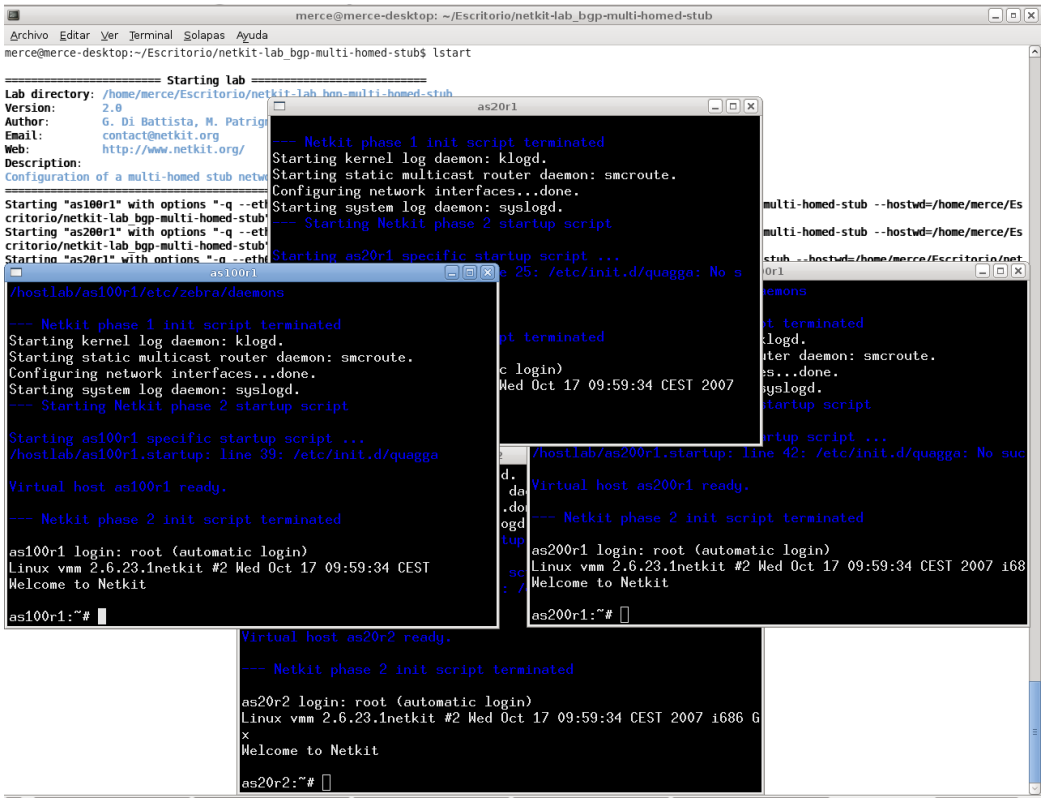



Figura 5.3: Ejecución del escenario con `lstart`

## 5.5. Escenario de estudio

Netkit nos permite crear escenarios de red fácilmente redistribuibles mediante la creación de lo que se llama un **laboratorio**, de forma que se pueden mover a otra máquina o equipo simplemente comprimiéndolos en un archivo, normalmente un `tar.gz`. Hay que tener en cuenta que no hay necesidad de mover imágenes de sistemas de ficheros potencialmente grandes, un laboratorio puede ser comprimido en un archivo muy pequeño y por tanto, ser transferido muy rápido, por ejemplo, a través de correo electrónico.

Un laboratorio es un conjunto completo de máquinas virtuales preconfiguradas que se pueden arrancar o parar conjuntamente mediante el uso de las herramientas *ltools*, que pueden ser consultadas en el Apéndice B. En realidad, un laboratorio consiste en una jerarquía de ficheros y directorios, como se puede ver en la Figura 5.4, que tienen permisos especiales.

- **Directorios**: cada directorio especifica la existencia de una máquina virtual llamada con el mismo nombre que el directorio. Los archivos contenidos en un directorio especifican las opciones de configuración de cada máquina virtual y son copiados automáticamente al directorio raíz `/` del sistema de ficheros de la máquina virtual correspondiente. Opcionalmente puede haber un directorio `shared` cuyo contenido se copia al directorio raíz de cada una de las máquinas virtuales que componen el laboratorio.

- `lab.conf`: fichero que describe la topología de red del escenario y otras configuraciones de las máquinas virtuales si fuera necesario, como por ejemplo la cantidad de memoria asignada.

- **Scripts de inicio y parada**: se utilizan para aplicar ciertas configuraciones durante la fase de inicio, ya sean específicas de una máquina virtual o compartidas.

- `_test`: directorio que contiene unos *scripts* especiales para descargar y comprobar el estado de las máquinas virtuales.

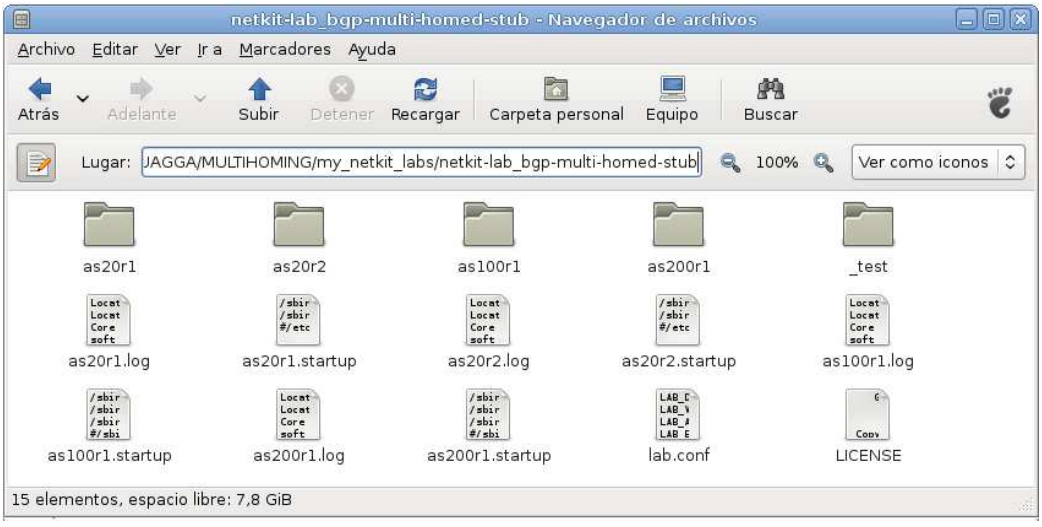



Figura 5.4: Estructura de directorios de un laboratorio Netkit

### 5.5.1. Definición de la topología

Antes de empezar a configurar el laboratorio es muy recomendable hacer un mapa detallado de la topología de red que se va a implementar. Para ello se utiliza el formalismo propuesto por Netkit[67] y que es el utilizado en toda su documentación, ver Figura 5.5.

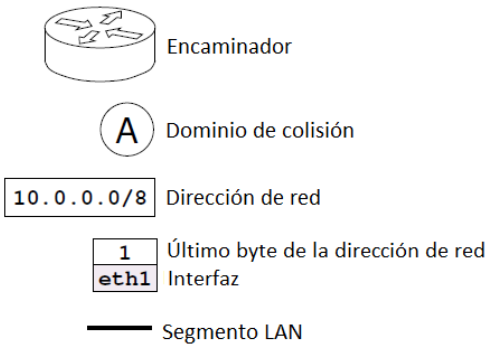



Figura 5.5: Componentes laboratorio Netkit

Como figura en la leyenda, el primer símbolo representa **encaminadores** en nuestro caso, pero podría asociarse a cualquier otro dispositivo emulado. Los círculos con una letra representan **dominios de colisión**, la letra sirve como identificador para el dominio de colisión, y será utilizada por Netkit para asociarla a un concentrador virtual. Cada dominio de colisión está enlazado con un rectángulo que contiene la **dirección de red** de la correspondiente subred. Los recuadros alrededor de los encaminadores detallan información sobre las **interfaces de red**: la mitad inferior contiene el nombre de la interfaz, y la mitad superior especifica el último byte de la dirección IP de la subred a la que pertenece. Por último, una línea de trazo continuo grueso representa una **red local**.

La Figura 5.15, al final del capítulo, representa la topología del escenario de wedgies implementado. Para conseguir este escenario final se han ido probando escenarios más pequeños para ver cómo funcionaban unas políticas locales en presencia de otras. En el Apéndice G se encuentran todos los archivos de inicio de los encaminadores que forman el laboratorio junto con el archivo de configuración del mismo.

### 5.5.2. Implementación de la topología

Como se ha explicado en la Sección 5.5 se ha de crear un directorio por cada máquina virtual que se haya especificado en la topología.

En segundo lugar hay que definir la topología de red en el archivo `lab.conf`, Figura 5.6. La primera parte del archivo contiene una información descriptiva opcional sobre el laboratorio, que puede ser útil cuando se distribuye a otras personas. El resto del archivo contiene un mapeado entre las interfaces de red y los dominios de colisión. Esta última información ha de especificarse en el formato `máquina[arg]=valor`, donde `máquina` es el nombre de la máquina virtual, `arg` es un número entero (por ejemplo `i`) y `valor` es el nombre del dominio de colisión al cual se enlazará la interfaz `ethi`. Opcionalmente, se pueden proporcionar otros parámetros de configuración, donde `arg` sea una cadena que se refiere al nombre de una opción `vstart` y `valor` es el argumento para dicha opción.

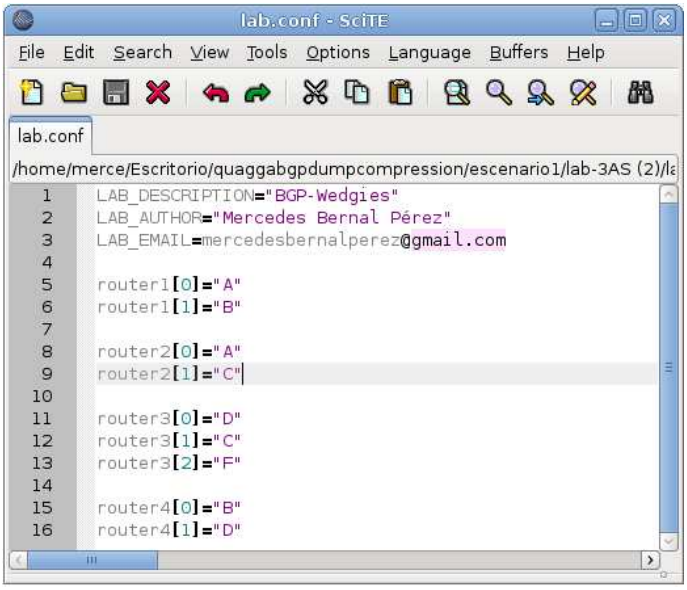



Figura 5.6: Archivo de configuración `lab.conf`

### 5.5.3. Configuración del laboratorio

Las máquinas virtuales pueden ejecutar comandos específicos en el arranque. Durante la fase de inicio, cada máquina virtual ejecuta los archivos `sha-`

`red.startup` y *`máquina`*`.startup`. Por tanto, estos scripts deben contener secuencias de comandos disponibles en el sistema de ficheros de las máquinas virtuales. Aparte de esta restricción, casi cualquier cosa se puede poner en los archivos de inicio. En la práctica son generalmente utilizados para asignar las direcciones IP a las interfaces de red y arrancar servicios de red.

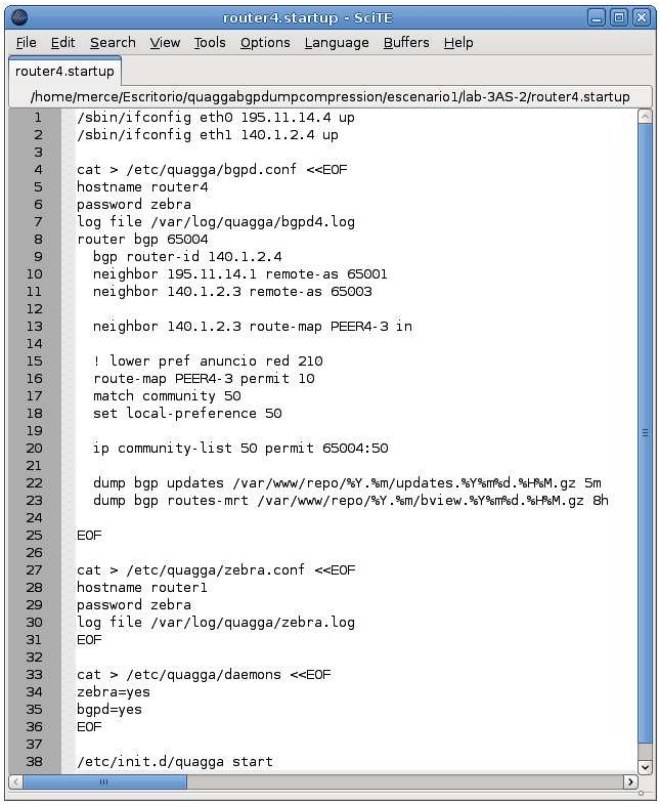



Figura 5.7: Archivo de inicio de un encaminador

En la Figura 5.7 se puede observar un archivo de inicio de uno de los encaminadores del laboratorio, que se explica a continuación. En las primeras líneas del archivo se configuran las direcciones de red para las interfaces de red, en concreto la dirección IP `193.10.11.1` para la interfaz `eth0` y la dirección IP `195.11.14.1` para la interfaz `eth1` como se observa en la Figura 5.8, detalle de la Figura 5.15.

En la última línea del fichero se indica el servicio de red a iniciar, en este caso Quagga. También han de proporcionarse los archivos de configuración.

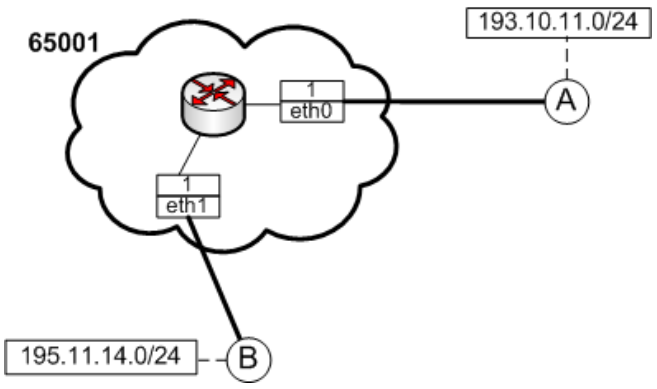



Figura 5.8: Laboratorio multihoming: detalle encaminador

En Netkit es posible mediante un mecanismo que hace que estén disponibles algunos archivos que forman parte del laboratorio dentro de las máquinas virtuales. En particular, antes de iniciar la máquina virtual, Netkit copia todos los archivos situados en el directorio asociado con esa máquina dentro de su sistema de ficheros. Este enfoque de reflejar archivos desde la máquina real tiene una doble ventaja: no añade ninguna sobrecarga a la configuración, y además, de esta manera no hay ninguna restricción sobre el número o tipo de servicios que pueden ser configurados.

En la configuración de *bgpd* se especifican atributos como `router-id`, `network` o `neighbor`. Para una descripción detallada de los atributos BGP se puede consultar el Apéndice D.

También se configura el almacenamiento de los ficheros con la información de almacenamiento mediante las líneas:

```
dump bgp updates /var/www/repo/%Y.%m/updates.%Y%m%d.%H%M.gz 5m
dump bgp routes-mrt /var/www/repo/%Y.%m/bview.%Y%m%d.%H%M.gz 8h
```

Por último, cabe destacar la siguiente configuración:

```
route-map PEER4-3 permit 10
match community 50
set local-preference 50
```

Que se corresponde con la modificación de la preferencia local en respuesta a una marca de comunidad BGP, cuya utilización se explica a continuación.

### Uso de comunidades

Una comunidad es un conjunto de prefijos que comparten una propiedad común y que puede ser configurada a través del atributo BGP `COMMUNITY`. Es un atributo opcional transitivo, cuya sintaxis se explica en el apartado 2.4.3.

Mientras que las comunidades por ellas mismas no alteran el proceso de decisión de BGP, pueden ser utilizadas como *flags* para marcar un conjunto de rutas [4]. Los encaminadores del ISP pueden utilizar estos indicadores para aplicar políticas específicas de encaminamiento dentro de su red, por ejemplo, `LOCAL_PREF` en nuestro caso.

Los proveedores establecen una correspondencia entre los valores configurables de las comunidades y los valores correspondientes a las preferencias locales dentro de la red del proveedor. La idea es que los clientes con políticas específicas que requieren una modificación del atributo `LOCAL_PREF` en la red del proveedor, establecen los valores correspondientes de las comunidades en sus actualizaciones de encaminamiento.

En la Figura 5.9 se observa que el consumidor identificado con el número de AS `AS100` anuncia el prefijo `100.1.2.0/24` con un atributo `COMMUNITY` igual a `7675:80`, entonces el ISP superior establece el atributo `LOCAL_PREF` de estas rutas a `80` si el atributo `COMMUNITY` es igual a `7675:80`.

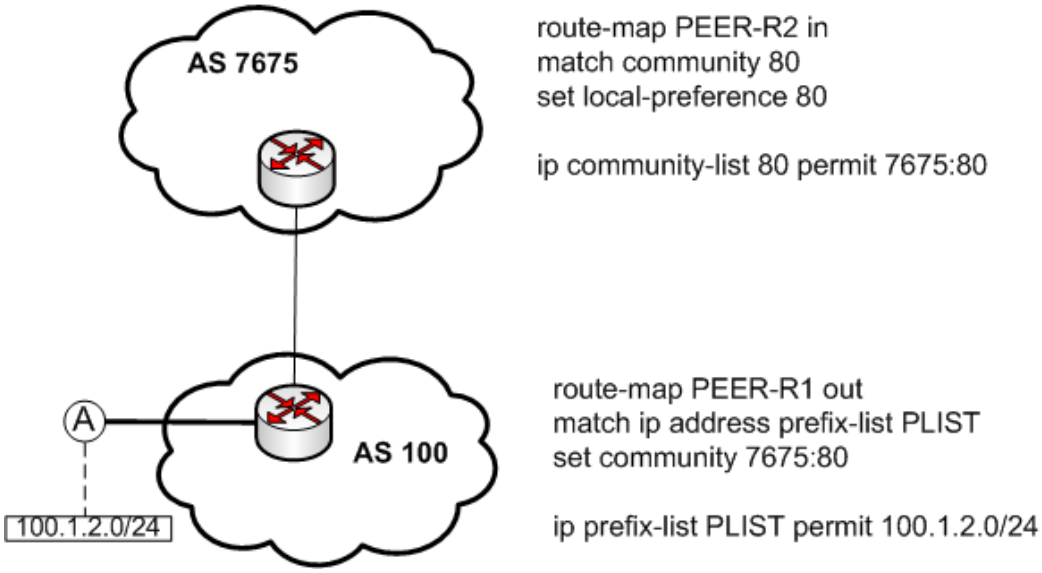



Figura 5.9: Atributo `COMMUNITY` con `LOCAL_PREF`

El uso de las comunidades BGP se ha incrementado a lo largo del tiempo, como se observa en la Figura 5.10. Por ejemplo, el encaminador Amsix (base de datos de RIPE) se encuentra con valores de comunidades BGP diferentes más de tres veces en Septiembre de 2007 que en Septiembre de 2004. Además, se nota una caída del crecimiento del uso de comunidades en Marzo de 2007.

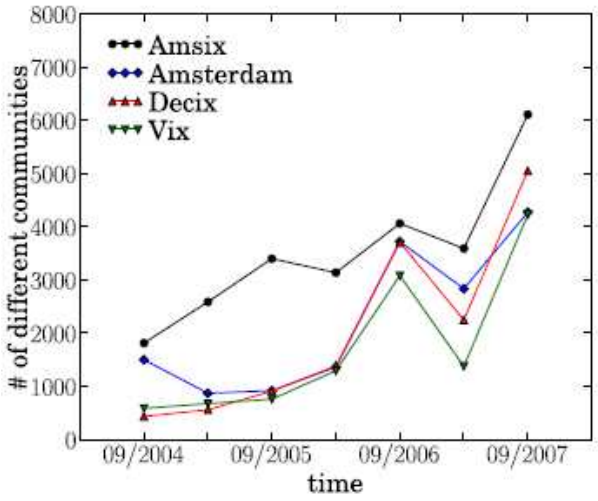



Figura 5.10: Evolución comunidades BGP en el tiempo

## 5.6. Resultados prácticos

Durante el estudio del escenario se han almacenado ficheros de registro para poder, una vez parado el escenario, estudiar el comportamiento de las rutas de tráfico y tener constancia de la oscilación y/o los cambios que han sucedido; para lo cual se ha utilizado la librería *libbgpdump* explicada en la Sección 5.2.3.

Una vez iniciado el laboratorio, en la Figura 5.11 se puede observar como todo el tráfico circula a través del enlace primario. Este es el comportamiento deseado de la política de salida del AS `65003`, en la que en el estado normal no circula tráfico a través del enlace secundario.

Esto es debido a la configuración mediante comunidades en la que el AS `65003` marca sus anuncios hacia `65004` como *backup*, y hacia el `65002` como principal. Entonces, el AS `65002` anuncia el prefijo `210.1.2.0/24` de `65003` a

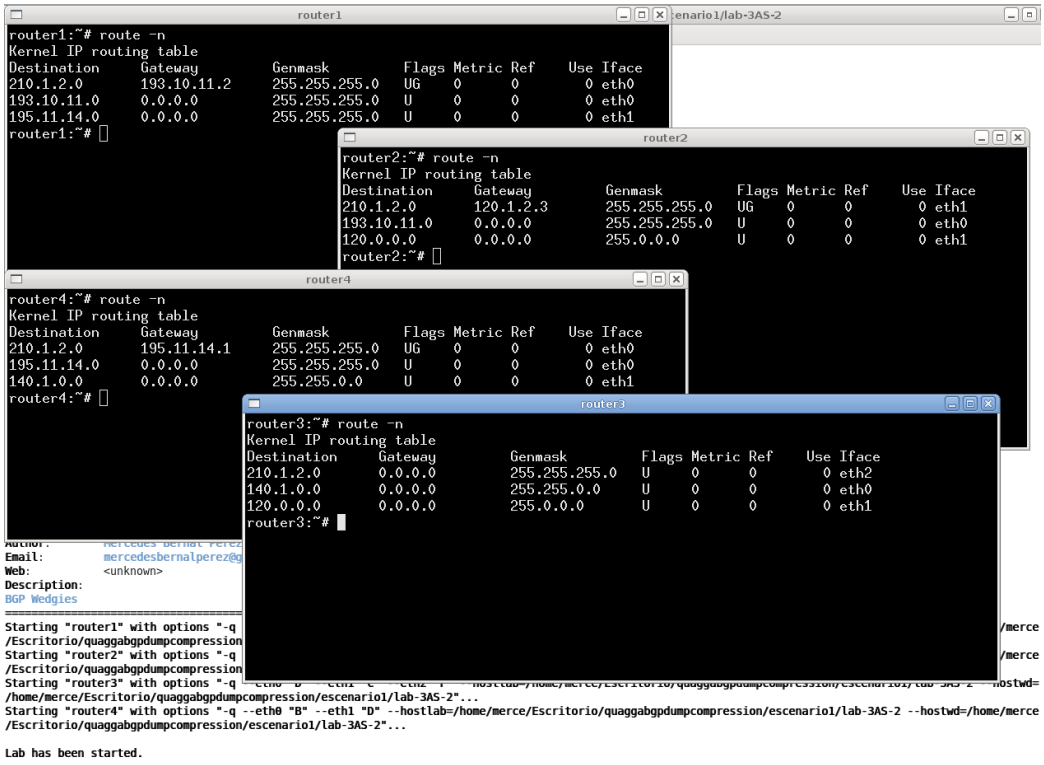



Figura 5.11: Laboratorio Wedgies: encaminamiento enlace principal

65001, el cual selecciona el prefijo con 65002 65003 y a su vez, lo anuncia al AS 65004, que como se observa en la Figura 5.12 recibe el anuncio también por el enlace directo. Sin embargo, selecciona la ruta con el camino más largo que la ruta de *backup*, pues se ha establecido un LOCAL_PREF menor para ella.

