UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Emulador de redes CORE

Jose María Valero Muñoz

Tutor: Jose Óscar Romero Martínez

Cotutor: Antonio León Fernández

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2020-21

Valencia, 11 de diciembre de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV-000030

Resumen

Mediante la emulación de redes, se puede replicar un escenario de red real de forma totalmente virtual. Esto otorga una serie de beneficios, ya que sería posible por ejemplo asegurar el correcto funcionamiento de una aplicación antes de llevarla a la práctica, minimizando por lo tanto los riesgos financieros. En este Trabajo de Final de Grado se ha analizado el emulador de redes de código abierto CORE, mostrando cada una de sus posibilidades así como de los servicios que ofrece. A continuación se han configurado algunos de estos servicios, poniendo especial énfasis en los que han sido estudiados en el grado. Para comprobar el correcto funcionamiento de dichos servicios, se ha utilizado el analizador de tráfico de red Wireshark.

Resum

Mitjançant l'emulació de xarxes, es pot replicar un escenari d'una xarxa real de forma totalment virtual. Això suposa una serie de beneficis, ja que seria possible per example assegurar el funcionament correcte d'una aplicació abans de dur-la a la pràctica, minimizant d'aquesta manera els riscs econòmics. En aquest Treball de Final de Grau s'ha analitzat l'emulador de xarxes de codi obert CORE, mostrant cadascuna de les seues possibilitats, així com els servicis que ofereix. A continuació s'han configurat alguns d'aquests servicis, posant especial èmfasi en aquells que han sigut estudiats al grau. Amb tal de comprovar el correcte funcionament dels servicis, s'ha utilitzat l'analitzador de tràfic de xarxa Wireshark.

Abstract

A real network scenario can be replicated, through virtualization, using network emulation. This provides a number of benefits, since for example it would be possible to ensure that an application shows a correct behavior before putting it into practice, minimizing financial risks this way. In this Final Degree Project, the open source network emulator CORE has been analized, showing each of its possibilities as well as the services it offers. Furthermore, some of these services have been configured, making a special emphasis on those that have been studied in the degree. To check for the correct operation of these services, the network traffic analyzer tool Wireshark has been used.

Índice general

Índice de figuras

Capítulo 1

Introducción y objetivos

1.1. Introducción

Hoy en día se puede disfrutar de una innumerable cantidad de servicios y aplicaciones a través de la red, como por ejemplo servicios de compra por internet, aplicaciones que permiten la comunicación de manera rápida y efectiva sin importar la distancia a la que se encuentren los usuarios, o la enseñanza por parte de escuelas y universidades, u otras entidades, a través de recursos alojados en la red.

Estos servicios tienen como objetivo ofrecer mejoras y facilidades en la calidad de vida de las personas, y por lo tanto no es de extrañar que ganen popularidad. Dicha popularidad, junto con un ambiente competitivo causa que el número de servicios esté siempre en constante aumento, ofreciendo un gran número de alternativas que tratarán de satisfacer las necesidades de los usuarios. Por este motivo, una empresa que quiera mantener su relevancia en este mercado tan competitivo se verá en la necesidad de ofrecer nuevos servicios que capten la atención de los usuarios, o de expandir las infraestructuras de red de sus actuales servicios y aplicaciones, de forma que la experiencia de uso se haga más placentera.

Realizar cambios en la infraestructura de red, además de aumentar la complejidad de la misma, suele venir acompañado de un número de riesgos: Posibles problemas de incompatibilidad con el resto de elementos de la red, errores en el diseño que no se detecten a tiempo, o la caída de los servicios que se ofrecen de forma temporal mientras se adapta el nuevo diseño. Estos riesgos se pueden mitigar mediante el uso de herramientas tales como simuladores o emuladores de red.

1.2. Objetivos

El principal objetivo de este Trabajo Fin de Grado es el estudio y análisis del emulador de redes CORE (*Common Open Research Emulator*), viendo las posibilidades que ofrece, así como la configuración de los protocolos de enrutamiento RIP (*Routing Information Protocol*), OSPF (*Open Shortest Path First*), y BGP (*Bridge Gateway Protocol*), y del servicio de VPN (*Virtual Private Network*), realizando los ajustes necesarios para que funcionen de manera apropiada. La explicación del funcionamiento de dichos protocolos se realiza con ayuda de *Wireshark*, que permitirá observar cómo se comportan con la red a nivel interno.

1.3. Estructura de la memoria

La memoria se divide en capítulos, que así mismo se dividen en subcapítulos. El contenido de dichos capítulos es el siguiente:

- Capítulo 1: En este capítulo se ha presentado el proyecto con una breve introducción y los objetivos que se tratan de abordar.
- Capítulo 2: El objetivo de este capítulo es el de explicar a grandes rasgos los conceptos de simulación y emulación de redes, así como de presentar la estructura general que siguen los emuladores de red.
- Capítulo 3: En este capítulo se presentan los protocolos y servicios en los que se va a centrar este TFG, y se muestra el funcionamiento teórico de los mismos.
- Capítulo 4: Este capítulo se centra más a fondo en CORE, analizando su estructura y detallando los distintos elementos de los que dispone.
- Capítulo 5: En este capítulo se despliegan las redes mediante la interfaz de CORE, y se analizan los diferentes protocolos de encaminamiento. Aquí es donde se muestra la parte más práctica del proyecto.
- Capítulo 6: En este último capítulo se extraen una serie de conclusiones, y se presentan unas ideas para posibles futuros proyectos.

1.4. Metodología

Para el desarrollo del proyecto se ha pasado por 3 fases.

La primera fase consiste en la recopilación de la información, principalmente acerca del funcionamiento de la emulación de redes, así como de las tecnologías que hacen esto posible. Para ello se han consultado múltiples recursos, contrastando la información obtenida de las fuentes.

La segunda fase consiste en la familiarización con el software CORE, tomando como referencia el manual de dicho software, y experimentando con las distintas posibilidades que ofrece.

En la tercera fase se diseñan diagramas de red sencillos, los cuales han sido configurados para permitir el funcionamiento de algunos de los servicios de CORE. A continuación se analizan los resultados obtenidos, poniendo en práctica los conocimientos adquiridos en las anteriores fases.

Capítulo 2

Emulación de redes

2.1. Introducción

Un emulador es un tipo de *software* o *hardware* que permite replicar el comportamiento de un componente o sistema de manera más o menos precisa, teniendo en cuenta que cuanto mayor sea la precisión del emulador, mayor será el consumo de recursos utilizado. El emulador permite obtener resultados que serán muy similares a los que podrían conseguirse en el sistema real. Un simulador, en cambio, imita el comportamiento del sistema, representándolo de forma abstracta y obteniendo unos resultados aproximados, pero no idénticos, a los que se obtendría en el sistema real. Conviene diferenciar estos dos términos ya que en muchas ocasiones se utilizan de forma similar, pero como se puede observar tienen significados distintos.

En el ámbito de las redes de ordenadores, tanto los simuladores como los emuladores tienen una finalidad distinta.

Un simulador de redes, por lo general, presenta un consumo de recursos y un número de funcionalidades menor al de un emulador, y por lo tanto se utiliza en entornos destinados a la educación, facilitando comprender el funcionamiento de los protocolos de red y de los distintos componentes que conforman una red, así como punto de partida en el diseño de una infraestructura de red.

Un emulador de redes, por otro lado, se utilizará principalmente en niveles avanzados del desarrollo de una aplicación, ya que al permitir cambiar el flujo de paquetes, introducir latencia, o generar pérdidas de paquetes, entre otras funcionalidades que modifican el comportamiento de la red, ofrece una visión más realista de los problemas que podría sufrir el servicio en un entorno real.

2.2. Herramientas Utilizadas

Algunos de los emuladores de red utilizan tecnologías de virtualización para formar los nodos, junto a *software* externo que añadirá funcionalidad adicional a los mismos.

En este apartado se detallarán algunos de los distintos elementos (*software*, *framework* o métodos de virtualización) que complementan a un emulador de redes, o bien que permiten su funcionamiento, y que aparecerán más adelante en posteriores capítulos de este trabajo.

2.2.1. Linux Containers

Linux Containers, o LXC, es un método de virtualización que permite generar distintos contenedores, aislados entre sí, en un único anfitrión. LXC es posible gracias a los grupos de control o *cgroups*, y a los espacios de nombres o *namespaces*, que son funcionalidades de las que dispone el *kernel* de Linux.

Por un lado, los grupos de control permiten gestionar el consumo de recursos por parte de un proceso o un conjunto de procesos. Los grupos de control, junto con los espacios de nombres, permitirán controlar el consumo de CPU o de memoria de los contenedores.