Asímismo, en la Figura 5.13 se observa después de decodificar el fichero MRT de encaminamiento, que recibe los dos anuncios con diferentes preferencias.

La siguiente Figura 5.14 muestra que si el enlace principal se cae, el tráfico se encamina entonces por el enlace secundario; los anteriores anuncios se retiran sucesivamente. De nuevo, este comportamiento es parte de la política establecida.

Finalmente, cuando se restablece la conexión en el enlace principal, el estado BGP no retorna a su configuración inicial. Obtenemos entonces, un estado BGP estable no intencionado, que es lo que se conoce como *wedgie*.

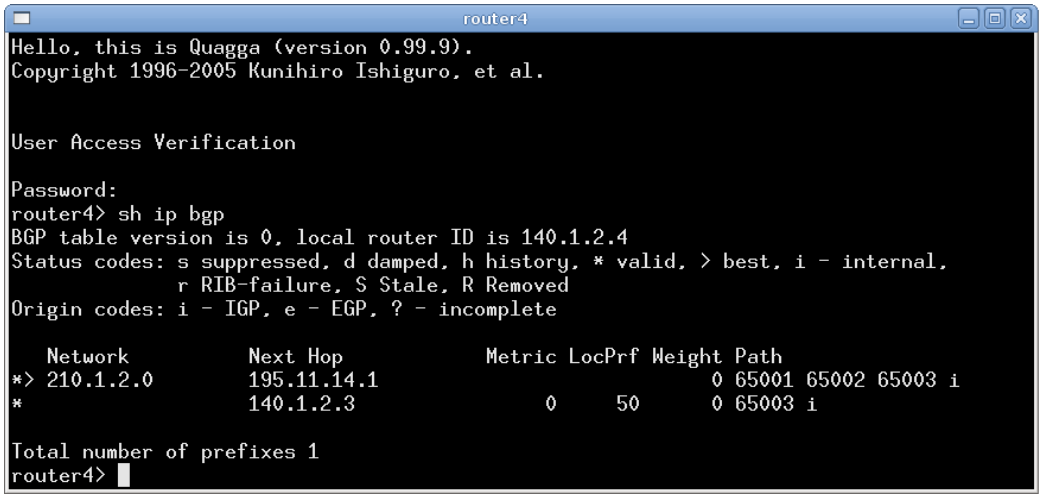



Figura 5.12: Preferencia local más baja para la ruta de *backup*

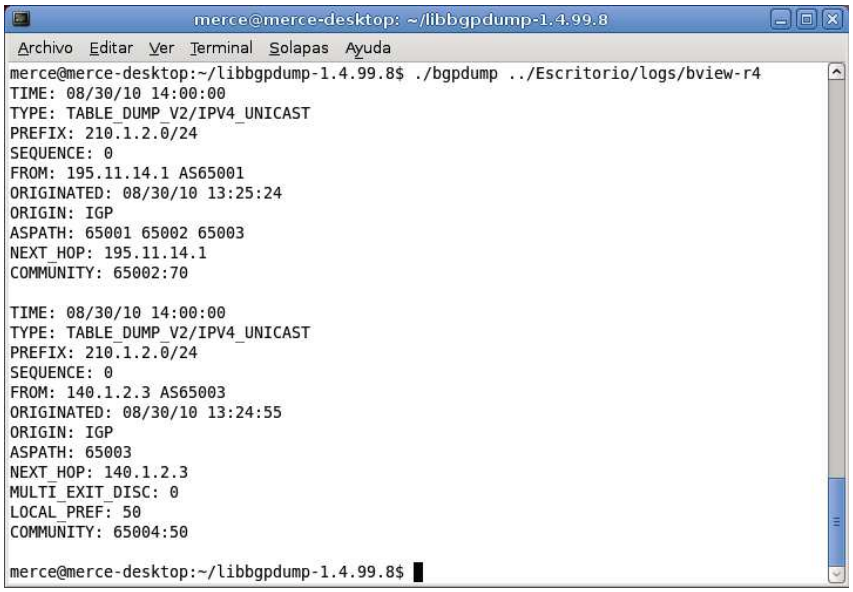



Figura 5.13: Fichero MRT de encaminamiento con *libbgpdump*

En resumen, aunque es frecuente el uso de comunidades para influir en decisiones de encaminamiento, es muy difícil predecir el impacto que pueden producir. No sólo depende de por dónde se envíe, sino también de si se asocia

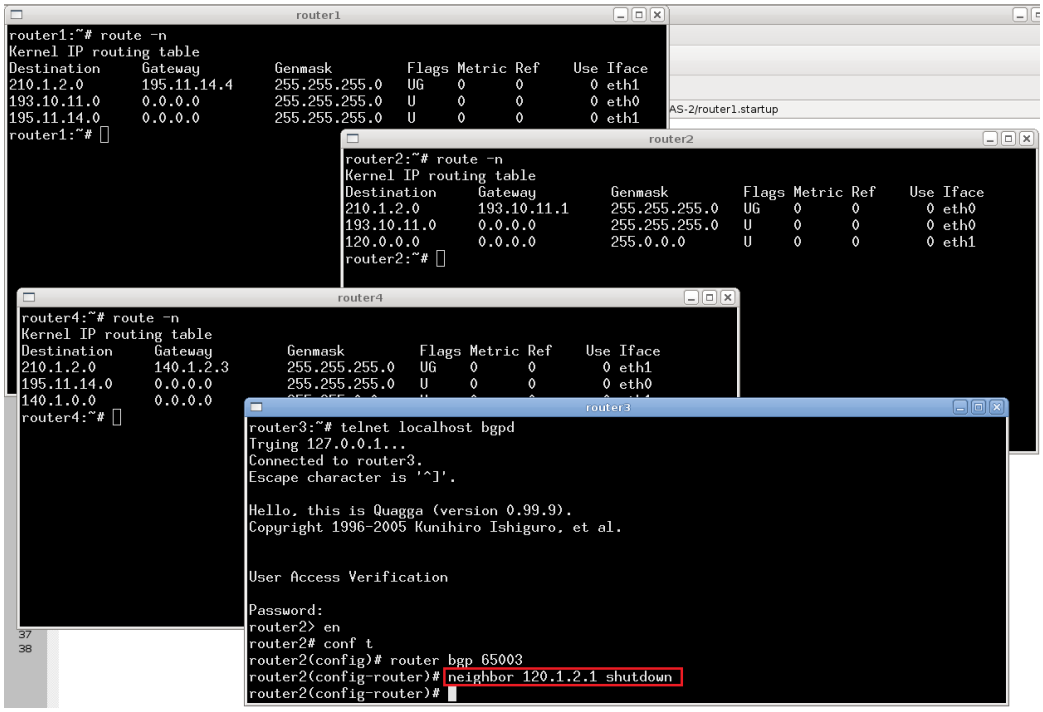



Figura 5.14: Laboratorio Wedgies: encaminamiento enlace *backup*

con otras comunidades o no. Además, otro inconveniente a tener en cuenta es que el AS tendrá que elegir entre un gran número de diferentes comunidades.

Se han propuestos enfoques más novedosos para permitir a un AS controlar su tráfico entrante de una manera determinista, como el de B. Quottin [74], cuyo enfoque se basa en el establecimiento de "Virtual Peerings" entre ASs que cooperan. Afirma que es escalabale y que se puede desplegar hoy en Internet con poco esfuerzo. También existen más propuestas [90].

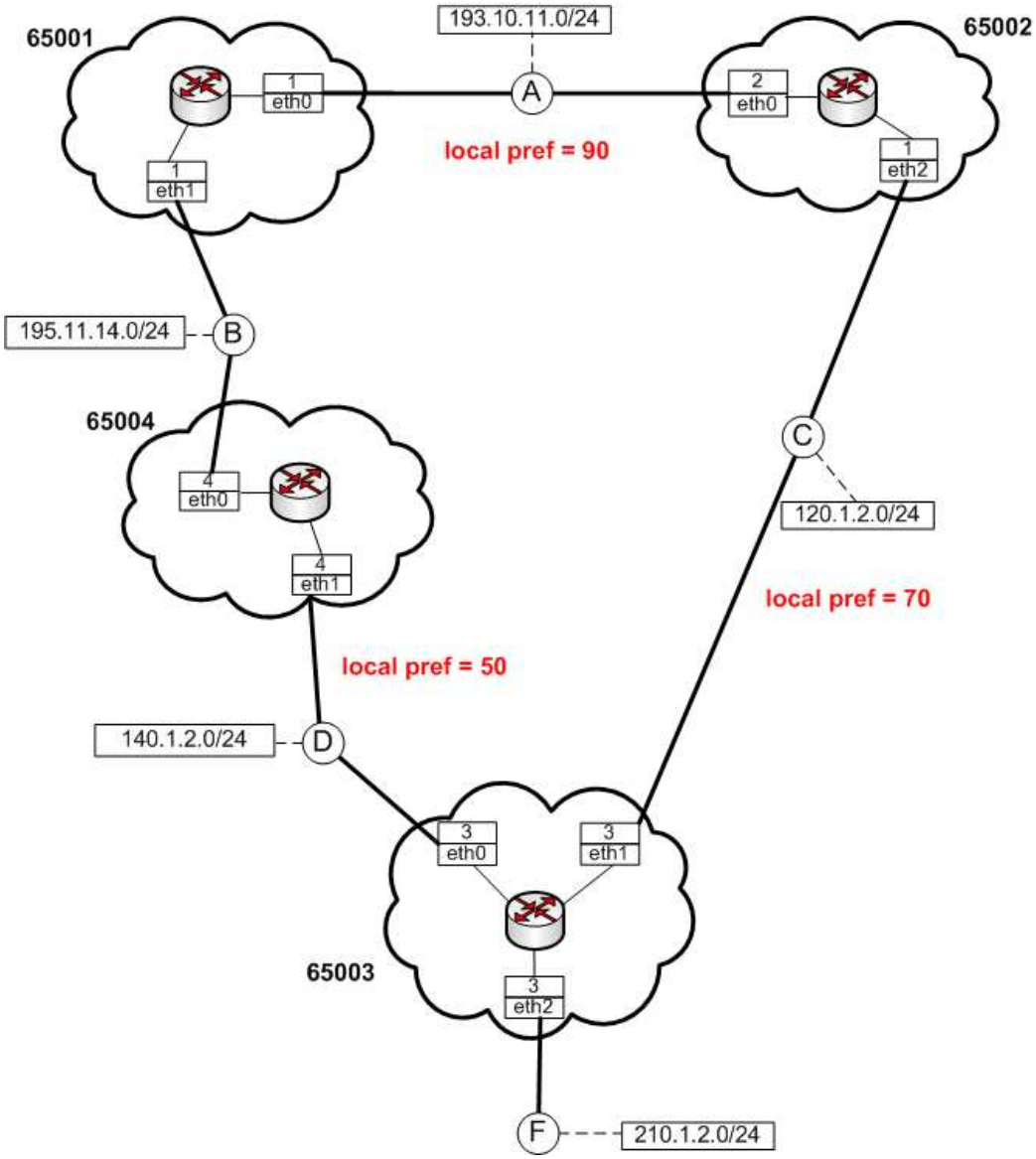



Figura 5.15: Laboratorio BGP-Wedgies: topología de red

# Capítulo 6

# Escenario Multihoming

Se pretende utilizar el entorno de emulación **Netkit**, que permite crear máquinas virtuales y utilizarlas como si de encaminadores reales se tratase, junto con una modificación del software de encaminamiento **Quagga**, que actualiza las rutas además de en la tabla por defecto en otra cuyo número se pasa por parámetro, para reproducir y estudiar lo que se conoce como **multihoming**.

## 6.1. Introducción

Actualmente instituciones de toda índole y tamaño requieren conectividad global para el funcionamiento cotidiano de las mismas. Esto implica que una interrupción en el acceso a Internet supone un alto costo, por lo que existe una fuerte demanda de mecanismos que brinden un alto nivel de tolerancia a fallos en la conexión a Internet. Una opción cada vez más utilizada para ofrecer tolerancia a fallos consiste en contratar múltiples accesos a Internet a través de distintos ISP, lo que se conoce normalmente como **multihoming** [15], y es hoy por hoy un componente esencial para muchos sitios conectados a Internet.

De acuerdo con Subramanian y otros [86] una gran parte de los dominios están multiconectados. Entre los 16.921 dominios diferentes vistos en su análisis, 13.872 (82 %) fueron dominios conectados a un único ISP. Entre estos dominios, 8453 (61 %) tienen al menos dos proveedores diferentes. Se muestra en la Figura 6.1 un desglose de dominios en función del número de proveedores. Se observa que la mayoría de los dominios multiconectados estaban

dualmente conectados. Es también bastante frecuente el número de dominios que tienen más de dos proveedores.

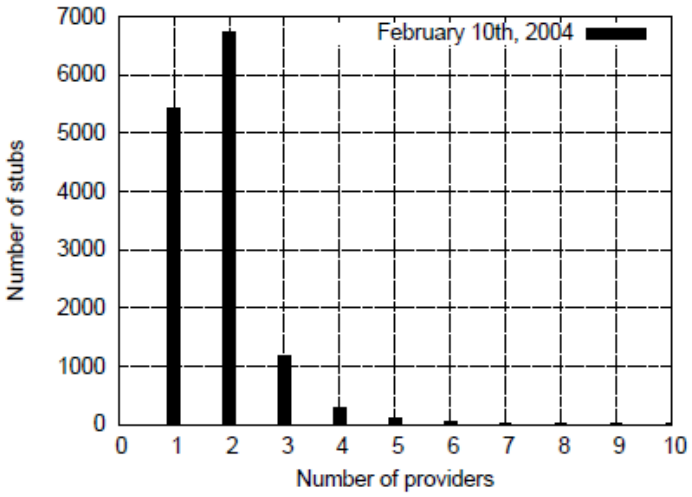



Figura 6.1: Desglose de ASs por el número de proveedores

### 6.1.1. Objetivos

Las técnicas actuales empleadas para conseguir **multihoming** [84] se apoyan en los principios básicos de la TE, vistos en la Sección 2.5. Es decir, el uso de atributos como `LOCAL_PREF`, `MED` y `AS_PATH` para influir en las decisiones de encaminamiento de BGP-4 y hacer fluir el tráfico por donde se desea en base a cuestiones de carga de tráfico, acuerdos de tipo de servicios, intereses económicos o cualquier otra política definida. La interacción de estas políticas puede dar lugar a comportamientos no deseados del protocolo como oscilaciones permanentes o comportamientos no deterministas como BGP-Wedgies, estudiadas en el escenario del Capítulo 5.

En este escenario se plantea una alternativa a estas técnicas tradicionales de TE para el concepto de *multihoming*. La idea principal consiste en la ejecución simultánea de dos demonios del protocolo BGP-4, cada uno de los cuales escribe en una tabla de rutas diferente. De este modo se consigue tener un escenario con un AS multiconectado, evitando políticas locales que pueden producir anomalías globales no deseadas.

## 6.2. Conceptos básicos

Se define multihoming como la conexión de una máquina o sitio a **más de un ISP a la vez**. Esta técnica permite aumentar la fiabilidad de conexión a Internet en una red IP. En el caso que un enlace de la red falle, entonces el tráfico se redirige automáticamente por otro de los enlaces restantes. Un caso típico de multihoming se ilustra en la Figura 6.2. El sitio X se conecta a dos ISP que le dan acceso a Internet.

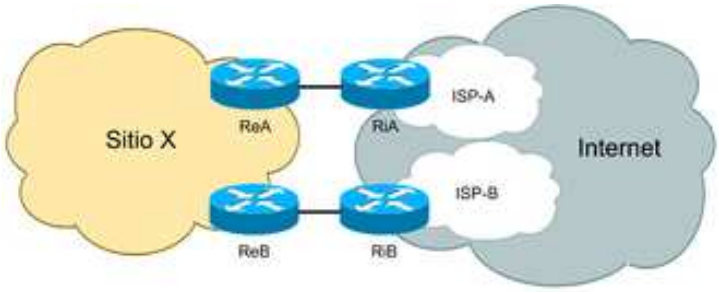



Figura 6.2: Caso típico de multihoming

El soporte de multihoming ofrece múltiples ventajas que hacen que sea la opción elegida por muchos clientes para asegurar su conectividad.

- **Tolerancia a fallos**. Un AS cliente con multihoming es más inmune a fallos de tránsito o de acceso. Si se detecta un fallo que afecta a uno de los ISP que le presta servicio, el AS puede utilizar un camino alternativo a través de otro proveedor. Esto funciona automáticamente debido al protocolo BGP, ya que los sitios remotos recibirán los anuncios correspondientes a las distintas rutas, eligiendo la que más les conviene basándose en criterios locales. Cuando ocurra un fallo en un enlace, la ruta afectada será retirada, por lo que los paquetes se encaminarán a través de las rutas alternativas que persistan.

- **Balanceo de carga**. Es posible distribuir el tráfico entrante y saliente entre los distintos ISP a los que se está conectado, de forma que se tiende a maximizar la utilización de los recursos.

- **Ingeniería de tráfico**. El AS cliente puede decidir qué tipo y qué volumen de tráfico enviar a cada ISP basándose en aspectos tales como

los acuerdos de nivel de servicio, el costo de enviar determinado tipo de tráfico a cada ISP o cualquier otra política que se haya definido.

- **Independencia de los ISP**. Normalmente un sitio busca no depender de los ISP para poder gozar de las ventajas del multihoming. Es por eso que las soluciones de multihoming suelen tener en cuenta que no se necesite ningún tipo de colaboración especial por parte de los ISP para implementarlas. Lo que implica que cada sitio puede contratar a distintos ISP de forma independiente e implementar multihoming por cuenta propia.

### 6.2.1. Diferentes implementaciones

Existen diferentes métodos y estrategias [7] para implementar el concepto de multihoming. A continuación se describirán algunos métodos que se utilizan en IPv4, no siendo estos los únicos disponibles pero sí los más comunes.

#### Multihoming con direcciones independientes del proveedor

La forma más común de implementar multihoming en IPv4 es obtener un bloque de direcciones independientes del proveedor, Provider Independent (PI), direcciones IP asignadas directamente por un RIR, junto con un número de AS y anunciar el bloque de direcciones vía BGP a cada uno de los ISP a los que se está conectado.

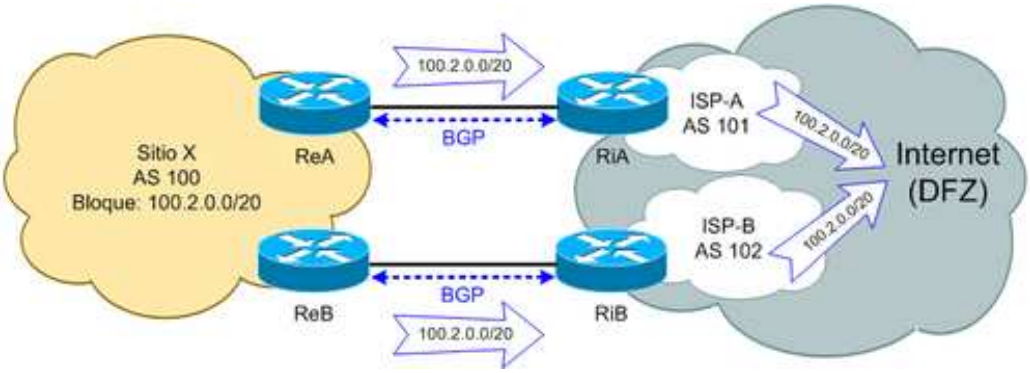



Figura 6.3: Ejemplo de multihoming IPv4 con direccionamiento independiente del proveedor

La Figura 6.3 muestra un escenario como el descrito, donde un sitio obtiene del registro de Internet correspondiente tanto el AS 100 como el bloque 100.2.0.0/20 y anuncia este bloque a los dos ISP a los que está conectado utilizando el protocolo BGP. Luego estos ISP se encargarán de anunciar a Internet que son capaces de encaminar tráfico destinado hacia tal bloque. De modo que al propagarse esta información por Internet todos los encaminadores que tengan la tabla de encaminamiento global sin ruta por defecto (DFZ) tendrán una entrada correspondiente al bloque 100.2.0.0/20 y podrán encaminar trafico hacia el mismo.

Este esquema presenta problemas de escalabilidad, puesto que por cada sitio con multihoming los encaminadores de la DFZ deben mantener una entrada en su tabla de encaminamiento. Por todo esto es que el multihoming con direcciones independientes del proveedor en IPv4 está reservado a sitios medianos a grandes.

### Multihoming con direcciones asignadas por el proveedor

Si a un sitio no le es posible hacer multihoming con direcciones independientes del proveedor aún le es posible obtener un bloque de direcciones de uno de los ISP que le prestan servicio. Estos bloques de direcciones se conocen como Provider Aggregatable (PA), direcciones IP asignadas por un RIR a un ISP que pueden ser incluidas en un anuncio de una sola ruta para mejorar la eficiencia de encaminamiento de Internet, y que forman parte de la arquitectura CIDR [36].

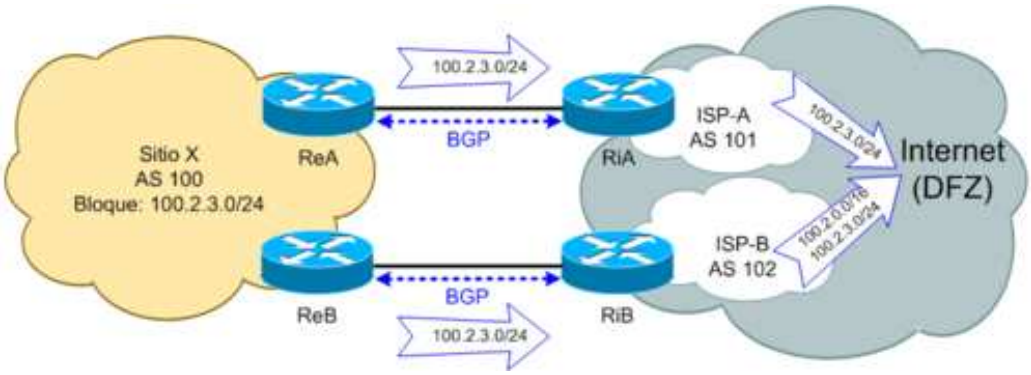



Figura 6.4: Ejemplo de multihoming IPv4 con direcciones asignadas por el proveedor

El AS cliente anuncia su bloque de direcciones por BGP a todos los ISP, los cuales a su vez lo anuncian hacia Internet. Pero el ISP que delegó el bloque, además de anunciar el bloque del AS cliente, anuncia su bloque mayor que lo engloba. De esta forma, aunque el bloque pequeño fuese filtrado en algún lugar de la red, al menos quedaría una ruta hacia el bloque que le dio origen. Esto se conoce con el nombre de agregación [19] [35] [53].

La Figura 6.4 muestra un ejemplo de este caso. El Sitio X está conectado a dos ISP y obtiene el bloque de direcciones 100.2.3.0/24 de ISP-B a quien su RIR le ha asignado el bloque 100.2.0.0/16. El Sitio X anuncia vía BGP a ambos ISP su bloque 100.2.3.0/24, utilizando para ello su número de AS 100. Tanto ISP-A como ISP-B vuelven a anunciar el prefijo 10.2.3.0/24 hacia Internet, pero ISP-B también anuncia su prefijo más grande 100.2.0.0/16. En condiciones normales el tráfico fluiría hacia el Sitio X por ambos ISP. Pero si algún ISP dentro de Internet filtrase el bloque 100.2.3.0/24 por considerarlo muy pequeño solo quedaría en esa parte de la red el prefijo mayor 100.2.0.0/16 y el tráfico fluiría hacia el Sitio X vía ISP-B.