Por otro lado, los espacios de nombres permiten separar unos procesos de otros, de forma que estén totalmente aislados entre sí, y son, por lo tanto, una parte fundamental de los contenedores. Para ello, uno o varios procesos se asignarán a un espacio de nombres concreto. De esta forma, los procesos pertenecientes a dicho espacio de nombres no podrán interactuar de ningún modo con otros procesos que pertenezcan a un espacio de nombres distinto, pero sí que podrán interactuar entre ellos mismos.

Hay 8 tipos distintos de espacios de nombres, y cada uno permite aislar una serie de recursos determinados: *Mount*, UTS, IPC, PID, *network*, *user*, *cgroup* y *time*.

2.2.2. Linux Bridge

Un *Network Bridge* o Puente de Red es un dispositivo de nivel de enlace de datos que permite la conexión entre redes separadas, formando de este modo una única red. Un *Linux Bridge* es una implementación por medio de *software* de un Puente de Red, y permite realizar tareas de filtrado de tráfico gracias a herramientas como *ebtables*.

Para crear un *Linux Bridge* de forma manual, se puede utilizar la herramienta *ip* junto con el comando *bridge* del paquete *iproute2*.

2.2.3. Network Scheduler

El *Network Scheduler* se encarga de gestionar el tráfico de la red. Se utiliza en el control del tráfico de red para aumentar o reducir la congestión, o controlar la pérdida de paquetes. La herramienta de interés para este Trabajo Final de Grado es *tc*, la cual forma parte del paquete *iproute2*.

2.2.4. Quagga

Quagga es un *software* de encaminamiento de código abierto, pensado principalmente para los sistemas Linux, FreeBSD, NetBSD y Solaris, que ofrece la implementación de los protocolos RIPv1 y v2, OSPFv2 y v3, BGP4 o RIPng.

Quagga está formado por un demonio llamado zebra, que presenta la API *Zserv* a los distintos clientes. Estos clientes son a su vez demonios que se encargan de implementar los protocolos de red, y comunican las actualizaciones de las tablas de encaminamiento a zebra mediante *Zserv*. Algunos ejemplos de los clientes de Quagga son *ospfd*, que implementa OSPFv2, o *ripd*, que implementa RIP versión 1 y versión 2.

2.2.5. BIRD

BIRD (*BIRD Internet Routing Daemon*), es un *software de encaminamiento* de código abierto que funciona, entre otros, en sistemas operativos Linux y FreeBSD, y soporta tanto IP v4 como v6, OSPF, BGP y RIP. A diferencia de Quagga, que está separado en múltiples procesos, BIRD únicamente dispone de un proceso, compuesto por una o varias rutas de encaminamiento, y por los protocolos que soporta, que se conectarán a dichas tablas.

Aunque BIRD es un software capaz, en este Trabajo de Final de Grado se ha optado por utilizar Quagga.

2.2.6. Wireshark

Wireshark es un analizador de paquetes de red de código abierto. Este *software* permite al usuario configurar una interfaz de red en modo promiscuo, y de este modo captar la gran mayoría del tráfico que pasa por ella mediante la herramienta de captura de la que dispone. Cabe destacar que se pueden utilizar filtros de captura para obtener únicamente el tipo de tramas de interés. Una vez capturado, el tráfico podrá ser analizado para detectar posibles anomalías que puedan ocurrir en la red.

Además de la detección de problemas en la red, wireshark puede utilizarse también para observar el comportamiento de los diferentes elementos de la red, o para realizar mediciones del tráfico.

Capítulo 3

Protocolos y Servicios

3.1. RIP

El protocolo de información de encaminamiento, o RIP (*Routing Information Protocol*), es un protocolo de puerta de enlace interior (IGP) de tipo vector-distancia que utiliza el número de saltos para determinar el mejor camino por el que encaminar el tráfico.

Los protocolos de puerta de enlace interior, como es el caso de RIP, se utilizan para el encaminamiento dentro de un sistema autónomo. Se denomina sistema autónomo al conjunto de redes y routers que siguen una misma política de encaminamiento.

El número de saltos equivale al número de routers que hay entre la dirección de origen y de destino. RIP pone un límite de 15 saltos antes de considerar la red inalcanzable, de este modo se previenen los bucles de encaminamiento. Además, utiliza el protocolo de transporte UDP con número de puerto 520, y tiene un valor de distancia administrativa de 120.

La distancia administrativa define la prioridad que se le da a un protocolo. Un protocolo con menor distancia administrativa será escogido antes que uno con una distancia administrativa mayor.

Existen tres versiones de RIP: RIP versión 1, RIP versión 2 y RIPNG. RIPNG soporta redes IPv6, mientras que la versión 2 difiere de la versión 1 en que envía las actualizaciones por *multicast*, frente a las actualizaciones por *broadcast* de la versión 1. Además, la versión 2 envía la información de la máscara de subred en sus actualizaciones. En este Trabajo Final de Grado se ha empleado la versión 2 de RIP.

RIP funciona mediante el uso de mecanismos de temporización, la RFC 1058 define 3 de ellos:

- Temporizador de actualizaciones (*Update Timer*): El tiempo por defecto es de 30 segundos, y establece la frecuencia con la que los routers comparten la información de sus tablas de encaminamiento con otros vecinos.
- Temporizador de expiración (*Invalid Timer*): El valor por defecto es de 180. Si después de 180 segundos no se ha producido ninguna actualización en una entrada de la tabla, se actualiza el número de saltos de dicha entrada con 16, marcándola como inalcanzable.
- Temporizador de purga (*Flush Timer*): El tiempo de este temporizador es de 240 segundos, 60 segundos superior al del anterior temporizador. Una vez venza este temporizador, si una

entrada no ha recibido actualizaciones en todo ese tiempo, se eliminará de la tabla. Su valor debe ser superior al del temporizador de expiración.

- Existe un cuarto temporizador, el temporizador de espera (*Hold-Down Timer*), con un valor de 180 segundos por defecto, que se pone en marcha cuando un vecino envía una actualización donde se marca una ruta como inalcanzable. Si se recibe una actualización del vecino donde se indica que la ruta vuelve a estar disponible, o se recibe una actualización con una ruta mejor a la original, la entrada se actualiza y el temporizador se detiene.

Al producirse errores en la red, como podría ser la caída de un enlace, es posible que se produzcan bucles de encaminamiento entre un grupo de routers si no hay ningún mecanismo que impida que esto ocurra. Con tal de evitar los bucles de encaminamiento, así como de acelerar la convergencia del protocolo, RIP implementa el horizonte dividido (*Split Horizon*), las rutas envenenadas (*Route Poisoning*), y las actualizaciones inmediatas (*Triggered Updates*).

- Horizonte dividido: Este mecanismo impide que se envíe a un router vecino las entradas de la tabla que se han aprendido de dicho router. De este modo se previenen los bucles de encaminamiento ya que se impedirá que dos routers se envíen la misma entrada de forma recursiva en el caso de que un enlace conectado a uno de ellos esté caído.
- Rutas envenenadas: Cuando se desconecta un enlace, este mecanismo permite que RIP envíe a todos sus vecinos una métrica de 16 (inalcanzable) para la red que se ha desconectado.
- Actualizaciones inmediatas: Permite el envío de mensajes de actualización cuando se detecta un cambio en la topología de la red. Este mecanismo evita el tener que esperar 30 segundos entre mensajes de actualización y por lo tanto reduce el tiempo de convergencia.

RIP es un protocolo sencillo de implementar, y tiene un reducido consumo de procesador y de memoria. No obstante, tiene problemas como la convergencia lenta o los bucles de encaminamiento, aunque estos pueden ser mitigados gracias a la ayuda de los mecanismos que han sido mencionados.

3.2. OSPF

Open Shortest Path First, u OSPF, es un protocolo de puerta de enlace interior (IGP) al igual que RIP, aunque a diferencia de este, que era de tipo vector-distancia, OSPF es de tipo estado de enlace. Este protocolo utiliza el algoritmo de *Dijkstra* para realizar el descubrimiento de la red, y a diferencia de RIP, que utilizaba temporizadores, las actualizaciones de OSPF se desencadenan por eventos, haciendo que no sea susceptible a los bucles de encaminamiento. Sin embargo, el consumo de recursos es superior al de RIP, y también es más complejo de configurar. Su valor de distancia administrativa es de 110 y se utilizan principalmente dos versiones, OSPF versión 2 y versión 3, siendo la versión 3 compatible con IPv6. En este TFG se emplea la versión 2 del protocolo.