Este modelo de multihoming con direcciones asignadas por el proveedor permite gozar de buena parte de las ventajas del multihoming a sitios que no son lo suficientemente grandes como para obtener un bloque independiente del proveedor. Una de las desventajas, además del peligro del filtrado, es que si se cambia el ISP que provee las direcciones el sitio tiene que enumerarse de nuevo, y la enumeración es un proceso costoso en IPv4.

### Múltiples conexiones a un mismo ISP

Otro caso que proporciona algunas de las ventajas del multihoming es el hecho de disponer de múltiples conexiones a un mismo ISP. En realidad, como sólo se tiene un punto de salida hacia otro AS, el protocolo BGP-4 no es necesario y se puede configurar una ruta por defecto hacia la pasarela o encaminador de borde para los paquetes que tengan como destino el exterior, de forma que las tablas de encaminamiento quedan bastante sencillas. En este caso no se necesita contar con ningún prefijo en especial ni obtener un número de AS, pero se tiene el inconveniente que ante un fallo general del ISP se pierde conectividad.

### Traducción de direcciones de red (NAT)

Finalmente, cabe comentar otra forma de multihoming muy utilizada, la traducción de direcciones de red (NAT) [85]. En este caso un sitio de Internet realiza una numeración interna con direcciones privadas y traduce las mismas en los encaminadores de salida. De esta forma no hay impacto en la tabla global de encaminamiento. Éste es el único método de multihoming accesible a sitios pequeños, aunque también lo utilizan muchos sitios medianos y grandes.

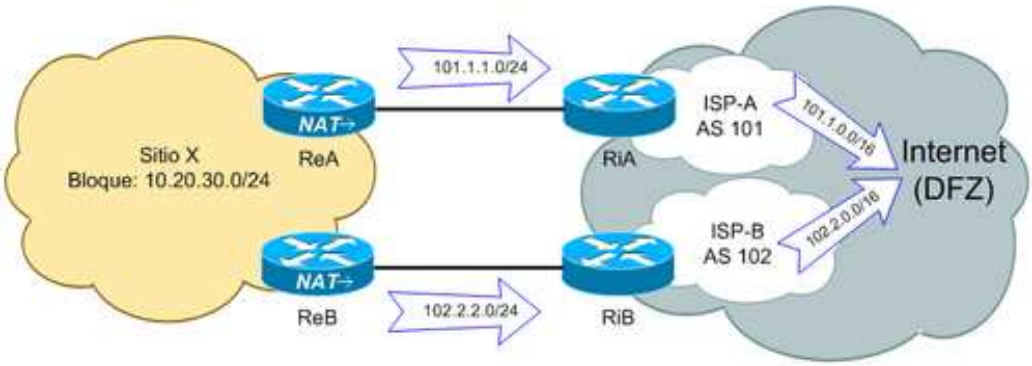



Figura 6.5: Ejemplo de multihoming en IPv4 utilizando NAT

En el ejemplo de la Figura 6.5 el Sitio X está conectado a dos ISP, y recibe un bloque de direcciones de cada uno de ellos. Del ISP-A recibe el bloque 101.1.1.0/24 y del ISP-B recibe el bloque 102.2.2.0/24. Internamente el sitio utiliza el bloque de direcciones privadas [77] 10.20.30.0/24. Cuando los paquetes salen hacia cada uno de los ISP las direcciones de origen se traducen a direcciones del bloque asignado al sitio por el ISP correspondiente, es decir que el tráfico que sale por ReA utiliza direcciones del bloque 101.1.1.0/24 y el tráfico que sale por ReB utiliza direcciones del bloque 101.2.2.0/24.

## 6.3. Modificación Quagga

La modificación realizada sobre el software de encaminamiento Quagga consiste en añadir la capacidad de escritura en una tabla de rutas diferente a la principal cuyo número se indica por parámetro al correspondiente demonio. Para conseguir este objetivo se han introducido modificaciones en el código

de los demonios *zebra* y *bgpd*, se han implementado unos *scripts* de inicio para lanzar dos demonios *bgpd* al mismo tiempo con sus respectivas configuraciones y además, ha sido necesario compilar de nuevo la versión utilizada del núcleo de Linux. Estas modificaciones se comentan en los siguientes apartados.

En el Apéndice H se puede encontrar un archivo *diff* [62] con las diferencias entre los archivos de Quagga originales y la modificación realizada.

### 6.3.1. Modificaciones demonios *zebra* y *bgpd*

Como se ha visto en el Capítulo 4, el demonio *zebra* es el encargado de actualizar la tabla de rutas del núcleo. Por defecto, la tabla de rutas que se actualiza es la tabla principal, que corresponde al número 0.

La modificación realizada referente a ***zebra*** ha consistido en redefinir el `NETLINK` [81] que utiliza Zebra para poder actualizar cualquier tabla de rutas del núcleo. *Netlink* se utiliza para transferir información entre el núcleo y los procesos de espacio de usuario, proporcionando enlaces de comunicación bidireccional entre ambos. Consta de una interfaz basada en *sockets* para los procesos de espacio de usuario y una API interna para los módulos del núcleo. Además, con objeto de evitar inconsistencias ha sido necesario redefinir todas las funciones que hacen uso del *NETLINK* para comunicarse con Zebra. Asímismo, en el servidor de Zebra ha sido necesario modificar las funciones existentes relacionadas con añadir, borrar o consultar una ruta. Todos los cambios se han hecho tanto para direccionamiento IPv4 como para IPv6.

Por otro lado, la modificación referente a ***bgpd*** ha consistido en añadir un parámetro más al demonio *bgpd* de modo que acepte un número de tabla como argumento, y en ausencia de éste se tome la tabla principal. Además, se ha modificado la estructura de información que el demonio *bgpd* le pasa al gestor *zebra* para poder incluir el número de tabla de rutas en la que se desea actualizar la ruta.

### 6.3.2. Scripts de inicio

Son un conjunto de tres scripts en un archivo `inits.tar.gz` para el arranque y configuración de Quagga, que se debe descomprimir en el directorio raíz de las máquinas virtuales y cuyo contenido se instala en las siguientes ubicaciones.

- Directorio `/etc/default`:

- *quagga*. Indica los demonios que se van a lanzar, en nuestro caso *zebra* y *bgpd*. Es equivalente a modificar manualmente el archivo *daemons*.
- *bgpd*. Para configurar los dos demonios *bgpd*. A través de este *script* se deja el primer demonio con la configuración por defecto, y al segundo demonio se le pasa el número de la tabla del núcleo en la que actualizará las rutas con la opción `-t`, el archivo `pid-file` por defecto en `/var/run/bgpd.pid` con la opción `-i` en el cual escribe el identificador de proceso cuando el demonio arranca y es posteriormente utilizado por el sistema para parar o reiniciar *bgpd*, el fichero de configuración con la opción `-f`, el número de puerto por el que escucha con la opción `-p` y el puerto *vty* con la opción `-P`

◦ Directorio `/etc/init.d`:

- *quagga*. Arranca los demonios con la configuración definida en los archivos anteriores y saca algunos mensajes de registro por pantalla,ver Apéndice J.

### 6.3.3. Compilación del núcleo de Linux

Como se ha introducido, ha sido necesario compilar el núcleo de Linux de la versión utilizada, concretamente 2.6.27.7 [1], para activar la opción de escribir en múltiples tablas para IPv4 y para IPv6, pues no viene activada por defecto, y que se corresponde con `IP_MULTIPLES_TABLES` y `IPv6_MULTIPLES_TABLES`. De lo contrario, la ejecución del escenario con la modificación de Quagga introducida no tenía ningún efecto, pues los demonios no podían actualizar las rutas en una tabla de rutas distinta a la principal.

En el Apéndice I se detalla el proceso seguido para realizar la compilación[2] de dicho núcleo para su utilización en Netkit.

## 6.4. Entorno de ejecución

De la misma forma que se ha implementado el escenario para el estudio del comportamiento de BGP-Wedgies, Capítulo 5, para el desarrollo del escenario

[1] http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.7.tar.bz2
[2] Páginas man, entrada *netkit-kernel*

de multihoming también se ha utilizado por completo Linux, en concreto Ubuntu 8.04, la versión 2.4 del software de emulación Netkit y la versión 0.99.10 de Quagga.

Para la instalación, configuración y ejecución del escenario se han seguido los pasos que se indican en los siguientes apartados, suponiendo instalado el software de emulación Netkit, cuya instalación puede consultarse en el Apéndice A.

### 6.4.1. Creación del paquete Quagga

La creación del paquete Quagga se ha llevado a cabo exactamente siguiendo el mismo procedimiento utilizado para la creación del paquete Quagga para BGP-Wedgies, consultar Sección 5.4.1.

### 6.4.2. Instalación de Quagga en Netkit

Como la instalación de un paquete Quagga en Netkit no depende del contenido de dicho paquete, consultar Sección 5.4.2.

### 6.4.3. Ejecución del escenario

Aunque el escenario es diferente al de BGP-Wedgies, ni tiene el mismo número de máquinas virtuales ni topología de red, la ejecución de un laboratorio en Netkit no es dependiente de dicha configuración, basta ejecutar dentro del directorio raíz el comando `lstart`. De la misma forma que para el desarrollo del escenario BGP-Wedgies, los *scripts* de inicio instalados ponen en marcha los demonios *zebra* y *bgpd* con los argumentos esperados, y los archivos de inicio de cada encaminador se han modificado de forma que cuando arranquen lean la configuración y no sea necesario hacerlo manualmente. El escenario implementado se explica detalladamente en la Sección 6.5.

## 6.5. Escenario de estudio

Del mismo modo que en el escenario de BGP-Wedgies, se ha explotado la capacidad de Netkit de creación de un laboratorio. En la Sección 5.5 se explica en qué consiste un laboratorio Netkit y la estructura de directorios que soporta.

### 6.5.1. Definición de la topología

Como se ha comentado ya, es recomendable hacer un mapa detallado de la topología de red que se va a implementar. De nuevo, se sigue el mismo procedimiento empleado en el escenario anterior, y se utiliza el formalismo propuesto por Netkit[67] para la representación gráfica del escenario, Sección 5.5.1.

La Figura 6.13, al final del capítulo, representa la topología del escenario de multihoming implementado. Aunque no aparece en dicha figura, se ha utilizado la misma topología para el direccionamiento IPv6, utilizando los comandos `address-family ipv6` y `exit-address-family`. En el Apéndice J se encuentran todos los archivos de inicio de los encaminadores que forman el laboratorio junto con el archivo de configuración de dicho laboratorio.

### 6.5.2. Implementación de la topología

Ver Sección 5.5.2.

### 6.5.3. Configuración del laboratorio

Como se explica en la Sección 5.5.3, las máquinas virtuales de Netkit pueden ejecutar instrucciones en el arranque. En concreto, durante la fase de inicio ejecutan los archivos `shared.startup` y *`máquina`*`.startup`, los cuales deben contener secuencias de comandos válidos en su sistema de ficheros.

En la Figura 6.6 se puede observar un archivo de inicio de uno de los encaminadores del laboratorio, que se explica a continuación. En las primeras líneas del archivo se configuran las direcciones de red para las interfaces de red, en concreto la dirección IP `210.1.2.1` para la interfaz `eth0` y la dirección IP `193.10.11.1` para la interfaz `eth1` como se observa en la Figura 6.7, detalle de la Figura 6.13.

En la última línea del fichero se indica el servicio de red a iniciar, en este caso Quagga. También han de proporcionarse los archivos de configuración. Como se comenta en la Sección 5.5.3, en Netkit es posible gracias a que antes de iniciar la máquina virtual, Netkit copia todos los archivos situados en el directorio asociado con esa máquina dentro de su sistema de ficheros. En nuestro caso, le indicamos no sólo la ubicación de los archivos de configuración en el archivo de inicio, sino también el contenido, para no necesitar configurar nada una vez iniciadas las máquinas virtuales. En la configuración de *bgpd*

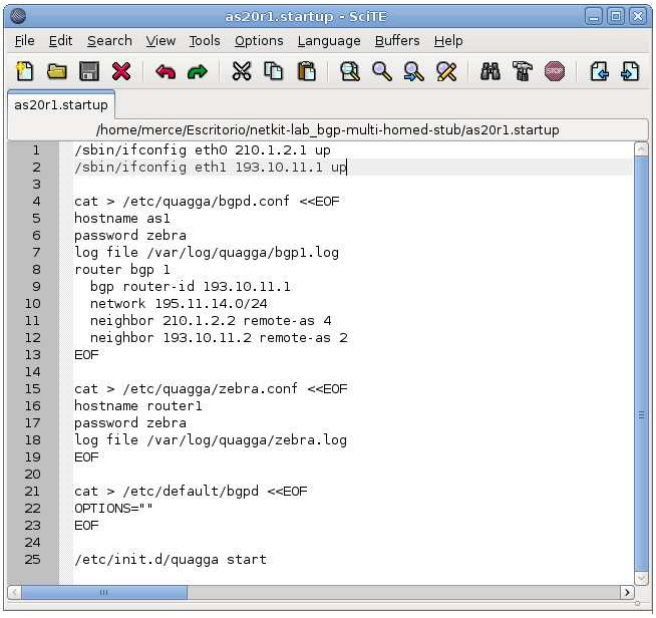



Figura 6.6: Archivo de inicio de un encaminador

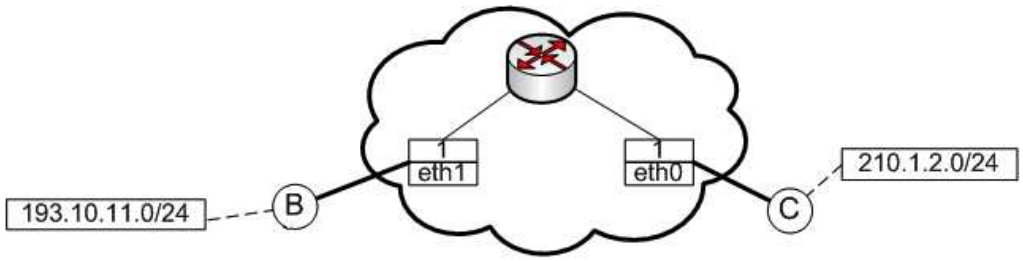



Figura 6.7: Laboratorio multihoming: detalle encaminador

se especifican atributos como `router-id`, `network` o `neighbor`. Para una descripción detallada de los atributos BGP se puede consultar el Apéndice D.

### Demonios *bgpd* en paralelo

Como se puede ver en el archivo de inicio de dos de los encaminadores del laboratorio, Figura 6.8, existen dos configuraciones para *bgpd*, pues se lanzan a ejecución dos demonios *bgpd*. Esto es posible gracias a que con la

opción `-pid-file` se pueden tener dos demonios en paralelo y con la opción `-bgp-port` se puede especificar otro puerto para el segundo demonio sin que haya problemas o interferencias con el primero.

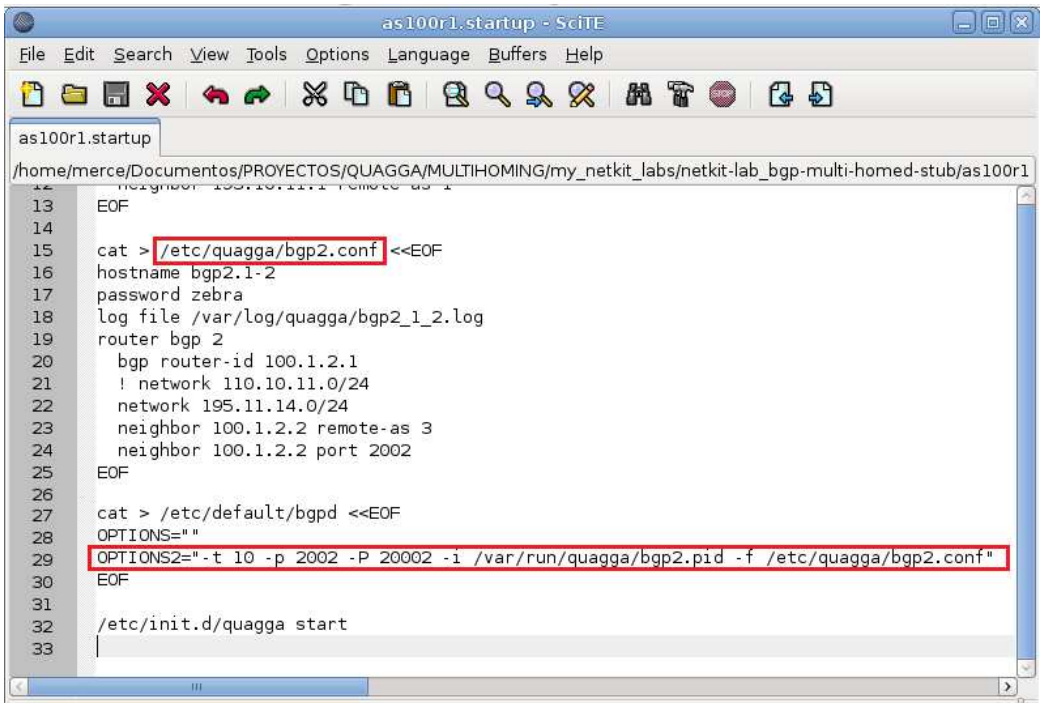



Figura 6.8: Laboratorio multihoming: Archivo de inicio *bgpd* en paralelo

Si nombramos como demonio principal al primero de ellos, y como secundario al restante, la idea subyacente es que el demonio principal se configure con las opciones por defecto, es decir, escritura en la tabla principal del núcleo y escucha en el puerto 179; y el demonio secundario se configure con el segundo de los archivos de configuración, algunos de cuyos parámetros indican que utilice el archivo `bgp2.conf`, el puerto `2002` y la tabla de rutas `10`. Estos parámetros de configuración estań definidos en unos de los scripts de inicio, explicados en la Sección 6.3.2, por lo que no haría falta incluirlos en el archivo de inicio; sin embargo, como se va a implementar otro escenario a parte del actual de multihoming, el cual no lanza dos demonios en paralelo, no se instalarán los scripts para no tener esta configuración de forma permanente.

Esto es representado en la tolopología del laboratorio como dos encami-

nadores rodeados por una línea discontinua, donde el encaminador de color azul representa el demonio *bgpd* secundario, ver Figura 6.9. Así, el demonio que escribe en la tabla de rutas principal implementará un enlace de red primordial y el otro un enlace de *back-up*.

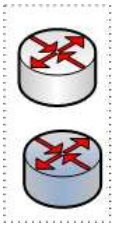


Figura 6.9: Demonios *bgpd* en paralelo

## 6.6. Resultados prácticos

Una vez iniciado el laboratorio, en la Figura 6.10 se puede observar como los encaminadores han aprendido mediante BGP-4 las rutas anunciadas por sus vecinos. En particular, se puede observar como los dos encaminadores de los extremos de la topología, `as100r1` y `as200r1`, han aprendido la ruta anunciada por el otro encaminador a través de los dos demonios *bgpd*, y se han instalado en ambas tablas 0 y 10 del núcleo, consiguiendo así el concepto de multihoming.

Se comprueba ahora en la Figura 6.11 que si se produjese algún problema en el enlace principal, ya sea por problemas de conectividad, de algún interfaz o equipo, se podría seguir teniendo conexión a través del enlace secundario, pues se siguen aprendiendo las rutas ahora únicamente por la tabla 10, que es la que corresponde al enlace de *back-up*. Para emular este comportamiento en el laboratorio, se ha parado Quagga, `/etc/init.d/quagga/stop`, en uno de los encaminadores de tránsito del enlace principal, Figura 6.12.

Por último, si se restaura el enlace principal, en este caso se re-arranca quagga, `/etc/init.d/quagga/restart`, el laboratorio vuelve a la normalidad y los encaminadores aprenden de nuevo las rutas por ambos enlaces.

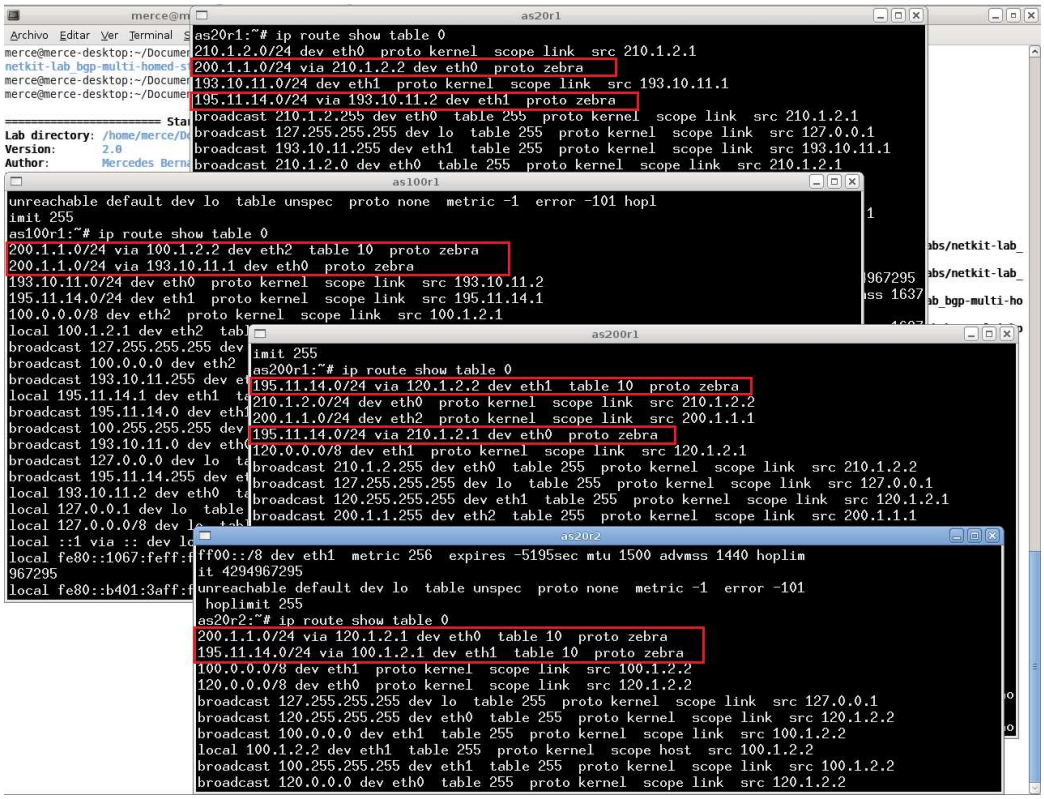



Figura 6.10: Laboratorio multihoming: aprendizaje de rutas por *bgpd* en paralelo