OSPF utiliza el coste del enlace como métrica. Para calcular el coste, se divide el valor de referencia de OSPF de 100 Mbps entre el ancho de banda del enlace. Por ejemplo, para un enlace de 10Mbps, el coste sería 100Mbps / 10Mbps = 10. Para enlaces con un ancho de banda de 100Mbps o superior, el coste será de 1.

Se emplea el concepto de áreas dentro de un sistema autónomo para el diseño de una red basada en OSPF. Las áreas tienen un identificador númerico y una de ellas será conocida como área 0 o *backbone*. La peculiaridad de este área es que el resto de áreas deben estar conectadas a ella, y para realizar la conexión entre distintas áreas se emplea un router de borde de área (ABR). El ABR conecta una o varias áreas al *backbone*, y forma parte de todas las áreas a las que se conecta.

El router de frontera del sistema autónomo (ASBR) se utiliza para interconectar sistemas autónomos distintos. Este tipo de router utilizará un protocolo de puerta de enlace exterior (EGP), como puede ser BGP, para compartir información entre diferentes sistemas autónomos.

Finalmente, en OSPF se utilizan los términos de Router designado (DR) y router designado de respaldo (BDR) para dos de los routers que forman parte de un enlace multiacceso (si el enlace es punto a punto entre dos routers no se elegirá un DR), y sirven para minimizar el número de adyacencias formadas y como eje central para el intercambio de la información de encaminamiento. Para elegir el router designado, se tiene en cuenta la prioridad de cada router, o el ID mayor en el caso de que la prioridad sea la misma.

En cuanto al funcionamiento del protocolo, en primer lugar se establecen adyacencias entre routers vecinos, y una vez se han establecido las adyacencias, se procede a la sincronización de las bases de datos entre los distintos routers. Estas bases de datos contienen los estados de los enlaces.

Para la formación de adyacencias entre routers vecinos, se envían mensajes HELLO a intervalos regulares. Estos paquetes sirven en primer lugar para descubrir a los routers vecinos, y en segundo lugar para establecer una adyacencia al recibir un paquete HELLO de un router al que se le había enviado un HELLO anteriormente. Este paquete se utiliza también para comprobar que los vecinos sigan activos.

A continuación se sincronizan las bases de datos de estados de enlaces de los routers vecinos mediante el envío de *Link State Advertisements* (LSA). Los LSA se emiten cuando hay un cambio en la topología de la red y existen 4 tipos:

- Router Link (RL): Describen el estado de las interfaces y los vecinos de cada router.
- Network Link (NL): El router designado recoge los *Router Links* enviados y genera un resumen con esta información que envía al resto del área.
- Summary Link (SL). Generados por el ABR, sirven para proporcionar información de un área al resto del sistema autónomo.
- External Link (EL). Generados por el ASBR.

Como se ha mencionado, el comportamiento de OSPF varía si los routers se encuentran en una red multiacceso o en una red punto a punto. En el caso de las redes multiacceso, se elige un DR y un BDR y se utilizan direcciones *multicast*. En el caso de las redes punto a punto, se utilizan direcciones *unicast* y se omite el paso de la elección del DR.

3.3. BGP

El protocolo de puerta de enlace de frontera, o BGP (*Border Gateway Protocol*), es un protocolo utilizado para intercambiar información de encaminamiento entre distintos sistemas autónomos

(AS). Se utilizan los routers de frontera de área para este fin, y por lo tanto estos tienen que ser compatibles con BGP. Este protocolo permite compartir información ya sea tanto dentro de un AS, donde se empleará una sesión interna de BGP (iBGP), como entre AS distintos, donde el tipo de sesión de BGP será externa (eBGP). A diferencia de RIP y OSPF, que utilizan UDP como protocolo de transporte, BGP utiliza TCP con el puerto 179.

En OSPF, el descubrimiento de vecinos se realizaba de manera automática gracias al protocolo HELLO, pero en BGP, los vecinos (también conocidos como *peers*), se deben establecer de forma manual.

BGP pasa por una serie de estados hasta que finaliza la adyacencia con un *peer*, que se pueden resumir a continuación:

- Idle: Este es el estado inicial. Cuando se detecta un evento de inicio, se intenta establecer una conexión TCP con un *peer*.
- Connect: Se realiza un *handshake* de tres pasos (*3-way handshake*) donde se envía un paquete TCP al *peer*, este responde y finalmente el primer router envía un ACK. Cuando se completa la conexión con éxito, se envía un mensaje OPEN y se avanza al estado OpenSent. Si se producen errores en la conexión y vence el temporizador de intento de conexión, se avanza al estado Active.
- Active: En este estado se intenta nuevamente un *handshake* de tres pasos. Si se completa con éxito, se avanza al estado OpenSent, pero si se producen errores en el establecimiento de la conexión TCP, se cambia al estado Connect de nuevo.
- OpenSent: Una vez se ha enviado el mensaje OPEN, el router espera a recibir un mensaje OPEN por parte de su *peer*. Cuando se recibe, se comparan sus versiones de BGP para ver si son compatibles. Además, la dirección IP de origen que contiene el mensaje OPEN debe coincidir con la dirección IP que está configurada para el *peer*. Cuando la comparación se realiza sin errores, se negocia un tiempo de espera entre los *peers* y se envía un mensaje KEEPALIVE y, a continuación, se avanza al estado OpenConfirm. Si por el contrario se encuentran errores en el mensaje OPEN, se envía un mensaje de notificación y se pasa nuevamente al estado Idle.
- OpenConfirm: En este estado simplemente se espera a la recipción de un mensaje KEEPALIVE. Una vez se ha recibido, se pasa el estado Established. Si expira el tiempo de espera, se recibe un mensaje de notificación y se pasa al estado Idle.
- Established: Se ha finalizado el establecimiento de la sesión BGP. En este estado se intercambia información de encaminamiento mediante mensajes UPDATE. Los mensajes de UPDATE se envían cada vez que se determina una ruta mejor para un destino, o cuando se produzcan cambios en una ruta. Además, se envían periódicamente mensajes KEEPALIVE para mantener la comunicación. Si expira el tiempo de espera, se detecta un error y se vuelve al estado Idle.

RIP utiliza métricas de número de saltos, y OSPF utiliza el coste del enlace, calculado a partir de su ancho de banda. A diferencia de estos, BGP dispone de una serie de atributos que ayudan a tomar la decisión de cuál es la mejor ruta a escoger, y son los siguientes:

- Origin: Incluye información acerca del origen del prefijo IP aprendido. Puede tomar los valores IGP (0) si el prefijo se aprendió a través de un protocolo interno al AS como OSPF, EGP (1) si por el contrario se aprendió mediante un protocolo externo al AS como BGP, o INCOMPLETE (2) si no se cumple ninguno de los dos casos anteriores.

- AS-Path: Indica los AS por los que ha ido pasando el anuncio de un prefijo. Cuando un router de frontera envía un anuncio de un prefijo a otro router de un AS distinto, dicho anuncio incluye el número de AS al que pertenece el primer router. Las entradas con una cadena de AS-Path menor se elegirán antes que las que tengan una cadena más larga.

- Next-Hop: Contiene la dirección IP del router correspondiente al siguiente salto hacia un destino. Para que la ruta se considere activa, el router debe poder acceder a la dirección especificada en este campo.

- Multiple-Exit-Discriminator (MED): Indica si desde un sistema autónomo hay múltiples enlaces dirigidos a otro sistema autónomo. Se utiliza para distribuir la carga entre los distintos enlaces, tomando los que contengan un valor menor de MED como prioridad.

- Local-Pref: En el caso de que un AS esté conectado a múltiples sistemas autónomos, otorga prioridad a uno de los enlaces de forma que se enviarán prioritariamente los datos por el enlace que tenga un valor de Local-Pref mayor.

Como se ha mencionado, estos atributos se utilizan con tal de elegir la mejor ruta disponible. Las rutas que tengan menor LOCAL-PREF, una cadena de AS-PATH más larga, un ORIGIN mayor y un MED mayor se eliminarán. Además, la ruta será ignorada si la dirección de NEXT-PATH no está disponible desde un router. Finalmente, eBGP tiene una prioridad mayor a iBGP, lo que quiere decir que si se aprende una ruta por eBGP, se eliminará la entrada antigua si se había aprendido por iBGP. Esto ocurre porque la distancia administrativa de eBGP es de 20 mientras que la de iBGP es de 200.