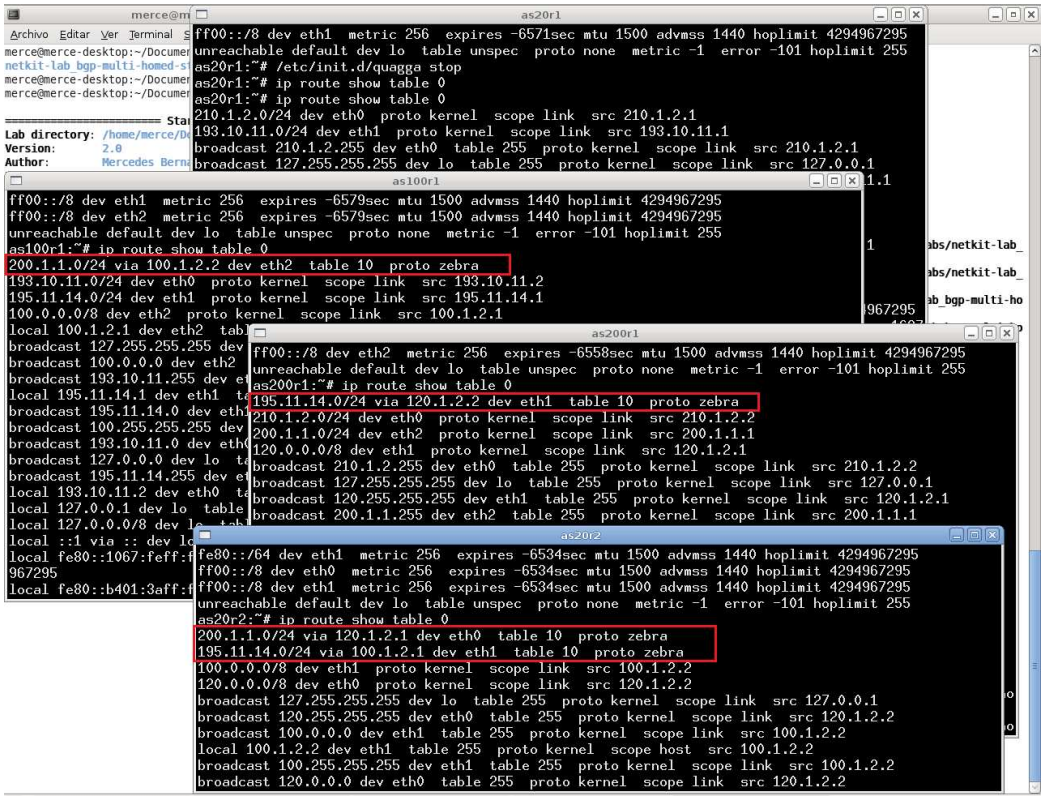



Figura 6.11: Laboratorio multihoming: conectividad a través del enlace secundario

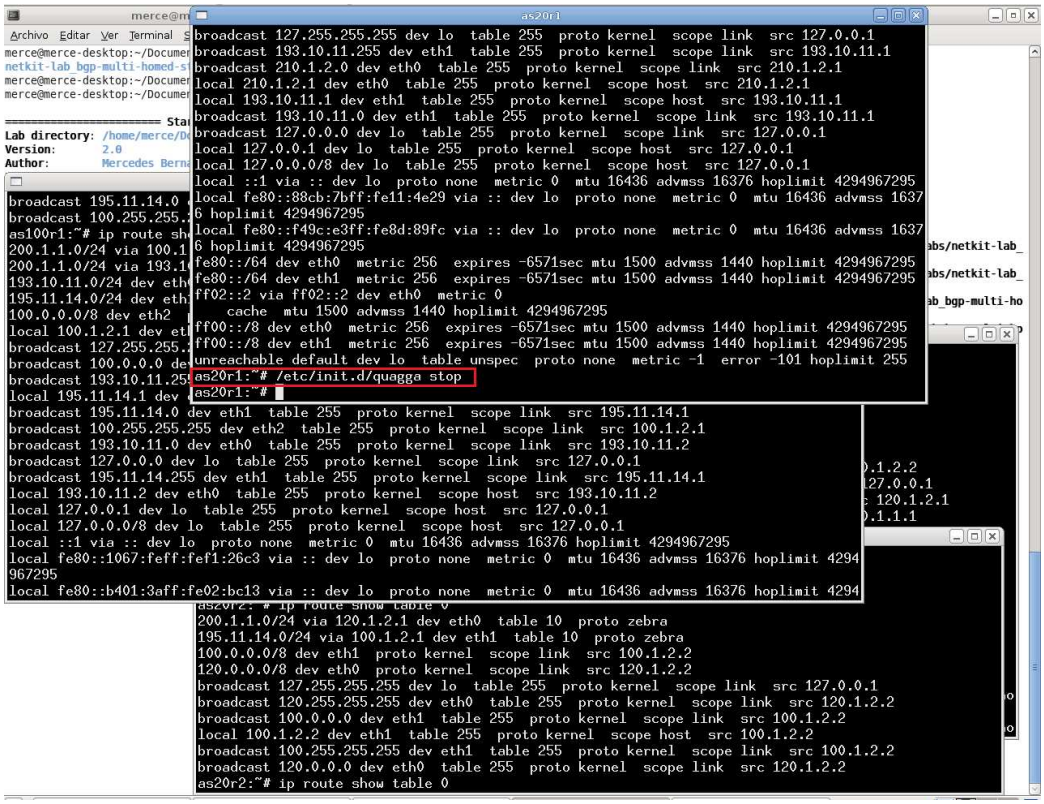



Figura 6.12: Laboratorio multihoming: fallo enlace principal

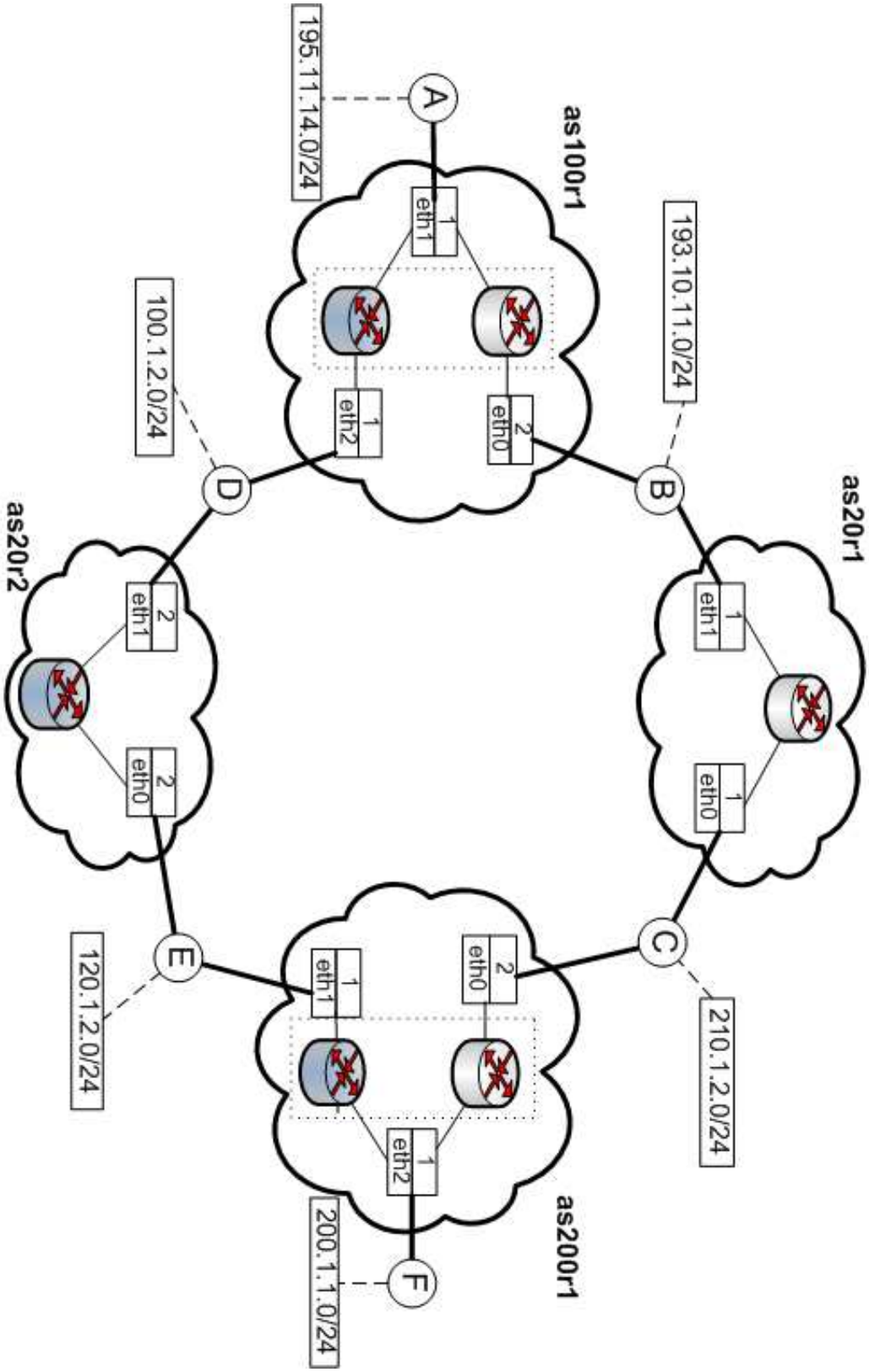



Figura 6.13: Laboratorio multihoming: topología de red

# Capítulo 7

# Conclusiones y trabajo futuro

En primer lugar, decir que se han satisfecho todos los objetivos propuestos, tanto personales como técnicos. Desde el punto de vista personal, se ha adquirido una mejor capacidad de control de proyectos a escala real, se han podido poner en práctica los conocimientos adquiridos durante la carrera y sin lugar a dudas, ha supuesto un gran reto personal.

Desde el punto de vista técnico, por una parte, se ha tenido que estudiar en profundidad el comportamiento del protocolo BGP, sus características, atributos y configuración. Lo que ha conllevado leer multitud de presentaciones, *papers*, artículos, RFCs, e incluso correos con autores de las mismas. Por otro lado, se ha tenido que investigar profundamente en la estructura del software de encaminamiento Quagga para poder modificarla. Esto ha supuesto un gran esfuerzo, no sólo por la falta de conocimiento o experiencia en su utilización, sino por la gran cantidad de directorios y archivos de código por los que está compuesto, y la nula documentación acerca de su estructura interna. A su vez, se ha tenido que adquirir destreza en el manejo del software de emulación Netkit, pues muchas veces las máquinas virtuales no funcionaban como se deseaba o quedaban corruptas para siempre. Por último, se ha aprendido también el entorno LaTeX y se han conseguido los resultados esperados con la memoria.

Como trabajo futuro, creo que la propia palabra BGP lo describe. Existen multitud de campos en los que se puede investigar, ingeniería de tráfico, nuevas adaptaciones del protocolo al inminente direccionamiento IPv6, seguridad, ataques, estudio y observación del crecimiento de las tablas de encaminamiento y posibles soluciones de ingeniería que eviten un incremento exponencial, entre otras.

# Apéndice A

# Instalación de Netkit

Se trata de una instalación bastante sencilla. Primero se han de descargar tres ficheros:

- El programa principal.
- El sistema de ficheros.
- El kernel que usarán las máquinas virtuales.

Una vez descargados deben ser descomprimidos todos juntos en el directorio que se elija (por ejemplo, `/usr/share/netkit`) y luego se han de configurar unas cuantas variables de entorno para que quede constancia de donde se ha instalado Netkit. Para ello, lo más recomendable es añadir al final del archivo `/etc/profile` las líneas que aparecen en la Figura A.1.

Una vez configuradas las variables de entorno, por último, es recomentable comprobar que no ha habido ningún fallo. Para ello basta ejecutar un script de comprobación que incluye Netkit, `check_configuration.sh`, cuya salida podemos ver en la Figura A.2.

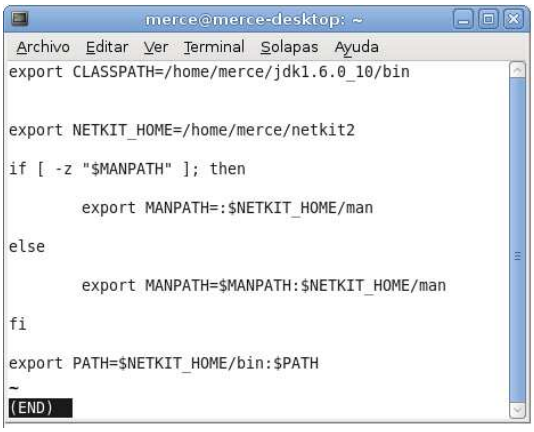



Figura A.1: Variables de entorno

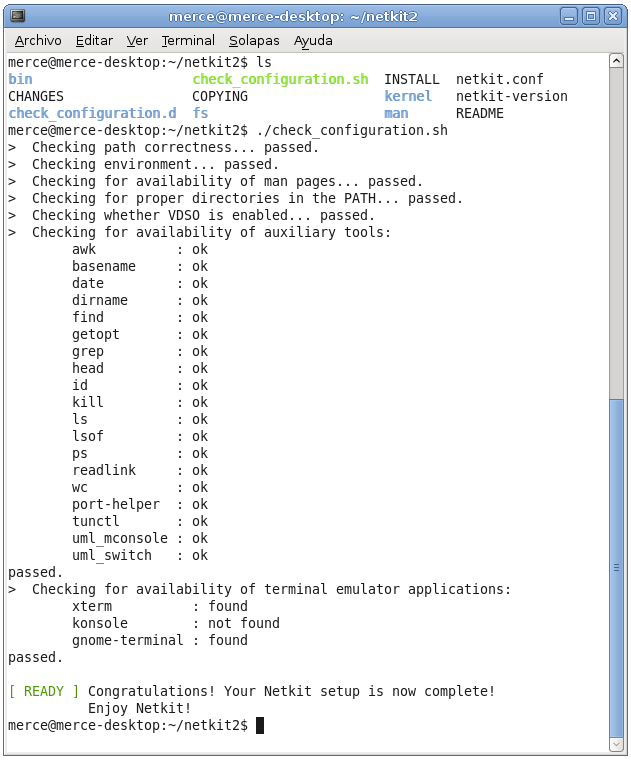



Figura A.2: Script de comprobación de instalación de Netkit

# Apéndice B

# Principales comandos en Netkit

Una de las características más interesantes de Netkit es la interfaz de usuario, que incluye herramientas para la creación rápida y sencilla de escenarios de red complejos. Las máquinas virtuales de Netkit pueden ser gestionadas a través de un conjunto de comandos *ltools* y *vtools* que se deben ejecutar en la máquina y se implementan como *shell scripts*.

Las herramientas **vtools** están concebidas para configurar y gestionar máquinas virtuales individuales, ofreciendo las siguientes funcionalidades:

`vstart`

Puede ser usada para configurar y crear una nueva máquina virtual, identificada con un nombre aleatorio. Permite configurar parámetros como la cantidad de memoria disponible, el núcleo y sistema de archivos que se utilizará, así como las interfaces de red y los dominios de colisión. Los ajustes por defecto de Netkit suelen adaptarse a la mayoría de las necesidades, de modo que arrancar un dispositivo de red virtual a menudo simplemente consiste en especificar las interfaces de red.

`vconfig`

Se puede utilizar para conectar "sobre la marcha" una interfaz de red a una máquina virtual en ejecución. Esto es útil para modificar la configuración de un escenario que ya se está ejecutando o simplemente para evitar tener que reiniciar una máquina, porque una de sus interfaces se ha olvidado.

`vlist`

Proporciona información sobre las máquinas virtuales que se están ejecutando en ese momento. Si se invoca sin argumentos, produce una lista de máquinas virtuales incluido el usuario que las ha comenzado, el PID del hilo, la cantidad de memoria que consume, y la lista de las interfaces de red con los dominios de colisión. Si se proporciona como argumento un nombre de máquina virtual, proporciona información detallada al respecto.

`vhalt`

Es exactamente el equivalente a ejecutar el comando `halt` dentro de una máquina virtual. Detiene "con cortesía" la máquina mediante sus scripts de apagado, desmontando correctamente el sistema de archivos y parándolo.

`vcrash`

Proporciona un método para detener de inmediato una máquina virtual en ejecución. En realidad, es equivalente a la desconexión del cable de corriente bruscamente. Esto se logra, primero, solicitando pararse al kernel UML mediante un *socket* especial de gestión. Para este propósito se hace uso de la utilidad UML [71] `uml_mconsole`. En caso de que este intento falle, los procesos de la máquina virtual son automáticamente matados por `vcrash`. En el inicio posterior, una máquina rota realizará una verificación del sistema de ficheros para recuperar inconsistencias. Este comando se utiliza a menudo en los experimentos de red que consisten en varias máquinas, ya que es mucho más rápido que el comando `vhalt`.

`vclean`

Es el "botón de pánico" de Netkit. En caso de desastre, una máquina virtual colgada o configuraciones de túneles olvidadas en la máquina, `vclean` ayuda a deshacerse de todo en un sólo comando. `vclean` también es muy útil cuando se utiliza en combinación con el comando `vconfig`.

Netkit proporciona un nivel mayor, las herramientas **ltools**, que permiten fácilmente configurar, lanzar o parar un escenario complejo de una manera directa. El prefijo l significa "laboratorio", que es el nombre que se asocia a menudo en Netkit a los escenarios preconfigurados redistribuibles. Las utilidades `ltools` se basan en las funcionalidades proporcionadas por las herramientas `vtools` y ofrecen la siguiente interfaz:

`lstart`

Se utiliza para iniciar las máquinas virtuales que conforman un laboratorio. Opcionalmente, se puede utilizar para arrancar sólo un subconjunto de ellas.

`ltest`

Soporta la creación de laboratorios de pruebas redistribuibles. Básicamente, el principio detrás de este comando es que cada máquina virtual está automáticamente ordenada para ejecutar unas descargas de información significante definidas por el usuario. Por ejemplo, pueden incluir el contenido de una tabla de encaminamiento o los resultados de un `ping`. Esta información se puede recoger y guardar como una firma de una red emulada correctamente. Una vez que el laboratorio se mueve a otra máquina diferente, verificar que aún se está ejecutando correctamente es simplemente cuestión de arrancarlo en modo de prueba y verificar que la información obtenida coincide con la firma.

`lhalt` y `lcrash`

Se comportan como sus homólogos `vhalt` y `vcrash`, pero éstos ejecutan de forma automática la operación en todas las máquinas virtuales que conforman un laboratorio. Opcionalmente, pueden afectar sólo a un subconjunto de las máquinas del laboratorio, por ejemplo, en caso de que algunas de ellas necesiten ser reiniciadas sin tener que reiniciar todo el laboratorio.

`linfo`

Se puede utilizar para obtener información básica acerca de un laboratorio, incluidos los datos descriptivos y una lista de las máquinas virtuales que lo componen. También se puede utilizar para hacer un dibujo de la topología de enlace de datos de un laboratorio, incluidos las máquinas, interfaces, y dominios de colisión. Para esta última función requiere que la librería de dibujo de gráficos Graphviz [2] esté correctamente instalada.

`lclean`

Sólo realiza una limpieza de los archivos temporales que quedan después de ejecutar un laboratorio. No tiene nada que ver con su homólogo `vclean`.

Ambas herramientas `vtools` y `ltools`, así como otros componentes de Netkit, están ampliamente documentados en las páginas `man` disponibles con la distribución Netkit.

# Apéndice C

# Instalación de Quagga

Se han de seguir los pasos de configuración, compilación e instalación en el orden indicado que figura en los siguientes apartados.

## C.1. Configuración

Quagga tiene un excelente *script* de configuración que detecta automáticamente la mayoría de las configuraciones. Hay diferentes opciones de configuración, por ejemplo, para desactivar el soporte de IPv6, para desactivar la compilación de demonios específicos, para habilitar el soporte de Simple Network Management Protocol (SNMP), etc. Una vez configurado las opciones disponibles basta escribir en el terminal la instrucción `./configure`.

## C.2. Compilación

Después de configurar el software, hay que compilarlo para el sistema. Simplemente es cuestión de ejecutar el comando `make`en la raíz del directorio de origen y el software será compilado.

## C.3. Instalación

Instalar el software consiste en copiar los programas compilados y el soporte a archivos a una ubicación estándar, a través del comando `make install`.

Después de que el proceso de instalación ha finalizado, estos archivos se han copiado del directorio de trabajo a `/usr/local/bin` y `/usr/local/etc`.

Suele ser necesario realizar cambios en los archivos de configuración en `/etc/quagga/*.conf` como se explica en la Sección 4.2.1 y Sección 4.2.2.

# Apéndice D

# Comandos BGP

Ver [21] para información más detallada de algún comando o síntaxis específica.

`address-family ipv4 (BGP)`
To enter address family or router scope address family configuration mode to configure a routing session using standard IP Version 4 address prefixes.

`address-family l2vpn`
To enter address family configuration mode to configure a routing session using Layer 2 Virtual Private Network (L2VPN) endpoint provisioning address information.

`address-family nsap`
To enter address family configuration mode to configure Connectionless Network Service (CLNS)-specific parameters for **BGP!** (**BGP!**) routing sessions.

`address-family vpnv4`
To enter address family configuration mode to configure a routing session using Virtual Private Network (VPN) Version 4 address prefixes.

`aggregate-address`
To create an aggregate entry in a Border Gateway Protocol (BGP) database.

`auto-summary (BGP)`

To configure automatic summarization of subnet routes into network-level routes.

`bgp additional-paths install`

To enable BGP to calculate a backup path for a given address family and to install it into the Routing Information Base (RIB) and Cisco Express Forwarding.

`bgp advertise-best-external`

To enable BGP to calculate an external route as the best backup path for a given address family and to install it into the Routing Information base (RIB) and Cisco Express Forwarding, and to advertise the best external path to its neighbors.

`bgp aggregate-timer`

To set the interval at which BGP routes will be aggregated or to disable timer-based route aggregation.

`bgp always-compare-med`

To enable the comparison of the Multi Exit Discriminator (MED) for paths from neighbors in different autonomous systems.

`bgp asnotation dot`

To change the default display and regular expression match format of Border Gateway Protocol (BGP) 4-byte autonomous system numbers from asplain (decimal values) to dot notation.

`bgp bestpath as-path ignore`

To configure Border Gateway Protocol (BGP) to not consider the autonomous system (AS) path during best path route selection.

`bgp bestpath compare-routerid`

To configure a Border Gateway Protocol (BGP) routing process to compare identical routes received from different external peers during the best path selection process and to select the route with the lowest router ID as the best path.

`bgp bestpath cost-community ignore`

To configure a router that is running the Border Gateway Protocol (BGP) to not evaluate the cost community attribute during the best path selection process.

`bgp bestpath med confed`

To configure a Border Gateway Protocol (BGP) routing process to compare the Multi Exit Discriminator (MED) between paths learned from confederation peers.

`bgp bestpath med missing-as-worst`

To configure a Border Gateway Protocol (BGP) routing process to assign a value of infinity to routes that are missing the Multi Exit Discriminator (MED) attribute (making the path without a MED value the least desirable path).

`bgp client-to-client reflection`

To enable or restore route reflection from a BGP route reflector to clients.

`bgp cluster-id`

To set the cluster ID on a route reflector in a route reflector cluster.

`bgp confederation identifier`

To specify a BGP confederation identifier.

`bgp confederation peers`

To configure subautonomous systems to belong to a single confederation.

`bgp dampening`

To enable BGP route dampening or change BGP route dampening parameters.

`bgp default ipv4-unicast`

To set the IP version 4 (IPv4) unicast address family as default for BGP peering session establishment.

`bgp default local-preference`

To change the default local preference value.

`bgp deterministic-med`

To enforce the deterministic comparison of the Multi Exit Discriminator (MED) value between all paths received from within the same autonomous system.

`bgp dmzlink-bw`

To configure BGP to distribute traffic proportionally over external links with unequal bandwidth when multipath load balancing is enabled.

`bgp enforce-first-as`

To configure a router to deny an update received from an external BGP (eBGP) peer that does not list its autonomous system number at the beginning of the AS_PATH in the incoming update.

`bgp fast-external-fallover`

To configure a Border Gateway Protocol (BGP) routing process to immediately reset external BGP peering sessions if the link used to reach these peers goes down.

`bgp graceful-restart`

To enable the Border Gateway Protocol (BGP) graceful restart capability globally for all BGP neighbors.

`bgp inject-map`

To configure conditional route injection to inject more specific routes into a Border Gateway Protocol (BGP) routing table.

`bgp listen`

To associate a subnet range with a Border Gateway Protocol (BGP) peer group and activate the BGP dynamic neighbors feature.

`bgp log-neighbor-changes`

To enable logging of BGP neighbor resets.