3.4. VPN

Una red privada virtual o VPN (*Virtual Private Network*), permite a un usuario conectarse a una red privada y acceder a la red pública de forma que parezca que el terminal del usuario está directamente conectado a esa red privada. Se podría decir por lo tanto que extiende una red local sobre la red pública. Para la conexión de la red local original con la red privada se emulan las propiedades de un enlace privado punto a punto, y para lograr este enlace, los paquetes se encapsulan con una cabecera que proporciona información de encaminamiento. Aunque no es obligatorio, las VPN suelen disponer de mecanismos de encriptación.

Una conexión VPN está compuesta por los siguientes elementos:

- Servidor VPN: El servidor VPN está configurado para aceptar conexiones desde un cliente VPN compatible.

- Cliente VPN: El cliente VPN inicia la conexión con un servidor VPN.

- Túnel: Un túnel es un enlace por el que se transmiten paquetes encapsulados entre un origen y un destino. Para incrementar la seguridad, los paquetes encapsulados pueden ir encriptados para evitar que terceros obtengan información relativa a dichos paquetes.
- Protocolos de tunneling: Existe un gran número de protocolos de tunneling que ayudan a establecer una VPN. PPTP, L2TP o IPSec son algunos de ellos. Para este TFG se ha empleado el protocolo de tunneling de código abierto OpenVPN. OpenVPN permite crear un túnel y encriptar su contenido mediante OpenSSL, y admite los protocolos de transporte TCP y UDP.
- Red de Tránsito: Se denomina red de tránsito a la red pública que es atravesada mediante el túnel.

Las redes privadas virtuales proporcionan por lo tanto confidencialidad, puesto que al encriptar el contenido de los paquetes, un atacante que pudiera analizar el tráfico de la red únicamente vería el tráfico encriptado. Asímismo, también proporcionan mecanismos de autenticación que impiden el acceso no deseado a la red privada.

Capítulo 4

Emulador de redes CORE

4.1. Introducción

CORE (*Common Open Research Emulator*) es un emulador de redes *open source* que debe su origen al proyecto IMUNES, del cual deriva. Fue desarrollado originalmente por el equipo de *Boeing Research and Technology*, y financiado por el Laboratorio de Investigación Naval de los Estados Unidos. Funciona tanto en sistemas BSD como en sistemas Linux, aunque este Trabajo Final de Grado se va a centrar únicamente en los sistemas operativos GNU/Linux.

El proyecto dispone de un repositorio de GitHub donde se aloja su código fuente bajo la Licencia FreeBSD. Se trata de un proyecto activo, ya que recibe actualizaciones con frecuencia, así como contribuciones al código, las cuales se han ido incrementando a lo largo de los últimos años.

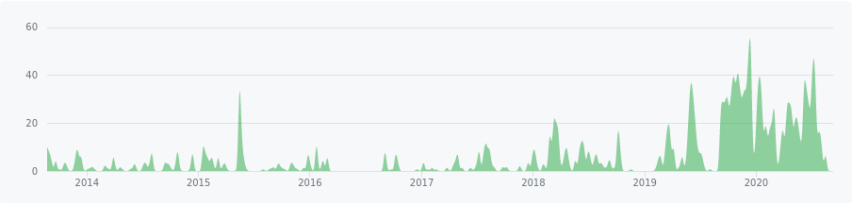

Figura 4.1: Número de contribuciones al repositorio de CORE entre 2013 y 2020.

Siendo un emulador de redes, los principales casos de uso de CORE son la investigación de protocolos de red y demostración del funcionamiento de los mismos, la realización de pruebas que permitan comprobar si una plataforma o aplicación funcionan de forma correcta, o la evaluación de ciertos escenarios en una red. También puede utilizarse en entornos docentes, aunque en este último caso se preferirá normalmente un simulador de redes, ya que el consumo de recursos del sistema será muy inferior.

Los diferentes nodos que se pueden desplegar en core (servidores, routers, PCs) se definen por los servicios que pueden ejecutar. Los servicios son procesos o *scripts* asignados a cada uno de los distintos nodos, y constituyen muchas de las funcionalidades que ofrece este emulador de redes. Entre los servicios disponibles se incluyen RIP, OSPF y BGP, servidor y cliente VPN, o SSH. Cada nodo dispone de un diálogo de configuración desde el cual se pueden activar o desactivar los distintos servicios.