`bgp maxas-limit`

To configure Border Gateway Protocol (BGP) to discard routes that have a number of AS-path segments that exceed the specified value.

`bgp nexthop trigger delay`

The trigger and delay keywords for the bgp nexthop command are no longer documented as a separate command. See the bgp nexthop command documentation for more information.

`bgp nexthop trigger enable`

The trigger and enable keywords for the bgp nexthop command are no longer documented as a separate command. See the bgp nexthop command documentation for more information.

`bgp nexthop`

To configure Border Gateway Protocol (BGP) next-hop address tracking.

`bgp recursion host`

To enable the recursive-via-host flag for IP Version 4 (IPv4), Virtual Private Network (VPN) Version 4 (VPNv4), and Virtual Routing and Forwarding (VRF) address families.

`bgp redistribute-internal`

To configure iBGP redistribution into an interior gateway protocol (IGP), such as IS-IS or OSPF.

`bgp regexp deterministic`

To configure Cisco IOS software to use the deterministic processing time regular expression engine.

`bgp router-id`

To configure a fixed router ID for the local Border Gateway Protocol (BGP) routing process.

`bgp rr-group`

To create a route-reflector group and enable automatic inbound filtering for VPN version 4 (VPNv4) updates based on the allowed route target (RT) extended communities.

`bgp soft-reconfig-backup`

To configure a Border Gateway Protocol (BGP) speaker to perform inbound soft reconfiguration for peers that do not support the route refresh capability.

`bgp suppress-inactive`

To suppress the advertisement of routes that are not installed in the routing information base (RIB).

`bgp transport`

To enable TCP transport session parameters globally for all Border Gateway Protocol (BGP) sessions.

`bgp update-delay`

To set the maximum initial delay period before a Border Gateway Protocol (BGP)-speaking networking device sends its first updates.

`bgp update-group split as-override`

To keep peers that are configured with neighbor as-override in separate, single-member update groups.

`bgp upgrade-cli`

To upgrade a Network Layer Reachability Information (NLRI) formatted router configuration file to the address-family identifier (AFI) format and set the router command-line interface (CLI) to use only AFI commands.

`bgp-policy`

To enable Border Gateway Protocol (BGP) policy accounting or policy propagation on an interface.

`clear bgp nsap`

To clear and then reset Connectionless Network Service (CLNS) network service access point (NSAP) Border Gateway Protocol (BGP) sessions.

`clear bgp nsap flap-statistics`

To clear Border Gateway Protocol (BGP) flap statistics for the network service access point (NSAP) address family.

`clear bgp nsap peer-group`

To clear the Border Gateway Protocol (BGP) TCP connections to all members of a BGP peer group for the network service access point (NSAP) address family.

`clear ip bgp`

To reset Border Gateway Protocol (BGP) connections using hard or soft reconfiguration.

`clear ip bgp dampening`

To clear BGP route dampening information and to unsuppress suppressed routes.

`clear ip bgp external`

To reset external Border Gateway Protocol (eBGP) peering sessions using hard or soft reconfiguration.

`clear ip bgp flap-statistics`

To clear BGP route dampening flap statistics.

`clear ip bgp in prefix-filter`

The in and prefix-filter keywords for the clear ip bgp command are no longer documented as a separate command. The information for using the in and prefix-filter keywords with the clear ip bgp command has been incorporated into all the appropriate clear ip bgp command documentation.

`clear ip bgp ipv4`

To reset Border Gateway Protocol (BGP) connections using hard or soft reconfiguration for IPv4 address family sessions.

`clear ip bgp ipv6`

To reset Border Gateway Protocol (BGP) connections using hard or soft reconfiguration for IPv6 address family sessions.

`clear ip bgp l2vpn`

To reset Border Gateway Protocol (BGP) neighbor session information for Layer 2 Virtual Private Network (L2VPN) address family.

`clear ip bgp peer-group`
To reset Border Gateway Protocol (BGP) connections using hard or soft reconfiguration for all the members of a BGP peer group.

`clear ip bgp table-map`
To refresh table-map configuration information in the Border Gateway Protocol (BGP) routing table.

`clear ip bgp update-group`
To reset Border Gateway Protocol (BGP) connections for all the members of a BGP update group.

`clear ip bgp vpnv4`
To reset Border Gateway Protocol (BGP) connections using hard or soft reconfiguration for IPv4 Virtual Private Network (VPNv4) address family sessions.

`clear ip bgp vpnv6`
To reset Border Gateway Protocol (BGP) connections using hard or soft reconfiguration for IPv6 Virtual Private Network (VPNv6) address family sessions.

`clear ip prefix-list`
To reset IP prefix-list counters.

`continue`
To configure a route map to go to a route-map entry with a higher sequence number.

`default-information originate (BGP)`
To configure a Border Gateway Protocol (BGP) routing process to distribute a default route (network 0.0.0.0).

`default-metric (BGP)`
To set a default metric for routes redistributed into Border Gateway Protocol (BGP).

`distance bgp`

To configure the administrative distance for BGP routes.

`distribute-list in (BGP)`
To filter routes or networks received in incoming Border Gateway Protocol (BGP) updates.

`distribute-list out (BGP)`
To suppress networks from being advertised in outbound Border Gateway Protocol (BGP) updates.

`exit-peer-policy`
To exit policy-template configuration mode and enter router configuration mode.

`exit-peer-session`
To exit session-template configuration mode and enter router configuration mode.

`export map`
To associate an export map with a VPN Routing and Forwarding (VRF) instance.

`ha-mode graceful-restart`
To enable or disable the Border Gateway Protocol (BGP) graceful restart capability for a BGP peer session template.

`import ipv4`
To configure an import map to import IPv4 prefixes from the global routing table to a VRF table.

`import path limit`
To specify the maximum number of Border Gateway Protocol (BGP) paths, per VPN routing and forwarding (VRF) importing net, that can be imported from an exporting net.

`import path selection`
To specify the Border Gateway Protocol (BGP) import path selection policy for a specific VPN routing and forwarding (VRF) instance.

`inherit peer-policy`
To configure a peer policy template to inherit the configuration from another peer policy template.

`inherit peer-session`
To configure a peer session template to inherit the configuration from another peer session template.

`ip as-path access-list`
To configure an autonomous system path filter using a regular expression.

`ip bgp fast-external-fallover`
To configure per-interface fast external fallover.

`ip bgp-community new-format`
To configure BGP to display communities in the format AA:NN (autonomous system:community number/4-byte number).

`ip community-list`
To create or configure a Border Gateway Protocol (BGP) community list and to control access to it.

`ip extcommunity-list`
To create an extended community list to configure Virtual Private Network (VPN) route filtering.

`ip policy-list`
To create a Border Gateway Protocol (BGP) policy list.

`ip prefix-list`
To create a prefix list or to add a prefix-list entry.

`ip prefix-list description`
To add a text description of a prefix list.

`ip prefix-list sequence-number`

To enable the generation of default sequence numbers for entries in a prefix list.

`ip verify unicast vrf`
To enable Unicast Reverse Path Forwarding (Unicast RPF) verification for a specified VRF.

`match source-protocol`
To match Enhanced Interior Gateway Routing Protocol (EIGRP) external routes based on a source protocol and autonomous system number.

`maximum-paths ibgp`
To control the maximum number of parallel internal Border Gateway Protocol (iBGP) routes that can be installed in a routing table.

`neighbor activate`
To enable the exchange of information with a Border Gateway Protocol (BGP) neighbor.

`neighbor advertise-map`
To install a Border Gateway Protocol (BGP) route as a locally originated route in the BGP routing table for conditional advertisement.

`neighbor advertisement-interval`
To set the minimum route advertisement interval (MRAI) between the sending of BGP routing updates.

`neighbor capability orf prefix-list`
To advertise outbound route filter (ORF) capabilities to a peer router.

`neighbor default-originate`
To allow a BGP speaker (the local router) to send the default route 0.0.0.0 to a neighbor for use as a default route.

`neighbor description`
To associate a description with a neighbor.

`neighbor disable-connected-check`

To disable connection verification to establish an eBGP peering session with a single-hop peer that uses a loopback interface.

`neighbor distribute-list`
To distribute BGP neighbor information as specified in an access list.

`neighbor dmzlink-bw`
To configure Border Gateway Protocol (BGP) to advertise the bandwidth of links that are used to exit an autonomous system.

`neighbor ebgp-multihop`
To accept and attempt BGP connections to external peers residing on networks that are not directly connected.

`neighbor fall-over`
To enable Border Gateway Protocol (BGP) to monitor the peering session of a specified neighbor for adjacency changes and to deactivate the peering session.

`neighbor filter-list`
To set up a BGP filter.

`neighbor ha-mode graceful-restart`
To enable or disable the Border Gateway Protocol (BGP) graceful restart capability for a BGP neighbor or peer group.

`neighbor inherit peer-policy`
To send a peer policy template to a neighbor so that the neighbor can inherit the configuration.

`neighbor inherit peer-session`
To send a peer session template to a neighbor so that the neighbor can inherit the configuration.

`neighbor local-as`
To customize the AS_PATH attribute for routes received from an external Border Gateway Protocol (eBGP) neighbor.

`neighbor maximum-prefix`
To control how many prefixes can be received from a neighbor.

`neighbor maximum-prefix (BGP)`
To control how many prefixes can be received from a neighbor.

`neighbor next-hop-self`
To configure the router as the next hop for a BGP-speaking neighbor or peer group.

`neighbor next-hop-unchanged`
To enable an external BGP (eBGP) multihop peer to propagate the next hop unchanged.

`neighbor password`
To enable Message Digest 5 (MD5) authentication on a TCP connection between two BGP peers.

`neighbor peer-group (assigning members)`
To configure a BGP neighbor to be a member of a peer group.

`neighbor peer-group (creating)`
To create a BGP or multiprotocol BGP peer group.

`neighbor prefix-list`
To prevent distribution of Border Gateway Protocol (BGP) neighbor information as specified in a prefix list, a Connectionless Network Service (CLNS) filter expression, or a CLNS filter set.

`neighbor remote-as`
To add an entry to the BGP or multiprotocol BGP neighbor table.

`neighbor remove-private-as`
To remove private autonomous system numbers from t in outbound routing updates.

`neighbor route-map`

To apply a route map to incoming or outgoing routes.

`neighbor route-reflector-client`
To configure the router as a BGP route reflector and configure the specified neighbor as its client.

`neighbor send-community`
To specify that a communities attribute should be sent to a BGP neighbor.

`neighbor shutdown`
To disable a neighbor or peer group.

`neighbor soft-reconfiguration`
To configure the Cisco IOS software to start storing updates.

`neighbor soo`
To set the site-of-origin (SoO) value for a Border Gateway Protocol (BGP) neighbor or peer group.

`neighbor timers`
To set the timers for a specific BGP peer or peer group.

`neighbor transport`
To enable a TCP transport session option for a Border Gateway Protocol (BGP) session.

`neighbor ttl-security`
To secure a Border Gateway Protocol (BGP) peering session and to configure the maximum number of hops that separate two external BGP (eBGP) peers.

`neighbor unsuppress-map`
To selectively advertise routes previously suppressed by the aggregate-address command.

`neighbor update-source`
To have the Cisco IOS software allow Border Gateway Protocol (BGP) sessions to use any operational interface for TCP connections.

`neighbor version`
To configure the Cisco IOS software to accept only a particular BGP version.

`neighbor weight`
To assign a weight to a neighbor connection.

`network (BGP and multiprotocol BGP)`
To specify the networks to be advertised by the Border Gateway Protocol (BGP) and multiprotocol BGP routing processes.

`network backdoor`
To specify a backdoor route to a BGP-learned prefix that provides better information about the network.

`redistribute (BGP to ISO IS-IS)`
To redistribute routes from a Border Gateway Protocol (BGP) autonomous system into an International Organization for Standardization (ISO) Intermediate System-to-Intermediate System (IS-IS) routing process.

`redistribute (ISO IS-IS to BGP)`
To redistribute routes from an International Organization for Standardization (ISO) Intermediate System-to-Intermediate System (IS-IS) routing process into a Border Gateway Protocol (BGP) autonomous system.

`redistribute dvmrp`
To configure redistribution of Distance Vector Multicast Routing Protocol (DVMRP) routes into multiprotocol BGP.

`router bgp`
To configure the Border Gateway Protocol (BGP) routing process.

`scope`
To define the scope for a Border Gateway Protocol (BGP) routing session and to enter router scope configuration mode.

`set as-path`

To modify an autonomous system path for BGP routes.

`set comm-list delete`
To remove communities from the community attribute of an inbound or outbound update.

`set community`
To set the BGP communities attribute.

`set dampening`
To set the BGP route dampening factors.

`set extcommunity`
To set Border Gateway Protocol (BGP) extended community attributes.

`set extcommunity cost`
To create a set clause to apply the cost community attribute to routes that pass through a route map.

`set ip next-hop (BGP)`
To indicate where to output packets that pass a match clause of a route map for policy routing.

`set metric (BGP-OSPF-RIP)`
To set the metric value for a routing protocol.

`set metric-type internal`
To set the Multi Exit Discriminator (MED) value on prefixes advertised to external BGP (eBGP) neighbors to match the Interior Gateway Protocol (IGP) metric of the next hop.

`set origin (BGP)`
To set the BGP origin code.

`set traffic-index`
To indicate how to classify packets that pass a match clause of a route map for Border Gateway Protocol (BGP) policy accounting.

`set weight`
To specify the BGP weight for the routing table.

`show bgp all neighbors`
To display information about Border Gateway Protocol (BGP) connections to neighbors of all address families.

`show bgp nsap`
To display entries in the Border Gateway Protocol (BGP) routing table for the network service access point (NSAP) address family.

`show bgp nsap community`
To display routes that belong to specified network service access point (NSAP) Border Gateway Protocol (BGP) communities.

`show bgp nsap community-list`
To display routes that are permitted by the Border Gateway Protocol (BGP) community list for network service access point (NSAP) prefixes.

`show bgp nsap dampened-paths`
Effective with Cisco IOS Release 12.2(33)SRB, the show bgp nsap dampened-paths command is replaced by the show bgp nsap dampening command. See the show bgp nsap dampening command for more information. To display network service access point (NSAP) address family Border Gateway Protocol (BGP) dampened routes in the BGP routing table.

`show bgp nsap dampening`
To display network service access point (NSAP) address family Border Gateway Protocol (BGP) dampened routes in the BGP routing table.

`show bgp nsap filter-list`
To display routes in the Border Gateway Protocol (BGP) routing table for the network service access point (NSAP) address family that conform to a specified filter list.

`show bgp nsap flap-statistics`
To display Border Gateway Protocol (BGP) flap statistics for network service access point (NSAP) prefixes.

`show bgp nsap inconsistent-as`
To display Border Gateway Protocol (BGP) network service access point (NSAP) prefix routes with inconsistent originating autonomous systems.

`show bgp nsap neighbors`
To display information about Border Gateway Protocol (BGP) network service access point (NSAP) prefix connections to neighbors.

`show bgp nsap paths`
To display all the Border Gateway Protocol (BGP) network service access point (NSAP) prefix paths in the database.

`show bgp nsap quote-regexp`
To display Border Gateway Protocol (BGP) network service access point (NSAP) prefix routes matching the AS-path regular expression as a quoted string of characters.

`show bgp nsap regexp`
To display Border Gateway Protocol (BGP) network service access point (NSAP) prefix routes matching the AS-path regular expression.

`show bgp nsap summary`
To display the status of all Border Gateway Protocol (BGP) network service access point (NSAP) prefix connections.

`show ip bgp`
To display entries in the Border Gateway Protocol (BGP) routing table.

`show ip bgp cidr-only`
To display routes with classless interdomain routing (CIDR).

`show ip bgp community`
To display routes that belong to specified BGP communities.

`show ip bgp community-list`
To display routes that are permitted by the Border Gateway Protocol (BGP) community list.

`show ip bgp dampened-paths`
To display BGP dampened routes.

`show ip bgp dampening dampened-paths`
To display Border Gateway Protocol (BGP) dampened routes on the Cisco 10000 series router.

`show ip bgp dampening flap-statistics`
To display Border Gateway Protocol (BGP) flap statistics for all paths on the Cisco 10000 series router.

`show ip bgp dampening parameters`
To display detailed Border Gateway Protocol (BGP) dampening information on the Cisco 10000 series router.

`show ip bgp filter-list`
To display routes that conform to a specified filter list.

`show ip bgp flap-statistics`
To display BGP flap statistics.

`show ip bgp inconsistent-as`
To display routes with inconsistent originating autonomous systems.

`show ip bgp injected-paths`
To display all the injected paths in the Border Gateway Protocol (BGP) routing table.

`show ip bgp ipv4`
To display entries in the IP version 4 (IPv4) Border Gateway Protocol (BGP) routing table.

`show ip bgp ipv4 multicast`
To display IP Version 4 multicast database-related information.

`show ip bgp ipv4 multicast summary`

To display a summary of IP Version 4 multicast database-related information.

`show ip bgp l2vpn`

To display Layer 2 Virtual Private Network (L2VPN) address family information from the Border Gateway Protocol (BGP) table.

`show ip bgp neighbors`

To display information about Border Gateway Protocol (BGP) and TCP connections to neighbors.

`show ip bgp paths`

To display all the BGP paths in the database.

`show ip bgp peer-group`

To display information about BGP peer groups.

`show ip bgp quote-regexp`

To display routes matching the autonomous system path regular expression.

`show ip bgp regexp`

To display routes matching the autonomous system path regular expression.

`show ip bgp replication`

To display update replication statistics for Border Gateway Protocol (BGP) update groups.

`show ip bgp rib-failure`

To display Border Gateway Protocol (BGP) routes that failed to install in the Routing Information Base (RIB) table.

`show ip bgp summary`

To display the status of all Border Gateway Protocol (BGP) connections.

`show ip bgp template peer-policy`

To display locally configured peer policy templates.

`show ip bgp template peer-session`
To display peer policy template configurations.

`show ip bgp update-group`
To display information about BGP update groups.

`show ip bgp vpnv4`
To display Virtual Private Network Version 4 (VPNv4) address information from the Border Gateway Protocol (BGP) table.

`show ip community-list`
To display configured community lists.

`show ip extcommunity-list`
To display routes that are permitted by an extended community list.

`show ip policy-list`
To display information about a configured policy list and policy list entries.

`show ip prefix-list`
To display information about a prefix list or prefix list entries.

`soo`
To set the site-of-origin (SoO) value for a Border Gateway Protocol (BGP) peer policy template.

`synchronization`
To enable the synchronization between BGP and your Interior Gateway Protocol (IGP) system.

`table-map`
To modify metric and tag values when the IP routing table is updated with BGP learned routes.

`template peer-policy`

To create a peer policy template and enter policy-template configuration mode.

`template peer-session`

To create a peer session template and enter session-template configuration mode.

`timers bgp`

To adjust BGP network timers.

# Apéndice E

# Modificación Quagga BGP-Wedgies

```
diff -rup quagga/bgpd/bgpd.h quagga_new/bgpd/bgpd.h
--- quagga/bgpd/bgpd.h 2007-11-01 15:29:11.000000000 +0100
+++ quagga_new/bgpd/bgpd.h 2008-01-10 11:57:30.000000000 +0100
@@ -934,4 +934,6 @@ extern int peer_clear_soft (struct peer

 extern void peer_nsf_stop (struct peer *);

+int extcheck(char *filename,char *ext);
+
 #endif /* _QUAGGA_BGPD_H */
diff -rup quagga/bgpd/bgp_dump.c quagga_new/bgpd/bgp_dump.c
--- quagga/bgpd/bgp_dump.c 2007-10-15 00:32:21.000000000 +0200
+++ quagga_new/bgpd/bgp_dump.c 2008-01-17 14:52:35.000000000
+0100
@@ -21,7 +21,27 @@ Software Foundation, Inc., 59 Temple Pla
 #include <zebra.h>

 #include "log.h"
+
+#ifndef ORIG
+#include <string.h>
+#ifdef _GNU_SOURCE
+#undef _GNU_SOURCE
```

```
+#include <libgen.h>
+#define _GNU_SOURCE
+#else
+#include <libgen.h>
+#endif
+
+// this is for dir creation
+# include <sys/stat.h>
+# include <sys/types.h>
+
+# define DIR_PERMS  S_IRWXU | S_IRGRP | S_IXGRP /* 0750 */
+#endif
+
 #include "stream.h"
+#include "zlib.h"
+
 #include "sockunion.h"
 #include "command.h"
 #include "prefix.h"
@@ -33,7 +53,17 @@ Software Foundation, Inc., 59 Temple Pla
 #include "bgpd/bgp_route.h"
 #include "bgpd/bgp_attr.h"
 #include "bgpd/bgp_dump.h"
+
+#ifdef  WITH_DIR_LEVEL
+void zlog_debug_failure(int level,const char *fname);
+int mkzebradirs(int level,char *path)
+#else
+void zlog_debug_failure(const char *fname);
+int mkzebradirs(char *path);
+#endif
+
+#define GZEXT ".gz"
+
 enum bgp_dump_type
 {
   BGP_DUMP_ALL,
@@ -68,11 +98,14 @@ struct bgp_dump
```

```
   FILE *fp;

+
+gzFile *gfp;
   unsigned int interval;

   char *interval_str;

   struct thread *t_interval;
+
 };

 /* BGP packet dump output buffer. */
@@ -89,9 +122,130 @@ struct bgp_dump bgp_dump_routes;

 /* Dump whole BGP table is very heavy process.  */
 struct thread *t_bgp_dump_routes;
+
+#define GZFLUSH_MODE 4
+
+#ifndef ORIG
+/*
+ * Auxiliary  functions to automate the creation the
+ * directory structure  needed to store  the rib and
+ * update dumps
+ *
+ * These  functions  are always executed by external
+ * scripts in the route collector.  Unifying them in
+ * bgpd will reduce the route collector installation
+ * overhead and the possibility of install/config
+ * errors
+ */
+
+// define WITH_DIR_LEVEL mainly for debugging purposes
+#ifdef  WITH_DIR_LEVEL
+void zlog_debug_failure(int level,const char *fname)
+#else
```

```
+void zlog_debug_failure(const char *fname)
+#endif
+{
+  const char *cause;
+
+  switch (errno)
+    {
+    case EACCES: cause = "EACCES"; break;
+    case EEXIST: cause = "EEXIST"; break;
+    case ELOOP: cause = "ELOOP"; break;
+    case EMLINK: cause = "EMLINK"; break;
+    case ENAMETOOLONG: cause = "ENAMETOOLONG"; break;
+    case ENOENT: cause = "ENOENT"; break;
+    case ENOSPC: cause = "ENOSPC"; break;
+    case ENOTDIR: cause = "ENOTDIR"; break;
+    case EROFS : cause = "EROFS "; break;
+    default: cause = "UNKNOWN"; break;
+    }
+#ifdef WITH_DIR_LEVEL
+  zlog_debug("(%2d) %s failure cause is %s",level,fname,cause)
;
+#else
+  zlog_debug("%s failure cause is %s",fname,cause);
+#endif
+}
+
+
+#ifdef WITH_DIR_LEVEL
+int mkzebradirs(int level,char *path)
+#else
+int mkzebradirs(char *path)
+#endif
+{
+  char *path1 = strdup(path);
+  char *dir   = dirname(path1);
+
+  struct stat dirstat;
+
```

```
+  int  result = 0;
+
+#ifdef WITH_DIR_LEVEL
+  zlog_debug("(%2d) dir  = %s",level,dir);
+#else
+  zlog_debug("(mkzebradir) dir  = %s",dir);
+#endif
+
+  if (stat(dir,&dirstat) == -1) {
+    int dirumask;
+
+#ifdef WITH_DIR_LEVEL
+    mkzebradirs(level+1,dir);
+#else
+    mkzebradirs(dir);
+#endif
+    dirumask=umask(0777 & (~DIR_PERMS));
+    result = mkdir(dir,DIR_PERMS);
+    if (result == -1)
+#ifdef WITH_DIR_LEVEL
+      zlog_debug_failure(level,"mkdir");
+#else
+      zlog_debug_failure("mkdir");
+#endif
+    umask(dirumask);
+  }
+  free(path1);
+
+  return result;
+}
+
+#if 0
+int docommand(char *cmd)
+{
+  int childpid;
+  char buffer[2*MAXPATHLEN+2];
+
+  //
```

```
+  // copy to a static buffer before switching to child space
+  //
+  strcpy(buffer,cmd);
+
+ if ((childpid = fork()) != 0)
+    return childpid;
+  zlog_debug("execl(\"/bin/sh\",\"/bin/sh\",\"-c\",\"%s\",
NULL);",buffer);
+  execl("/bin/sh","/bin/sh","-c",buffer,NULL);
+  exit (0);
+}
+#endif
+#endif
+
+
+//file extension checker
+int extcheck(char *filename, char *ext){
+ int result=-1;
+ char *p=strrchr(filename,'.');
+ if (NULL!=p)
+ result=strcmp(p,ext);
+ return result;
+ }
+
+
 /* Some define for BGP packet dump. */
-static FILE *
+
+
+static gzFile *
 bgp_dump_open_file (struct bgp_dump *bgp_dump)
 {
   int ret;
@@ -118,9 +272,10 @@ bgp_dump_open_file (struct bgp_dump *bgp
       return NULL;
     }

-  if (bgp_dump->fp)
```

```
-    fclose (bgp_dump->fp);
+#ifdef ORIG

+  if (bgp_dump->fp)
+ fclose (bgp_dump->fp);

   oldumask = umask(0777 & ~LOGFILE_MASK);
   bgp_dump->fp = fopen (realpath, "w");
@@ -133,7 +288,40 @@ bgp_dump_open_file (struct bgp_dump *bgp
     }
   umask(oldumask);

-  return bgp_dump->fp;
+#else
+  if (bgp_dump->gfp)
+    {
+ gzclose (bgp_dump->fp);
+ }
+
+if (bgp_dump->fp)
+    {
+       fclose(bgp_dump->fp);
+ }
+
+  //
+  // First make sure that the path for the next file exists
+  //
+#ifdef WITH_DIR_LEVEL
+  mkzebradirs(0,realpath);
+#else
+  mkzebradirs(realpath);
+#endif
+  //
+  // Then create the file for the next snapshot
+  //
+  oldumask = umask(0777 & ~LOGFILE_MASK);
+
+if (0==extcheck(bgp_dump->filename,GZEXT))
```

```
+  bgp_dump->gfp = gzopen (realpath, "wb");
+else bgp_dump->fp = fopen (realpath, "w");
+
+  umask(oldumask);
+
+#endif
+
+if (0==extcheck(bgp_dump->filename,GZEXT))
    return bgp_dump->gfp;
+ else return bgp_dump->fp;
 }

 static int
@@ -266,9 +454,13 @@ bgp_dump_routes_index_table(struct bgp *

   bgp_dump_set_size(obuf, MSG_TABLE_DUMP_V2);

-  fwrite (STREAM_DATA (obuf), stream_get_endp (obuf), 1,
bgp_dump_routes.fp);
-  fflush (bgp_dump_routes.fp);
-}
+  if (NULL!= bgp_dump_routes.gfp) gzwrite(bgp_dump_routes.gfp,
STREAM_DATA (obuf),stream_get_endp (obuf));
+    else {
+ fwrite (STREAM_DATA (obuf), stream_get_endp (obuf), 1,
bgp_dump_routes.fp);
+ fflush (bgp_dump_routes.fp);
+ }
+
+ }


 /* Runs under child process. */
@@ -285,7 +477,7 @@ bgp_dump_routes_func (int afi, int first
   if (!bgp)
     return seq;

-  if (bgp_dump_routes.fp == NULL)
```

```
+  if ((bgp_dump_routes.gfp == NULL) && (bgp_dump_routes.fp ==
NULL))
     return seq;

   /* Note that bgp_dump_routes_index_table will do ipv4 and
   ipv6 peers,
@@ -368,12 +560,14 @@ bgp_dump_routes_func (int afi, int first

       seq++;

-      bgp_dump_set_size(obuf, MSG_TABLE_DUMP_V2);
-      fwrite (STREAM_DATA (obuf), stream_get_endp (obuf), 1,
bgp_dump_routes.fp);
+ bgp_dump_set_size(obuf, MSG_TABLE_DUMP_V2);
+ if (NULL!=bgp_dump_routes.fp) fwrite (STREAM_DATA (obuf),
stream_get_endp (obuf), 1, bgp_dump_routes.fp);

+ else gzwrite(bgp_dump_routes.gfp,STREAM_DATA (obuf),
stream_get_endp (obuf));
     }

-  fflush (bgp_dump_routes.fp);
+    if (NULL!=bgp_dump_routes.fp) fflush (bgp_dump_routes.fp);
+

   return seq;
 }
@@ -397,7 +591,16 @@ bgp_dump_interval_func (struct thread *t
 #endif /* HAVE_IPV6 */
    /* Close the file now. For a RIB dump there's no point in
  * leaving
     * it open until the next scheduled dump starts. */
-  fclose(bgp_dump->fp); bgp_dump->fp = NULL;
+
+
+if (NULL!=bgp_dump->fp) {
+ fclose(bgp_dump->fp);
+ bgp_dump->fp = NULL;
```

```
+}
+else {  gzclose(bgp_dump->gfp);
+ bgp_dump->gfp = NULL;
+}
+
  }
     }

@@ -463,7 +666,7 @@ bgp_dump_state (struct peer *peer, int s
   struct stream *obuf;

   /* If dump file pointer is disabled return immediately. */
-  if (bgp_dump_all.fp == NULL)
+  if ((bgp_dump_all.gfp == NULL) && (bgp_dump_all.fp == NULL))
     return;

   /* Make dump stream. */
@@ -480,8 +683,14 @@ bgp_dump_state (struct peer *peer, int s
   bgp_dump_set_size (obuf, MSG_PROTOCOL_BGP4MP);

   /* Write to the stream. */
-  fwrite (STREAM_DATA (obuf), stream_get_endp (obuf), 1,
bgp_dump_all.fp);
-  fflush (bgp_dump_all.fp);
+
+if (NULL!= bgp_dump_all.fp) {
+fwrite (STREAM_DATA (obuf), stream_get_endp (obuf), 1,
bgp_dump_all.fp);
+ fflush (bgp_dump_all.fp);
+}
+else gzwrite(bgp_dump_all.gfp,STREAM_DATA (obuf),
stream_get_endp (obuf));
+
+
 }

 static void
@@ -491,7 +700,7 @@ bgp_dump_packet_func (struct bgp_dump *b
```

```
   struct stream *obuf;

   /* If dump file pointer is disabled return immediately. */
-  if (bgp_dump->fp == NULL)
+  if ((bgp_dump->gfp == NULL) && (bgp_dump->fp == NULL))
     return;

   /* Make dump stream. */
@@ -516,8 +725,13 @@ bgp_dump_packet_func (struct bgp_dump *b
   bgp_dump_set_size (obuf, MSG_PROTOCOL_BGP4MP);

   /* Write to the stream. */
-  fwrite (STREAM_DATA (obuf), stream_get_endp (obuf), 1,
bgp_dump->fp);
-  fflush (bgp_dump->fp);
+
+if (NULL!= bgp_dump->gfp) gzwrite(bgp_dump->gfp,
STREAM_DATA (obuf),stream_get_endp (obuf));
+ else { fwrite (STREAM_DATA (obuf), stream_get_endp (obuf), 1,
bgp_dump->fp);
+       fflush (bgp_dump->fp);
+
+}
+
 }

 /* Called from bgp_packet.c when BGP packet is received. */
@@ -628,6 +842,7 @@ bgp_dump_set (struct vty *vty, struct bg
     free (bgp_dump->filename);
   bgp_dump->filename = strdup (path);

+
   /* This should be called when interval is expired. */
   bgp_dump_open_file (bgp_dump);

@@ -645,12 +860,21 @@ bgp_dump_unset (struct vty *vty, struct
     }
```

```
   /* This should be called when interval is expired. */
-  if (bgp_dump->fp)
+  if (bgp_dump->gfp)
     {
-      fclose (bgp_dump->fp);
-      bgp_dump->fp = NULL;
-    }
+ gzclose(bgp_dump->gfp);

+
+    bgp_dump->gfp = NULL;
+    }
+
+if (bgp_dump->fp)
+    {
+      fclose (bgp_dump->fp);
+ bgp_dump->fp = NULL;
+    }
+
+
   /* Create interval thread. */
   if (bgp_dump->t_interval)
     {
diff -rup quagga/bgpd/bgp_main.c quagga_new/bgpd/bgp_main.c
--- quagga/bgpd/bgp_main.c 2007-11-01 15:29:11.000000000 +0100
+++ quagga_new/bgpd/bgp_main.c 2007-11-27 08:46:38.000000000
+0100
@@ -61,6 +61,10 @@ void sighup (void);
 void sigint (void);
 void sigusr1 (void);

+#ifndef ORIG
+void sigchld (void);
+#endif
+
 struct quagga_signal_t bgp_signals[] =
 {
   {
```

```
@@ -79,6 +83,15 @@ struct quagga_signal_t bgp_signals[] =
     .signal = SIGTERM,
     .handler = &sigint,
   },
+
+#ifndef ORIG
+  {
+    .signal = SIGCHLD,
+    .handler = &sigchld,
+  },
+#endif
+
+
 };

 /* Configuration file and directory. */
@@ -93,6 +106,7 @@ struct thread_master *master;
 /* Manually specified configuration file name.  */
 char *config_file = NULL;

+
 /* Process ID saved for use by init system */
 const char *pid_file = PATH_BGPD_PID;

@@ -192,6 +206,18 @@ sigusr1 (void)
   zlog_rotate (NULL);
 }
+#ifndef ORIG
+/* SIGCHLD handler cited and described in all books*/
+void sigchld()
+{
+  int stat;
+
+  zlog_debug("Entered sigchld()");
+  while(waitpid(-1, &stat, WNOHANG) > 0);
+}
+#endif
+
```

```
+
 /* Main routine of bgpd. Treatment of argument and start bgp
 finite
    state machine is handled at here. */
 int
@@ -273,6 +299,7 @@ main (int argc, char **argv)
    print_version (progname);
    exit (0);
    break;
+
  case 'C':
    dryrun = 1;
    break;
diff -rup quagga/bgpd/Makefile.am quagga_new/bgpd/Makefile.am
--- quagga/bgpd/Makefile.am 2007-05-10 04:38:51.000000000 +0200
+++ quagga_new/bgpd/Makefile.am 2007-12-12 12:22:22.000000000
+0100
@@ -1,6 +1,6 @@
 ## Process this file with automake to produce Makefile.in.

-INCLUDES = @INCLUDES@ -I.. -I$(top_srcdir) -I$(top_srcdir)/lib
@SNMP_INCLUDES@
+INCLUDES = @INCLUDES@ -I.. -I$(top_srcdir) -I$(top_srcdir)/lib
-I /usr/include/ @SNMP_INCLUDES@
 DEFS = @DEFS@ -DSYSCONFDIR=\"$(sysconfdir)/\"
 INSTALL_SDATA=@INSTALL@ -m 600

@@ -22,7 +22,7 @@ noinst_HEADERS = \
  bgp_advertise.h bgp_snmp.h bgp_vty.h

 bgpd_SOURCES = bgp_main.c
-bgpd_LDADD = libbgp.a ../lib/libzebra.la @LIBCAP@ @LIBM@
+bgpd_LDADD = libbgp.a ../lib/libzebra.la @LIBCAP@ @LIBM@ -lz

 examplesdir = $(exampledir)
 dist_examples_DATA = bgpd.conf.sample bgpd.conf.sample2
diff -rup quagga/configure.ac quagga_new/configure.ac
--- quagga/configure.ac 2007-09-07 18:54:01.000000000 +0200
```

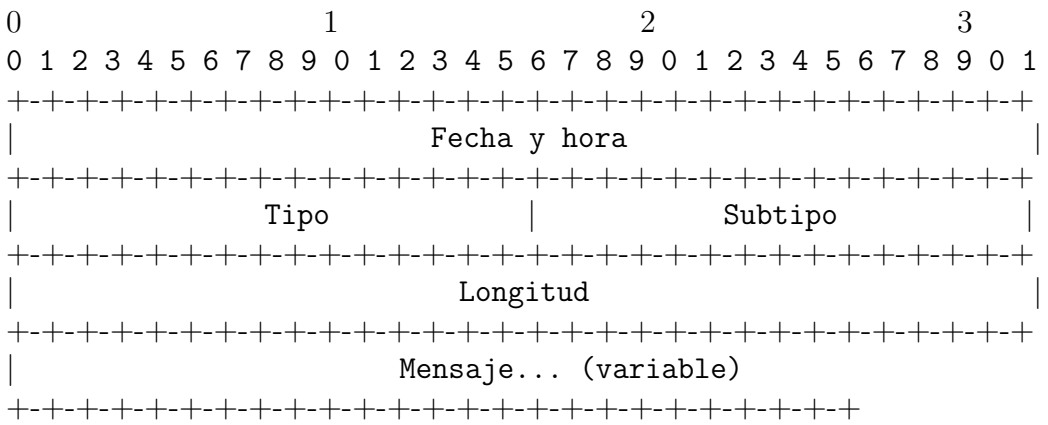
```
+++ quagga_new/configure.ac 2007-12-12 11:52:27.000000000 +0100
@@ -1239,6 +1239,16 @@ if test "${enable_snmp}" = "yes"; then
     AC_SUBST(SNMP_INCLUDES)
 fi

+
+dnl -------------------------------
+dnl libz needed
+dnl -------------------------------
+AC_CHECK_HEADER([/usr/include/zlib.h])
+LIBZ="-lz"
+
+
+
+
 dnl ---------------------------
 dnl sockaddr and netinet checks
 dnl ---------------------------
@@ -1452,7 +1462,7 @@ compiler                : ${CC}
 compiler flags          : ${CFLAGS}
 make                    : ${MAKE-make}
 includes                : ${INCLUDES} ${SNMP_INCLUDES}
-linker flags            : ${LDFLAGS} ${LIBS} ${
${LIBREADLINE} ${LIBM}
+linker flags            : ${LDFLAGS} ${LIBS} ${LIBCAP}
${LIBREADLINE} ${LIBM} ${LIBZ}
 state file directory    : ${quagga_statedir}
 config file directory   : `eval echo \`echo ${sysconfdir}\``
 example directory       : `eval echo \`echo ${exampledir}\``
```

# Apéndice F

# Formato MRT

Todos los mensajes de formato MRT tienen una cabecera común que incluye fecha y hora, tipo, subtipo, y longitud de campo. La cabecera es seguida de un campo mensaje. La cabecera MRT común se ilustra a continuación.




Descripciones de los campos de la cabecera:

`Fecha y hora`
Tiempo en segundos desde el 1 Enero 1970 00:00:00 UTC

`Tipo`
Campo de dos octetos que indica el `Tipo` de información contenida en el campo `Mensaje`. Los tipos de 0 a 4 son mensajes informativos relacionados con el estado del colector MRT, mientras que los tipos de 5 en adelante son

usados para transmitir información de encaminamiento.

`Subtipo`
Campo de dos octetos usado para distinguir más información de los mensajes en un mensaje particular `Tipo`.

`Longitud`
Mensaje de cuatro octetos de longitud de campo. El campo longitud contiene el número de octetos del mensaje. No incluye la longitud de la cabecera común MRT.

`Mensaje`
Mensaje de longitud variable. El contenido de este campo depende del contexto de los campos `Tipo` y `Subtipo`.

# Apéndice G

# BGP Wedgies: configuración

## G.1. lab.conf

```
LAB_DESCRIPTION="BGP Wedgies"
LAB_AUTHOR="Mercedes Bernal Pérez"
LAB_EMAIL=mercedesbernalperez@gmail.com

router1[0]="A"
router1[1]="B"

router2[0]="A"
router2[1]="C"

router3[0]="D"
router3[1]="C"
router3[2]="F"

router4[0]="B"
router4[1]="D"
```

## G.2. router1.startup

```
/sbin/ifconfig eth0 193.10.11.1 up
/sbin/ifconfig eth1 195.11.14.1 up
```

```
cat > /etc/quagga/bgpd.conf <<EOF
hostname router1
password zebra
log file /var/log/quagga/bgpd1.log
router bgp 65001
  bgp router-id 193.10.11.1
  neighbor 193.10.11.2 remote-as 65002
  !neighbor 195.11.14.3 remote-as 65003
  neighbor 195.11.14.4 remote-as 65004

  neighbor 193.10.11.2 route-map lowerPrefPeer out

  ! lower pref peer vs customer
  route-map lowerPrefPeer permit 10
  set local-preference 90

  dump bgp updates /var/www/repo/%Y.%m/updates.%Y%m%d.%H%M.gz 5m
  dump bgp routes-mrt /var/www/repo/%Y.%m/bview.%Y%m%d.%H%M.gz 8h

EOF

cat > /etc/quagga/zebra.conf <<EOF
hostname router1
password zebra
log file /var/log/quagga/zebra.log
EOF

cat > /etc/quagga/daemons <<EOF
zebra=yes
bgpd=yes
EOF

/etc/init.d/quagga start
```

## G.3. router2.startup

```
/sbin/ifconfig eth0 193.10.11.2 up
```

```
/sbin/ifconfig eth1 120.1.2.1 up

cat > /etc/quagga/bgpd.conf <<EOF
hostname router2
password zebra
log file /var/log/quagga/bgpd2.log
router bgp 65002
  bgp router-id 193.10.11.2
  neighbor 193.10.11.1 remote-as 65001
  !neighbor 193.10.11.1 route-map lowerPrefPeer out
  neighbor 120.1.2.3 remote-as 65003

  neighbor 120.1.2.3 route-map PEER2-3 in

  ! lower pref peer vs customer
  route-map lowerPrefPeer permit 10
  set local-preference 90

  ! lower pref anuncio red 210
  route-map PEER2-3 permit 10
  match community 70
  set local-preference 70

  ip community-list 70 permit 65002:70

  dump bgp updates /var/www/repo/%Y.%m/updates.%Y%m%d.%H%M.gz 5m
  dump bgp routes-mrt /var/www/repo/%Y.%m/bview.%Y%m%d.%H%M.gz 8h

EOF

cat > /etc/quagga/zebra.conf <<EOF
hostname router2
password zebra
log file /var/log/quagga/zebra.log
EOF

cat > /etc/quagga/daemons <<EOF
zebra=yes
```

```
bgpd=yes
EOF

/etc/init.d/quagga start
```

## G.4. router3.startup

```
/sbin/ifconfig eth0 140.1.2.3 up
/sbin/ifconfig eth2 210.1.2.3 up
/sbin/ifconfig eth1 120.1.2.3 up

cat > /etc/quagga/bgpd.conf <<EOF
hostname router2
password zebra
log file /var/log/quagga/bgpd2.log
router bgp 65003
  bgp router-id 120.1.2.3
  network 210.1.2.0/24
  neighbor 120.1.2.1 remote-as 65002
  neighbor 140.1.2.4 remote-as 65004

  ! lower pref cuando anuncia red 210
  neighbor 120.1.2.1 send-community
  neighbor 120.1.2.1 route-map PEER3-2 out

  ! lower pref backup
  neighbor 140.1.2.4 send-community
  neighbor 140.1.2.4 route-map PEER3-4 out

  route-map PEER3-2 permit 10
  match ip address prefix-list PLIST
  set community 65002:70

  ip prefix-list PLIST permit 210.1.2.0/24

  route-map PEER3-4 permit 10
```

```
  match ip address prefix-list PLIST
  set community 65004:50

  dump bgp updates /var/www/repo/%Y.%m/updates.%Y%m%d.%H%M.gz 5m
  dump bgp routes-mrt /var/www/repo/%Y.%m/bview.%Y%m%d.%H%M.gz 8h

EOF

cat > /etc/quagga/zebra.conf <<EOF
hostname router2
password zebra
log file /var/log/quagga/zebra.log
EOF

cat > /etc/quagga/daemons <<EOF
zebra=yes
bgpd=yes
EOF

/etc/init.d/quagga start
```

## G.5. router4.startup

```
/sbin/ifconfig eth0 195.11.14.4 up
/sbin/ifconfig eth1 140.1.2.4 up

cat > /etc/quagga/bgpd.conf <<EOF
hostname router4
password zebra
log file /var/log/quagga/bgpd4.log
router bgp 65004
  bgp router-id 140.1.2.4
  neighbor 195.11.14.1 remote-as 65001
  neighbor 140.1.2.3 remote-as 65003

  neighbor 140.1.2.3 route-map PEER4-3 in
```

```
  ! lower pref anuncio red 210
  route-map PEER4-3 permit 10
  match community 50
  set local-preference 50

  ip community-list 50 permit 65004:50

  dump bgp updates /var/www/repo/%Y.%m/updates.%Y%m%d.%H%M.gz 5m
  dump bgp routes-mrt /var/www/repo/%Y.%m/bview.%Y%m%d.%H%M.gz 8h

EOF

cat > /etc/quagga/zebra.conf <<EOF
hostname router1
password zebra
log file /var/log/quagga/zebra.log
EOF

cat > /etc/quagga/daemons <<EOF
zebra=yes
bgpd=yes
EOF

/etc/init.d/quagga start
```

# Apéndice H

# Modificación Quagga Multihoming

```
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_advertise.o y
quagga-0.99.10.original/bgpd/bgp_advertise.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_aspath.o y
quagga-0.99.10.original/bgpd/bgp_aspath.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_attr.o y
quagga-0.99.10.original/bgpd/bgp_attr.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_clist.o y
quagga-0.99.10.original/bgpd/bgp_clist.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_community.o y
quagga-0.99.10.original/bgpd/bgp_community.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgpd y
quagga-0.99.10.original/bgpd/bgpd son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_damp.o y
quagga-0.99.10.original/bgpd/bgp_damp.o son distintos
diff quagga-0.99.10a/bgpd/bgpd.c quagga-0.99.10.original/bgpd/
bgpd.c
815d814
< #if 0
818,823c817
< #else
< //
< // prueba del -p de la linea de comando
< //
```

```
<   peer->port = bm->port;
< #endif
---
>
3006,3010d2999
< #if 0
<  // Lo suyo sería hacer esto para volver al puerto
<  // que hemos dicho en la linea de comando
<  peer->port = bm->port;
< #endif
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_debug.o y
quagga-0.99.10.original/bgpd/bgp_debug.o son distintos
diff quagga-0.99.10a/bgpd/bgpd.h quagga-0.99.10.original/bgpd/
bgpd.h
796,799d795
< //Inicio Mercedes
< extern int bgp_table;
< //Fin Mercedes
<
Los ficheros binarios quagga-0.99.10a/bgpd/bgpd.o y
quagga-0.99.10.original/bgpd/bgpd.o son distintos
Sólo en quagga-0.99.10a/bgpd/: bgpd.patch
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_dump.o y
quagga-0.99.10.original/bgpd/bgp_dump.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_ecommunity.o y
quagga-0.99.10.original/bgpd/bgp_ecommunity.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_filter.o y
quagga-0.99.10.original/bgpd/bgp_filter.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_fsm.o y
quagga-0.99.10.original/bgpd/bgp_fsm.o son distintos
diff quagga-0.99.10a/bgpd/bgp_main.c quagga-0.99.10.original/
bgpd/bgp_main.c
56d55
<   { "table",  required_argument, NULL, 't'},
125,129d123
< //Inicio Mercedes
< /* Routing table */
< int bgp_table = 0;
```

```
< //Fin Mercedes
<
155d148
< -t, --table        Routing table\n\
230c223
<       opt = getopt_long (argc, argv, "df:i:hp:l:A:P:rnu:g:vCt:
", longopts, 0);
---
>       opt = getopt_long (argc, argv, "df:i:hp:l:A:P:rnu:g:vC"
, longopts, 0);
236c229
<       {
---
>  {
250,252d242
< #if 0
<    fprintf(stderr,"bgpd: optarg = '%s' ; tmp_port = %d
(0x%04x)\n",optarg,tmp_port,tmp_port);
< #endif
257,259d246
< #if 0
<    fprintf(stderr,"bgpd: bm->port = %d (0x%04x)\n",
bm->port,bm->port);
< #endif
301,306d287
<  //Inicio Mercedes
<  case 't':
<    bgp_table = atoi(optarg);
<    zlog_warn("bgp_main: La tabla introducida es %d",
bgp_table);
<    break;
<  //Fin Mercedes
348c329
<   zlog_notice ("BGPd %s %s %s starting: vty@%d, bgp@%s:%d",
QUAGGA_VERSION, __DATE__, __TIME__,
---
>   zlog_notice ("BGPd %s starting: vty@%d, bgp@%s:%d",
QUAGGA_VERSION,
```

```
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_main.o y
quagga-0.99.10.original/bgpd/bgp_main.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_mplsvpn.o y
quagga-0.99.10.original/bgpd/bgp_mplsvpn.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_network.o y
quagga-0.99.10.original/bgpd/bgp_network.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_nexthop.o y
quagga-0.99.10.original/bgpd/bgp_nexthop.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_open.o y
quagga-0.99.10.original/bgpd/bgp_open.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_packet.o y
quagga-0.99.10.original/bgpd/bgp_packet.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_regex.o y
quagga-0.99.10.original/bgpd/bgp_regex.o son distintos
diff quagga-0.99.10a/bgpd/bgp_route.c quagga-0.99.10.original/
bgpd/bgp_route.c
6315d6314
<  vty_out (vty, "BGP table ID is %d%s", bgp_table,
VTY_NEWLINE);
9580d9578
<       vty_out (vty, "BGP table ID is %d%s", bgp_table,
VTY_NEWLINE);
9598d9595
<    vty_out (vty, "BGP table ID is %d%s", bgp_table,
VTY_NEWLINE);
9623d9619
<    vty_out (vty, "BGP table ID is %d%s", bgp_table,
VTY_NEWLINE);
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_routemap.o y
quagga-0.99.10.original/bgpd/bgp_routemap.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_route.o y
quagga-0.99.10.original/bgpd/bgp_route.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_snmp.o y
quagga-0.99.10.original/bgpd/bgp_snmp.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_table.o y
quagga-0.99.10.original/bgpd/bgp_table.o son distintos
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_vty.o y
quagga-0.99.10.original/bgpd/bgp_vty.o son distintos
```

```
diff quagga-0.99.10a/bgpd/bgp_zebra.c quagga-0.99.10.original/
bgpd/bgp_zebra.c
238c238
<
---
>
678d677
<
718,723d716
<
<       //Inicio Mercedes
<       /* Rellenamos el campo tabla de la estructura
        * zapi_ipv4 */
<       api.table = bgp_table;
<       zlog_warn("bgp_zebra.c (bgp_zebra_announce __ipv4__) :
Rellenar campo tabla de estructura zapi_ipv4 con valor %d.",
api.table);
<       //Fin Mercedes
800,805d792
<
<       //Inicio Mercedes
<       /* Rellenamos el campo tabla de la estructura
        * zapi_ipv6 */
<       api.table = bgp_table;
<       zlog_warn("bgp_zebra.c (bgp_zebra_announce __ipv6__) :
Rellenar campo tabla de estructura zapi_ipv6 con valor %d.",
api.table);
<       //Fin Mercedes
826d812
<
865,870d850
<
<       //Inicio Mercedes
<       /* Rellenamos el campo tabla de la estructura
        * zapi_ipv4 */
<       api.table = bgp_table;
<       zlog_warn("bgp_zebra.c (bgp_zebra_withdraw __ipv4__) :
Rellenar campo tabla de estructura zapi_ipv4 con valor %d.",
```

```
api.table);
<       //Fin Mercedes
928,933d907
<
<  //Inicio Mercedes
<       /* Rellenamos el campo tabla de la estructura
        * zapi_ipv6 */
<       api.table = bgp_table;
<       zlog_warn("bgp_zebra.c (bgp_zebra_withdraw __ipv6__) :
Rellenar campo tabla de estructura zapi_ipv6 con valor %d.",
api.table);
<       //Fin Mercedes
Los ficheros binarios quagga-0.99.10a/bgpd/bgp_zebra.o y
quagga-0.99.10.original/bgpd/bgp_zebra.o son distintos
Subdirectorios comunes: quagga-0.99.10a/bgpd/.deps y
quagga-0.99.10.original/bgpd/.deps
Los ficheros binarios quagga-0.99.10a/bgpd/libbgp.a y
quagga-0.99.10.original/bgpd/libbgp.a son distintos
Subdirectorios comunes: quagga-0.99.10a/bgpd/.libs y
quagga-0.99.10.original/bgpd/.libs
diff quagga-0.99.10a/bgpd/Makefile quagga-0.99.10.original/
bgpd/Makefile
96,97c96,97
< ACLOCAL = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run aclocal-1.10
< AMTAR = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run tar
---
> ACLOCAL = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run aclocal-1.10
> AMTAR = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run tar
99,101c99,101
< AUTOCONF = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run autoconf
< AUTOHEADER = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run autoheader
< AUTOMAKE = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
```

```
missing --run automake-1.10
---
> AUTOCONF = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run autoconf
> AUTOHEADER = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run autoheader
> AUTOMAKE = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run automake-1.10
107c107
< CONFDATE = 20081212
---
> CONFDATE = 20080728
112,114c112,114
< CXXCPP =
< CXXDEPMODE = depmode=none
< CXXFLAGS =
---
> CXXCPP = g++ -E
> CXXDEPMODE = depmode=gcc3
> CXXFLAGS = -g -O2
118d117
< DSYMUTIL =
157c156
< MAKEINFO = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run makeinfo
---
> MAKEINFO = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run makeinfo
160d158
< NMEDIT =
182c180
< SHELL = /bin/sh
---
> SHELL = /bin/bash
190,193c188,191
< abs_builddir = /home/merce/Escritorio/quagga-0.99.10a/bgpd
< abs_srcdir = /home/merce/Escritorio/quagga-0.99.10a/bgpd
< abs_top_builddir = /home/merce/Escritorio/quagga-0.99.10a
```

```
< abs_top_srcdir = /home/merce/Escritorio/quagga-0.99.10a
---
> abs_builddir = /home/merce/Escritorio/quagga-0.99.10/bgpd
> abs_srcdir = /home/merce/Escritorio/quagga-0.99.10/bgpd
> abs_top_builddir = /home/merce/Escritorio/quagga-0.99.10
> abs_top_srcdir = /home/merce/Escritorio/quagga-0.99.10
195c193
< ac_ct_CXX =
---
> ac_ct_CXX = g++
215,216c213,214
< enable_vty_group = quagga
< exampledir = /etc/quagga
---
> enable_vty_group =
> exampledir = ${prefix}/etc
225,226c223,224
< infodir = ${prefix}/share/info
< install_sh = $(SHELL) /home/merce/Escritorio/quagga-0.99.10a/
install-sh
---
> infodir = ${datarootdir}/info
> install_sh = $(SHELL) /home/merce/Escritorio/quagga-0.99.10/
install-sh
230,231c228,229
< localstatedir = /var/run/quagga
< mandir = ${prefix}/share/man
---
> localstatedir = ${prefix}/var
> mandir = ${datarootdir}/man
235,237c233,235
< pkgsrcdir = pkgsrc
< pkgsrcrcdir = /etc/init.d
< prefix = /usr
---
> pkgsrcdir =
> pkgsrcrcdir =
> prefix = /usr/local
```

```
240c238
< quagga_statedir = /var/run/quagga
---
> quagga_statedir = /var/run
244c242
< sysconfdir = /etc/quagga
---
> sysconfdir = ${prefix}/etc
diff quagga-0.99.10a/bgpd/Makefile.in quagga-0.99.10.original/
bgpd/Makefile.in
118d117
< DSYMUTIL = @DSYMUTIL@
160d158
< NMEDIT = @NMEDIT@


Los ficheros binarios quagga-0.99.10a/zebra/connected.o y
quagga-0.99.10.original/zebra/connected.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/debug.o y
quagga-0.99.10.original/zebra/debug.o son distintos
Subdirectorios comunes: quagga-0.99.10a/zebra/.deps y
quagga-0.99.10.original/zebra/.deps
Los ficheros binarios quagga-0.99.10a/zebra/if_netlink.o y
quagga-0.99.10.original/zebra/if_netlink.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/if_proc.o y
quagga-0.99.10.original/zebra/if_proc.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/interface.o y
quagga-0.99.10.original/zebra/interface.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/ioctl_null.o y
quagga-0.99.10.original/zebra/ioctl_null.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/ioctl.o y
quagga-0.99.10.original/zebra/ioctl.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/ipforward_proc.o y
quagga-0.99.10.original/zebra/ipforward_proc.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/irdp_interface.o y
quagga-0.99.10.original/zebra/irdp_interface.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/irdp_main.o y
quagga-0.99.10.original/zebra/irdp_main.o son distintos
```

```
Los ficheros binarios quagga-0.99.10a/zebra/irdp_packet.o y
quagga-0.99.10.original/zebra/irdp_packet.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/kernel_null.o y
quagga-0.99.10.original/zebra/kernel_null.o son distintos
Subdirectorios comunes: quagga-0.99.10a/zebra/.libs y
quagga-0.99.10.original/zebra/.libs
Los ficheros binarios quagga-0.99.10a/zebra/main.o y
quagga-0.99.10.original/zebra/main.o son distintos
diff quagga-0.99.10a/zebra/Makefile quagga-0.99.10.original/
zebra/Makefile
100,101c100,101
< ACLOCAL = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run aclocal-1.10
< AMTAR = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run tar
---
> ACLOCAL = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run aclocal-1.10
> AMTAR = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run tar
103,105c103,105
< AUTOCONF = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run autoconf
< AUTOHEADER = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run autoheader
< AUTOMAKE = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run automake-1.10
---
> AUTOCONF = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run autoconf
> AUTOHEADER = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run autoheader
> AUTOMAKE = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run automake-1.10
111c111
< CONFDATE = 20081212
---
> CONFDATE = 20080728
```

```
116,118c116,118
< CXXCPP =
< CXXDEPMODE = depmode=none
< CXXFLAGS =
---
> CXXCPP = g++ -E
> CXXDEPMODE = depmode=gcc3
> CXXFLAGS = -g -O2
122d121
< DSYMUTIL =
161c160
< MAKEINFO = ${SHELL} /home/merce/Escritorio/quagga-0.99.10a/
missing --run makeinfo
---
> MAKEINFO = ${SHELL} /home/merce/Escritorio/quagga-0.99.10/
missing --run makeinfo
164d162
< NMEDIT =
186c184
< SHELL = /bin/sh
---
> SHELL = /bin/bash
194,197c192,195
< abs_builddir = /home/merce/Escritorio/quagga-0.99.10a/
zebra
< abs_srcdir = /home/merce/Escritorio/quagga-0.99.10a/
zebra
< abs_top_builddir = /home/merce/Escritorio/
quagga-0.99.10a
< abs_top_srcdir = /home/merce/Escritorio/
quagga-0.99.10a
---
> abs_builddir = /home/merce/Escritorio/quagga-0.99.10/
zebra
> abs_srcdir = /home/merce/Escritorio/quagga-0.99.10/
zebra
> abs_top_builddir = /home/merce/Escritorio/
quagga-0.99.10
```

```
> abs_top_srcdir = /home/merce/Escritorio/quagga-0.99.10
199c197
< ac_ct_CXX =
---
> ac_ct_CXX = g++
219,220c217,218
< enable_vty_group = quagga
< exampledir = /etc/quagga
---
> enable_vty_group =
> exampledir = ${prefix}/etc
229,230c227,228
< infodir = ${prefix}/share/info
< install_sh = $(SHELL) /home/merce/Escritorio/
quagga-0.99.10a/install-sh
---
> infodir = ${datarootdir}/info
> install_sh = $(SHELL) /home/merce/Escritorio/
quagga-0.99.10/install-sh
234,235c232,233
< localstatedir = /var/run/quagga
< mandir = ${prefix}/share/man
---
> localstatedir = ${prefix}/var
> mandir = ${datarootdir}/man
239,241c237,239
< pkgsrcdir = pkgsrc
< pkgsrcrcdir = /etc/init.d
< prefix = /usr
---
> pkgsrcdir =
> pkgsrcrcdir =
> prefix = /usr/local
244c242
< quagga_statedir = /var/run/quagga
---
> quagga_statedir = /var/run
248c246
```

```
< sysconfdir = /etc/quagga
---
> sysconfdir = ${prefix}/etc
diff quagga-0.99.10a/zebra/Makefile.in quagga-0.99.10.original/
zebra/Makefile.in
122d121
< DSYMUTIL = @DSYMUTIL@
164d162
< NMEDIT = @NMEDIT@
Los ficheros binarios quagga-0.99.10a/zebra/misc_null.o y
quagga-0.99.10.original/zebra/misc_null.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/redistribute_null.o
y quagga-0.99.10.original/zebra/redistribute_null.o son
distintos
Los ficheros binarios quagga-0.99.10a/zebra/redistribute.o y
quagga-0.99.10.original/zebra/redistribute.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/router-id.o y
quagga-0.99.10.original/zebra/router-id.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/rtadv.o y
quagga-0.99.10.original/zebra/rtadv.o son distintos
diff quagga-0.99.10a/zebra/rt_netlink.c quagga-0.99.10.original/
zebra/rt_netlink.c
955,959c955
< //Inicio Mercedes
<       /*if (h->nlmsg_type == RTM_NEWROUTE)
<         rib_add_ipv6 (ZEBRA_ROUTE_KERNEL, 0, &p, gate, index,
0, 0, 0);
<       else
<         rib_delete_ipv6 (ZEBRA_ROUTE_KERNEL, 0, &p, gate,
index, 0);*/
---
>
961c957
<         rib_add_ipv6 (ZEBRA_ROUTE_KERNEL, 0, &p, gate, index,
table, 0, 0);
---
>         rib_add_ipv6 (ZEBRA_ROUTE_KERNEL, 0, &p, gate, index,
0, 0, 0);
```

```
963,964c959
<         rib_delete_ipv6 (ZEBRA_ROUTE_KERNEL, 0, &p, gate,
index, table);
< //Fin Mercedes
---
>         rib_delete_ipv6 (ZEBRA_ROUTE_KERNEL, 0, &p, gate,
index, 0);
1848,1850d1842
<  //Inicio Mercedes
<  zlog_warn("***rt_netlink: kernel_add_ipv6***");
<  //Fin Mercedes
Los ficheros binarios quagga-0.99.10a/zebra/rt_netlink.o y
quagga-0.99.10.original/zebra/rt_netlink.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/rtread_netlink.o y
quagga-0.99.10.original/zebra/rtread_netlink.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/test_main.o y
quagga-0.99.10.original/zebra/test_main.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/testzebra y
quagga-0.99.10.original/zebra/testzebra son distintos
Los ficheros binarios quagga-0.99.10a/zebra/zebra y
quagga-0.99.10.original/zebra/zebra son distintos
diff quagga-0.99.10a/zebra/zebra_rib.c quagga-0.99.10.original/
zebra/zebra_rib.c
132,135d131
<   //Inicio Mercedes
<   struct vrf *table;
<   //Fin Mercedes
<
137,140c133
<   //Inicio Mercedes
<   //vrf_vector = vector_init (1);
<   vrf_vector = vector_init (2);
<   //Fin Mercedes
---
>   vrf_vector = vector_init (1);
147,153d139
<
<   //Inicio Mercedes
```

```
<   table = vrf_alloc("IP-Routing-Table-2");
<
<   vector_set_index (vrf_vector, 10, table);
<   //Fin Mercedes
<
1462d1447
<
1525d1509
<
1798,1801d1781
<   //Inicio Mercedes
<   zlog_warn("rib_add_ipv4_multipath: La tabla recibida es %d
\n", rib->table);
<   //Fin Mercedes
<
1808,1817c1788
<   //Inicio Mercedes
<   // table = vrf_table (AFI_IP, SAFI_UNICAST, 0);
<   if (rib->table!=0 && rib->table!=254 && rib->table!=255) {
<        table = vrf_table (AFI_IP, SAFI_UNICAST, rib->table);
<     }
<     else {
<  table = vrf_table (AFI_IP, SAFI_UNICAST, 0);
<     }
<    //Fin Mercedes
<
---
>   table = vrf_table (AFI_IP, SAFI_UNICAST, 0);
1839d1809
<
1849d1818
<
1856,1859d1824
<   //Inicio Mercedes
<   zlog_warn("Meto en la tabla %d la direccion: %s\n",
rib->table, inet_ntoa (p->prefix));
<   //Fin Mercedes
<
```

```
1899,1904d1863
<
<  //Inicio Mercedes
<  zlog_warn("\n**rib_delete_ipv4**\n");
<  zlog_warn("vrf_id %d", vrf_id);
<  //Fin Mercedes
<
1906,1915c1865
<   //Inicio Mercedes
<   // table = vrf_table (AFI_IP, SAFI_UNICAST, 0);
<   if (vrf_id!=0 && vrf_id!=254 && vrf_id!=255) {
<        table = vrf_table (AFI_IP, SAFI_UNICAST, vrf_id);
<     }
<     else {
<  table = vrf_table (AFI_IP, SAFI_UNICAST, 0);
<     }
<    //Fin Mercedes
<
---
>   table = vrf_table (AFI_IP, SAFI_UNICAST, 0);
2020,2023d1969
<   //Inicio Mercedes
<   zlog_warn("Elimino en la tabla %d la direccion: %s\n",
rib->table, inet_ntoa (p->prefix));
<   //Fin Mercedes
<
2038c1984
<
---
>
2378,2387c2324
<   //Inicio Mercedes
<   // table = vrf_table (AFI_IP6, SAFI_UNICAST, 0);
<   if (vrf_id!=0 && vrf_id!=254 && vrf_id!=255) {
<        table = vrf_table (AFI_IP6, SAFI_UNICAST, vrf_id);
<     }
<     else {
<  table = vrf_table (AFI_IP6, SAFI_UNICAST, 0);
```

```
<     }
<    //Fin Mercedes
<
---
>   table = vrf_table (AFI_IP6, SAFI_UNICAST, 0);
2458,2461d2394
<   //Inicio Mercedes
<   zlog_warn("Meto en la tabla %d la direccion ipv6: %s\n",
rib->table, inet6_ntoa (p->prefix));
<   //Fin Mercedes
<
2491,2500c2424
<   //Inicio Mercedes
<   // table = vrf_table (AFI_IP6, SAFI_UNICAST, 0);
<   if (vrf_id!=0 && vrf_id!=254 && vrf_id!=255) {
<        table = vrf_table (AFI_IP6, SAFI_UNICAST, vrf_id);
<     }
<     else {
<  table = vrf_table (AFI_IP6, SAFI_UNICAST, 0);
<     }
<    //Fin Mercedes
<
---
>   table = vrf_table (AFI_IP6, SAFI_UNICAST, 0);
2594,2597d2517
<
<     //Inicio Mercedes
<   zlog_warn("Elimino en la tabla %d la direccion ipv6: %s\n",
rib->table, inet6_ntoa (p->prefix));
<   //Fin Mercedes
Los ficheros binarios quagga-0.99.10a/zebra/zebra_rib.o y
quagga-0.99.10.original/zebra/zebra_rib.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/zebra_routemap.o y
quagga-0.99.10.original/zebra/zebra_routemap.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/zebra_snmp.o y
quagga-0.99.10.original/zebra/zebra_snmp.o son distintos
Los ficheros binarios quagga-0.99.10a/zebra/zebra_vty.o y
quagga-0.99.10.original/zebra/zebra_vty.o son distintos
```

```
diff quagga-0.99.10a/zebra/zserv.c quagga-0.99.10.original/
zebra/zserv.c
744,748d743
<
<   //Inicio Mercedes
<   /* Tabla en la que se va a introducir la ruta */
<   int table;
<   //Fin Mercedes
808,815c803
<
<    //Inicio Mercedes
<   /* Recibimos la tabla y la mostramos */
<   table = stream_getl(s);
<   zlog_warn("zread_ipv4_add: La tabla recibida es %d \n",
table);
<   //Fin Mercedes
<
<   //Inicio Mercedes
---
>
817,822c805
<   if(table==0)
<     rib->table = zebrad.rtm_table_default;
<   else
<     rib->table = table;
<   //Fin Mercedes
<
---
>   rib->table=zebrad.rtm_table_default;
840,844d822
<
<   //Inicio Mercedes
<   /* Tabla en la que se va a introducir la ruta */
<   int table;
<   //Fin Mercedes
900,918c878,880
<
<  //Inicio Mercedes
```

```
<   /* Recibimos la tabla y la mostramos */
<   table = stream_getl(s);
<   zlog_warn("\nzread_ipv4_delete: La tabla recibida es %d \n",
table);
<   //Fin Mercedes
<
<   //Inicio Mercedes
<   /* Table */
<   if(table==0){
<     zlog_warn("\nzread_ipv4_delete: Llamando a rib_delete_ipv4
con tabla %d \n", client->rtm_table);
<     rib_delete_ipv4 (api.type, api.flags, &p, &nexthop,
ifindex, client->rtm_table);
<   }
<   else{
<     zlog_warn("\nzread_ipv4_delete: Llamando a rib_delete_ipv4
con tabla %d \n", table);
<     rib_delete_ipv4 (api.type, api.flags, &p, &nexthop,
ifindex, table);
<   }
<   //Fin Mercedes
<
---
>
>   rib_delete_ipv4 (api.type, api.flags, &p, &nexthop, ifindex,
>     client->rtm_table);
956,960d917
<
<   //Inicio Mercedes
<   /* Tabla en la que se va a introducir la ruta */
<   int table;
<   //Fin Mercedes
1008,1016c965
<
<   //Inicio Mercedes
<   /* Recibimos la tabla y la mostramos */
<   table = stream_getl(s);
<   zlog_warn("\nzread_ipv6_add: La tabla recibida es %d \n",
```

```
table);
<   //Fin Mercedes
<
<   //Inicio Mercedes
<   // Sustituimos el sexto parámetro que era 0 por la variable
table, que contiene el número de tabla
---
>
1018c967
<     rib_add_ipv6 (api.type, api.flags, &p, NULL, ifindex,
table, api.metric,
---
>     rib_add_ipv6 (api.type, api.flags, &p, NULL, ifindex, 0,
api.metric,
1021c970
<     rib_add_ipv6 (api.type, api.flags, &p, &nexthop, ifindex,
table, api.metric,
---
>     rib_add_ipv6 (api.type, api.flags, &p, &nexthop, ifindex,
0, api.metric,
1023,1024d971
<   //Fin Mercedes
<
1038,1042d984
<
<   //Inicio Mercedes
<   /* Tabla en la que se va a introducir la ruta */
<   int table;
<   //Fin Mercedes
1089,1097c1031
<
<   //Inicio Mercedes
<   /* Recibimos la tabla y la mostramos */
<   table = stream_getl(s);
<   zlog_warn("\nzread_ipv6_delete: La tabla recibida es %d \n",
table);
<   //Fin Mercedes
<
```

```
<   //Inicio Mercedes
<   //Sustituimos el último parámetro que era 0 por la variable
table que contiene el  número de tabla
---
>
1099c1033
<     rib_delete_ipv6 (api.type, api.flags, &p, NULL, ifindex,
table);
---
>     rib_delete_ipv6 (api.type, api.flags, &p, NULL, ifindex,
0);
1101,1102c1035
<     rib_delete_ipv6 (api.type, api.flags, &p, &nexthop,
ifindex, table);
<   //Fin Mercedes
---
>     rib_delete_ipv6 (api.type, api.flags, &p, &nexthop,
ifindex, 0);
Los ficheros binarios quagga-0.99.10a/zebra/zserv.o y
quagga-0.99.10.original/zebra/zserv.o son distintos
```

# Apéndice I

# Compilación del núcleo Linux

Toda máquina virtual está compuesta por dos componentes fundamentales: un **núcleo** y un **sistema de ficheros**. El núcleo de la máquina virtual es un componente especial de software que emula completamente el núcleo de la máquina real. En realidad, un núcleo para una máquina virtual de Netkit no es otra cosa que una versión especial de un núcleo compilado para ser ejecutado como un proceso en espacio de usuario.

En principio, nada obliga a usar una combinación particular del núcleo o sistema de ficheros. Sin embargo, para llevar a cabo la prueba de concepto de **multihoming**, ha sido necesario compilar el núcleo para activar las opciones de **múltiples tablas**, deshabilitadas por defecto para la versión 2.6.27.7 del núcleo de Linux. A continuación se detalla el proceso.

Una vez descargadas la versión de las fuentes del núcleo[1] se tiene que preparar adecuadamente el entorno de compilación (`gcc`, `make`, `libc6-dev`, `autoconf`, `automake`...) pues sino pueden fallar los pasos siguientes si alguno de estos paquetes faltan.

1. Copiar el fichero `$NETKIT_HOME/kernel/netkit-kernel-config` al directorio donde se ha descomprimido el núcleo con el nombre ".config"

2. Aplicar los parches de Netkit (situarse en el directorio de las fuentes del núcleo descomprimido), normalmente situados en el subdirectorio `$NETKIT_HOME/kernel/patches`.

```
patch -p1 <patch_path_and_name.diff
```

[1] http://www.kernel.org/pub/linux/kernel/

3. Una vez hecho esto, el primer paso es configurar el núcleo. La configuración del núcleo significa elegir que características deben ser habilitadas en el núcleo compilado, qué sistemas de ficheros serán soportados, qué *drivers* de dispositivos serán incluidos, etc. Para ello se utilizan los siguientes comandos:

   `make oldconfig ARCH=um` o `make oldconfig ARCH=um SUBARCH=i386`

   Este último si la máquina es de 64 bits. Sirve para actualizar una configuración anterior del núcleo a una nueva versión de éste, formulando solamente las preguntas relativas a las nuevas características.

   ```
   make menuconfig ARCH=um
   ```

   Para establecer una configuración personalizada diferente a la existente en el archivo `netkit-kernel-config`. Aquí es donde se han activado las configuraciones para múltiples tablas `IP_MULTIPLES_TABLES` e `IPv6_MULTIPLES_TABLES`, ubicadas respectivamente dentro del directorio `Networking Support/Networking options` en los subdirectorios

   ```
   TCP/IP Networking/IP: Policy Routing
   The Ipv6 protocol/Ipv6: Multiple Routing Tables
   ```

4. Una vez que se ha terminado de configurar el núcleo, se compila con sus módulos. El núcleo que aparece con el nombre de `linux`, se copia el directorio `$NETKIT_HOME/kernel`, y los módulos se instalan en una ubicación dónde Netkit pueda encontrarlos, en particular, Netkit buscará los módulos en el subdirectorio llamado `modules` del directorio donde se ha instalado Netkit.

   ```
   make all modules_install ARCH=um INSTALL_MOD_PATH=
   $NETKIT_HOME/kernel/modules
   ```

5. Por último, falta indicar a Netkit qué núcleo se va a utilizar, pues los módulos ya están en el directorio adecuado. Una opción es mediante un enlace simbólico:

   ```
   ln -sf linux netkit-kernel
   ```

# Apéndice J

# Multihoming: configuración

## J.1. lab.conf

```
LAB_DESCRIPTION="Configuración de un escenario multihoming
con multiples tablas"
LAB_VERSION=2.0
LAB_AUTHOR="Mercedes Bernal Pérez"
LAB_EMAIL=mercedesbernalperez@gmail.com

as20r1[0]="C"
as20r1[1]="B"

as20r2[0]="E"
as20r2[1]="D"

as200r1[0]="C"
as200r1[1]="E"
as200r1[2]="F"

as100r1[0]="B"
as100r1[1]="A"
as100r1[2]="D"
```

## J.2. as20r1.startup

```
/sbin/ifconfig eth0 210.1.2.1 up
/sbin/ifconfig eth1 193.10.11.1 up

cat > /etc/quagga/bgpd.conf <<EOF
hostname as1
password zebra
log file /var/log/quagga/bgp1.log
router bgp 1
  bgp router-id 193.10.11.1
  neighbor 210.1.2.2 remote-as 4
  neighbor 193.10.11.2 remote-as 2
EOF

cat > /etc/quagga/zebra.conf <<EOF
hostname router1
password zebra
log file /var/log/quagga/zebra.log
EOF

cat > /etc/default/bgpd <<EOF
OPTIONS=""
EOF

/etc/init.d/quagga start
```

## J.3. as20r2.startup

```
/sbin/ifconfig eth0 120.1.2.2 up
/sbin/ifconfig eth1 100.1.2.2 up

cat > /etc/quagga/bgpd.conf <<EOF
hostname bgp3
password zebra
log file /var/log/quagga/bgp3.log
router bgp 3
```

```
  bgp router-id 100.1.2.2
  neighbor 120.1.2.1 remote-as 4
  neighbor 100.1.2.1 remote-as 2
EOF

cat > /etc/quagga/zebra.conf <<EOF
hostname router1
password zebra
log file /var/log/quagga/zebra.log
EOF

cat > /etc/default/bgpd <<EOF
OPTIONS="-t 10 -p 2002"
EOF

/etc/init.d/quagga start
```

## J.4. as100r1.startup

```
/sbin/ifconfig eth0 193.10.11.2 up
/sbin/ifconfig eth1 195.11.14.1 up
/sbin/ifconfig eth2 100.1.2.1 up

cat > /etc/quagga/bgpd.conf <<EOF
hostname bgp2.1-1
password zebra
log file /var/log/quagga/bgp2_1_1.log
router bgp 2
  bgp router-id 193.10.11.2
  network 195.11.14.0/24
  neighbor 193.10.11.1 remote-as 1
EOF

cat > /etc/quagga/bgp2.conf <<EOF
hostname bgp2.1-2
password zebra
```

```
log file /var/log/quagga/bgp2_1_2.log
router bgp 2
  bgp router-id 100.1.2.1
  network 195.11.14.0/24
  neighbor 100.1.2.2 remote-as 3
  neighbor 100.1.2.2 port 2002
EOF

cat > /etc/quagga/zebra.conf <<EOF
hostname router2-1
password zebra
log file /var/log/quagga/zebra.log
EOF

cat > /etc/default/bgpd <<EOF
OPTIONS=""
OPTIONS2="-t 10 -p 2002 -P 20002 -i /var/run/quagga/bgp2.pid
-f /etc/quagga/bgp2.conf"
EOF

/etc/init.d/quagga start
```

## J.5. as200r1.startup

```
/sbin/ifconfig eth0 210.1.2.2 up
/sbin/ifconfig eth1 120.1.2.1 up
/sbin/ifconfig eth2 200.1.1.1 up

cat > /etc/quagga/bgpd.conf <<EOF
hostname bgp2.2-1
password zebra
log file /var/log/quagga/bgp2_2_1.log
router bgp 4
  bgp router-id 210.1.2.2
  network 200.1.1.0/24
  neighbor 210.1.2.1 remote-as 1
```

```
EOF

cat > /etc/quagga/bgp2.conf <<EOF
hostname bgp2.2-2
password zebra
log file /var/log/quagga/bgp2_2_2.log
router bgp 4
  bgp router-id 120.1.2.1
  network 200.1.1.0/24
  neighbor 120.1.2.2 remote-as 3
  neighbor 120.1.2.2 port 2002
EOF

cat > /etc/quagga/zebra.conf <<EOF
hostname router2-2
password zebra
log file /var/log/quagga/zebra.log
EOF

cat > /etc/default/bgpd <<EOF
OPTIONS=""
OPTIONS2="-t 10 -p 2002 -P 20002 -i /var/run/quagga/bgp2.pid
-f /etc/quagga/bgp2.conf"
EOF


/etc/init.d/quagga start
```

## J.6. /etc/init.d/quagga

```
#!/bin/bash

unset DAEMONS

[ -f /etc/default/quagga ] && . /etc/default/quagga
```

```
if [ -z "$DAEMONS" ]; then
echo "Configure /etc/defaults/[quagga|zebra|bgpd|...] correctly"
exit 1
fi

case "$1" in
    start)
for d in $DAEMONS
do
    unset OPTIONS OPTIONS2
    [ -f /etc/default/$d ] && . /etc/default/$d
    echo "/usr/sbin/$d $OPTIONS -d" && /usr/sbin/$d $OPTIONS -d
    [ -n "$OPTIONS2" ] && echo "/usr/sbin/$d $OPTIONS2 -d" &&
    /usr/sbin/$d $OPTIONS2 -d
done
    ;;
    stop)
killall $DAEMONS
    ;;
    help|*)
echo "$0 [start|stop|help]"
echo "Configure /etc/defaults/quagga, /etc/defaults/quagga,
/etc/quagga/bgpd, etc"
    ;;
esac

unset DAEMONS OPTIONS OPTIONS2

exit 0
```

# Acrónimos

| | |
|---|---|
| **AS** | Autonomous System |
| **ASs** | Autonomous Systems |
| **EGP** | Exterior Gateway Protocol |
| **IETF** | Internet Engineering Task Force |
| **IGP** | Interior Gateway Protocol |
| **IP** | Internet Protocol |
| **ISP** | Internet Service Provider |
| **IXP** | Internet Exchange Point |
| **LINX** | London Internet Exchange |
| **EspaNIX** | España Internet Exchange |
| **CatNIX** | Cataluña Internet Exchange |
| **MED** | Multi-Exit Discriminator |
| **QoS** | Quality of Service |
| **RIR** | Regional Internet Registry |
| **RR** | Route Reflector |
| **PBR** | Policy Based Routing |
| **SPP** | Stable Paths Problem |

| | |
|---|---|
| **GPL** | General Public License |
| **UML** | User Mode Linux |
| **STP** | Spanning Tree Protocol |
| **PPP** | Point to Point Protocol |
| **LDP** | Label Distribution Protocol |
| **DNS** | Domain Name System |
| **SMTP** | Simple Mail Transfer Protocol |
| **POP** | Post Office Protocol |
| **IMAP** | Internet Message Access Protocol |
| **FTP** | File Transfer Protocol |
| **TFTP** | Trivial file transfer Protocol |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **NFS** | Network File System |
| **SSH** | Secure SHell |
| **PIM** | Protocol Independent Multicast |
| **PIM-SM** | Protocol Independent Multicast - Sparse Mode |
| **IGMP** | Internet Group Management Protocol |
| **MLD** | Multicast Listener Discovery |
| **IKE** | Internet Key Exchange |
| **RADIUS** | Remote Authentication Dial-In User Server |
| **IPSec** | Internet Protocol Security |
| **GRE** | Generic Routing Encapsulation |

| | |
|---|---|
| **NAT** | Network Address Translation |
| **SNMP** | Simple Network Management Protocol |
| **CIDR** | Classless Inter-Domain Routing |
| **FIB** | Forwarding Information Base |
| **MRT** | Multi-threaded Routing Toolkit |
| **RRC** | Remote Route Collector |
| **PI** | Provider Independent |
| **PA** | Provider Aggregatable |
| **TE** | Traffic Engineering |
| **BGP-4** | Border Gateway Protocol |
| **OSPF** | Open Shortest Path First |
| **RIP** | Routing Information Protocol |
| **ISIS** | Intermediate System to Intermediate System |
| **MPLS** | Multiprotocol Label Switching |
| **TCP** | Transmission Control Protocol |
| **UDP** | User Datagram Protocol |
| **DHCP** | Dynamic Host Configuration Protocol |
| **eBGP** | Exterior BGP-4 |
| **iBGP** | Interior BGP-4 |
| **RIPE** | Réseaux IP Européens |
| **RIPE RIS** | RIPE's Routing Information Service |

# Bibliografía


[1] Debian Binary Package Building HOWTO.

[2] Graphviz Graph Visualization Software.

[3] RIPE RIS - libbgpdump. http://www.ris.ripe.net/source/libbgpdump-1.4.99.11.tar.gz.

[4] Using BGP Community Values to Control Routing Policy in an Upstream Provider Network.

[5] Wikipedia Web site.

[6] YouTube Hijacking: A RIPE NCC RIS case study. http://www.ripe.net/news/study-youtube-hijacking.html.

[7] J. Abley, K. Lindqvist, E. Davies, B. Black, and V. Gill. IPv4 Multihoming Practices and Limitations. RFC 4116 (Informational), July 2005.

[8] Pedro A. Aranda-Gutiérrez. Detection of Trial and Error Traffic Engineering with BGP-4. In *The Fifth International Conference on Networking and Services; ICNS 2009*. INRIA, April 2009.

[9] Pedro A. Aranda-Gutiérrez. Inter-domain Routing and Traffic Engineering. FISS, 2009.

[10] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and Principles of Internet Traffic Engineering. RFC 3272 (Informational), May 2002. Updated by RFC 5462.

[11] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for Traffic Engineering Over MPLS. RFC 2702 (Informational), September 1999.

[12] Daniel O. Awduche and Uunet (mci Worldcom. Mpls and traffic engineering in ip networks. *IEEE Communications Magazine*, 37:42–47, 1999.

[13] S. Bartholomew. The art of peering. *BT Technology Journal*, 18(3):33–39, 2000.

[14] T. Bates and R. Chandra. BGP Route Reflection An alternative to full mesh IBGP. RFC 1966 (Experimental), June 1996. Obsoleted by RFC 4456, updated by RFC 2796.

[15] T. Bates and Y. Rekhter. Scalable Support for Multi-homed Multi-provider Connectivity. RFC 2260 (Informational), January 1998.

[16] O. Bonaventure, S. De Cnodder, J. Haas, B. Quoitin, and R. White. Controlling the redistribution of bgp routes, April 2003. Work in progress, draft-ietf-grow-bgp-redistribution-00.txt.

[17] Andre Broido, Evi Nemeth, and Kc Claffy. Internet expansion, refinement and churn, 2002.

[18] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376 (Proposed Standard), October 2002. Updated by RFC 4604.

[19] E. Chen and J. Stewart. A Framework for Inter-Domain Route Aggregation. RFC 2519 (Informational), February 1999.

[20] Cisco Systems. *BGP Best Path Selection Algorithm.*

[21] Inc Cisco Systems. *Cisco IOS IP Routing: BGP Command Reference.* Cisco Press, 2010.

[22] R. Coltun, D. Ferguson, and J. Moy. OSPF for IPv6. RFC 2740 (Proposed Standard), December 1999. Obsoleted by RFC 5340.

[23] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational), May 1996.

[24] P. Deutsch. GZIP file format specification version 4.3. RFC 1952 (Informational), May 1996.

[25] P. Deutsch and J-L. Gailly. ZLIB Compressed Data Format Specification version 3.3. RFC 1950 (Informational), May 1996.

[26] Jeff Dike. A user-mode port of the linux kernel. In *ALS'00: Proceedings of the 4th annual Linux Showcase & Conference*, pages 7–7, Berkeley, CA, USA, 2000. USENIX Association.

[27] Jeff Dike. User mode linux. Prentice Hall, 2006.

[28] Di fa Chang, Ramesh Govindan, and John Heidemann. An empirical study of router response to large bgp routing table load. In *in Proc. Internet Measurement Workshop*, pages 203–208. ACM Press, 2002.

[29] Nick Feamster, Jay Borkenhagen, and Jennifer Rexford. Guidelines for interdomain traffic engineering. *ACM SIGCOMM Computer Communication Review*, 33:2003, 2003.

[30] A Feldmann, A Greenberg, C Lund, N Reingold, and J Rexford. NetScope: Traffic Engineering for IP Networks. *IEEE Network Magazine*, 2000.

[31] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC 4601 (Proposed Standard), August 2006. Updated by RFC 5059.

[32] Bernard Fortz. Internet traffic engineering by optimizing ospf weights. In *in Proc. IEEE INFOCOM*, pages 519–528, 2000.

[33] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional ip routing protocols. *IEEE Communications Magazine*, 40:118–124, 2002.

[34] Free Software Foundation. GNU Web site. `http://www.gnu.org/licenses/gpl.html`.

[35] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632 (Best Current Practice), August 2006.

[36] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519 (Proposed Standard), September 1993. Obsoleted by RFC 4632.

[37] Lixin Gao. Stable internet routing without global coordination. In *IEEE/ACM Transactions on Networking*, pages 681–692, 2000.

[38] Geoff Huston. POTAROO Web site. `http://www.potaroo.net/`. Last visit, 18-Dec-2009.

[39] Geoff Huston. The 16-bit AS Number Report. `http://www.potaroo.net/tools/asn16/`. Last visit, 13-Aug-2009.

[40] Geoff Huston. The 32-bit AS Number Report. `http://www.potaroo.net/tools/asn32/`. Last visit, 25-Jul-2010.

[41] Geoff Huston. Università degli Studi Roma Tre Web site. `http://www.uniroma3.it/`.

[42] R Govindan and A Reddy. An analysis of Internet inter-domain topology and route stability. In *in Proc. IEEE INFOCOM*, 1997.

[43] T. Griffin and G. Huston. BGP Wedgies. RFC 4264 (Informational), November 2005.

[44] T G Griffin and G Wilfong. An analysis of BGP convergence properties. In *In Proc. of SIGCOMM'99*, pages 277–288. ACM Press, 1999.

[45] Timothy G. Griffin. An experimental analysis of bgp convergence time. In *In Proceedings of ICNP*, pages 53–61, 2001.

[46] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Transactions on Networking*, 10:232–243, 2002.

[47] P. Gross. Choosing a Common IGP for the IP Internet. RFC 1371 (Informational), October 1992.

[48] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930 (Best Current Practice), March 1996.

[49] C.L. Hedrick. Routing Information Protocol. RFC 1058 (Historic), June 1988. Updated by RFCs 1388, 1723.

[50] Iljitsch van Beijnum. BGP, Chapter 6,Traffic Engineering. `http://oreilly.com/catalog/bgp/chapter/ch06.html`. Last visit: 13-Jan-2010.

[51] ISC. Internet Domain Survey.

[52] Juniper Networks. *Selecting the Best Path.*

[53] Costas Kalogiros, Marcelo Bagnulo, and Alexandros Kostopoulos. Understanding incentives for prefix aggregation in bgp. In *ReArch '09: Proceedings of the 2009 workshop on Re-architecting the internet*, pages 49–54, New York, NY, USA, 2009. ACM.

[54] Kunihiro Ishiguro. GNU Zebra Routing Software.

[55] C. Labovitz L. Blunk, M. Karir. MRT routing information export format. Technical report.

[56] C. Labovitz. Mrt programmer's guide, 1999.

[57] C Labovitz, A Ahuja, A Bose, and F Jahanian. Delayed internet routing convergence. In *in SIGCOMM, 2000*, pages 175–187, 2000.

[58] C Labovitz, A Ahuja, and F Jahanian. Experimental Study of Internet Stability and Wide-Area Network Failures. In *in Proc. International Symposium on Fault-Tolerant Computing*, 1999.

[59] C Labovitz, R Malan, and F Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 1998.

[60] C Labovitz, R Malan, and F Jahanian. Origins of Internet routing instability. In *in INFOCOM*, 1999.

[61] T. Li, Fernando, and J. Abley. The AS_PATHLIMIT Path Attribute. Technical report, January 2007. Expired: July 8, 2007.

[62] D. MacKenzie, P. Eggert, , and R. Stallman. Comparing and merging files with gnu diff and patch, 2003.

[63] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. In *In Proc. ACM SIGCOMM*, pages 3–16, 2002.

[64] G. Malkin. RIP Version 2. RFC 2453 (Standard), November 1998. Updated by RFC 4822.

[65] G. Malkin and R. Minnear. RIPng for IPv6. RFC 2080 (Proposed Standard), January 1997.

[66] P. Marques and F. Dupont. Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing. RFC 2545 (Proposed Standard), March 1999.

[67] Massimo Rimondini. Emulating Computer Networks with Netkit. `http://www.dia.uniroma3.it/~compunet`, 2007.

[68] D. McPherson, V. Gill, D. Walton, and A. Retana. Border Gateway Protocol (BGP) Persistent Route Oscillation Condition. RFC 3345 (Informational), August 2002.

[69] J. Moy. OSPF Version 2. RFC 2328 (Standard), April 1998.

[70] D. Oran. OSI IS-IS Intra-domain Routing Protocol. RFC 1142 (Informational), February 1990.

[71] Paolo Giarrusso. UML utilities.

[72] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[73] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[74] Bruno Quoitin. Bgp-based interdomain traffic engineering for transit ass. 2006.

[75] Bruno Quoitin, Steve Uhlig, Cristel Pelsser, C. Pelsser, Louis Swinnen, and Olivier Bonaventure. Interdomain traffic engineering with bgp. *IEEE Communications Magazine*, 41, 2003.

[76] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006.

[77] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996.

[78] RIPE. RIS Raw Data. Last visit, 16-Dec-2009.

[79] RIPE. Autonomous system (as) number assignment policies and procedures. Technical report, 2009.

[80] E.C. Rosen. Exterior Gateway Protocol (EGP). RFC 827, October 1982. Updated by RFC 904.

[81] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. Linux Netlink as an IP Services Protocol. RFC 3549 (Informational), July 2003.

[82] S. Sangli, E. Chen, R. Fernando, J. Scudder, and Y. Rekhter. Graceful Restart Mechanism for BGP. RFC 4724 (Proposed Standard), January 2007.

[83] Aman Shaikh, Lampros Kalampoukas, Rohit Dube, and Anujan Varma. Routing stability in congested networks: Experimentation and analysis. In *In Proc. of SIGCOMM'00*, pages 163–174. ACM Press, 2000.

[84] Philip Smith. Bgp multihoming techniques, 2007.

[85] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), January 2001.

[86] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Y H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *In Proc. IEEE INFOCOM*, 2002.

[87] Lakshminarayanan Subramanian, VenkataÑ. Padmanabhan, and Randy H. Katz. Geographic properties of internet routing. In *In USENIX Annual Technical Conference*, pages 243–259, 2002.

[88] Renata Teixeira, Aman Shaikh, Tim Griffin, and Jennifer Rexford. Dynamics of hot-potato routing in ip networks, 2004.

[89] P. Traina. Autonomous System Confederations for BGP. RFC 1965 (Experimental), June 1996. Obsoleted by RFC 3065.

[90] Steve Uhlig and Olivier Bonaventure. Designing BGP-based outbound traffic engineering techniques for stub ASes. *Comput. Commun. Rev*, 34, 2004.

[91] Steve Uhlig, Olivier Bonaventure, and Bruno Quoitin. Interdomain Traffic Engineering with minimal BGP configurations, 2003.

[92] Steve Uhlig and Bruno Quoitin. Tweak-it: Bgp-based interdomain traffic engineering for transit ass. 2005.

[93] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent Route Oscillations in Inter-Domain Routing. Technical report, Computer Networks, 1999.

[94] R. Vida and L. Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810 (Proposed Standard), June 2004. Updated by RFC 4604.

[95] Q. Vohra and E. Chen. BGP Support for Four-octet AS Number Space. RFC 4893bis (Internet draft), 2009.

[96] XFree86, Inc. Bsd license.

Yo, Mercedes Bernal Pérez, a través de la presente declaración autorizo a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.