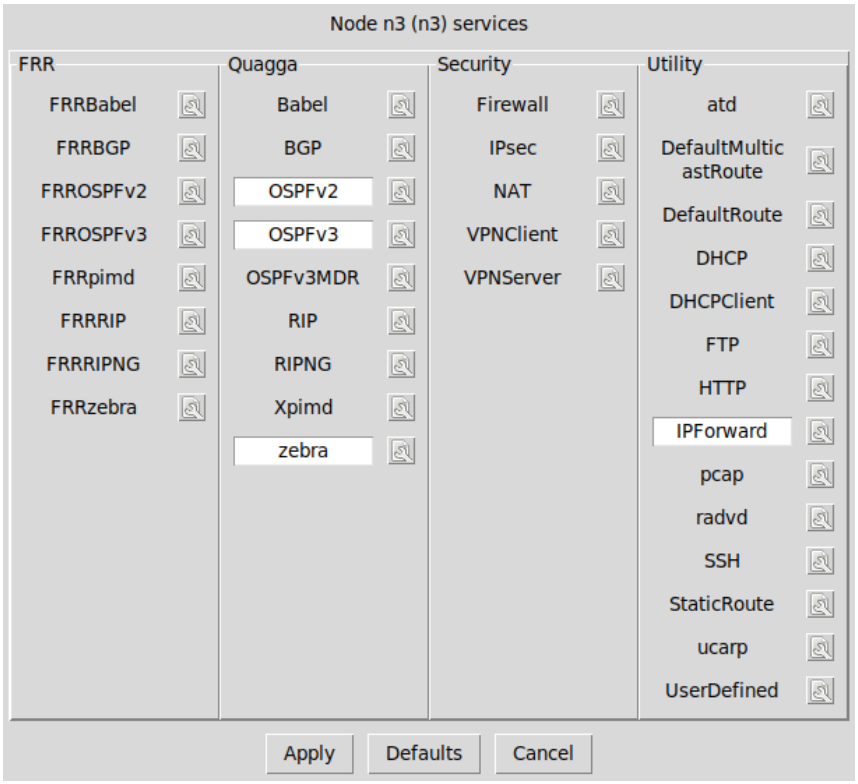

Figura 4.2: Lista de servicios disponibles para un nodo.

CORE es una herramienta que permite crear redes IP virtuales, ofreciendo la posibilidad de conectarlas a redes físicas. Algunas de sus características son:

- Utiliza herramientas de código abierto disponibles en el sistema operativo para ampliar su funcionalidad, lo cual hace que sea un software flexible y eficiente.
- Posee una interfaz de usuario sencilla. Esto posibilita crear una nueva red de forma simple e intuitiva.
- Ofrece la posibilidad de añadir servicios personalizados escritos en lenguaje Python, adaptándose así a las necesidades de cada usuario.
- Es de código abierto y tiene una comunidad activa, por lo tanto recibe mejoras y actualizaciones con frecuencia.

4.2. Arquitectura

CORE está formado por una arquitectura cliente-servidor, y se compone de dos elementos principales: El demonio o *daemon*, que hace de servidor, y la interfaz de usuario (CORE GUI), que actúa de cliente. Así mismo, también incluye las herramientas coresendmsg y vcmd.

Por un lado, el *daemon* será el encargado de crear y gestionar los nodos virtuales y los enlaces de datos. Los nodos se crean mediante el uso de *Linux Namespaces*, mientras que los enlaces se

crean mediante *Linux Bridges*. El demonio proporciona también una API que le permite interactuar con el resto de componentes de CORE.

Por otro lado, la GUI, basada en Tcl/Tk, permite controlar de manera sencilla al *daemon*, comunicándose con este mediante la API que se ha mencionado. La GUI contiene un gran número de utilidades preparadas para utilizarse de forma totalmente gráfica.

Aunque la GUI sea una de las mayores fortalezas de CORE, se proporciona también la herramienta coresendmsg que permite la interacción con el *daemon* mediante una interfaz de línea de comandos (CLI), sin la necesidad de pasar por una interfaz gráfica. De nuevo se utiliza la API del *daemon* para la comunicación entre ambos.

Por último, vcmd es un programa de línea de comandos que dará la posibilidad de enviar comandos de *shell* a los diferentes nodos. Puede ejecutarse directamente desde la GUI, o de forma manual desde un emulador de terminal.

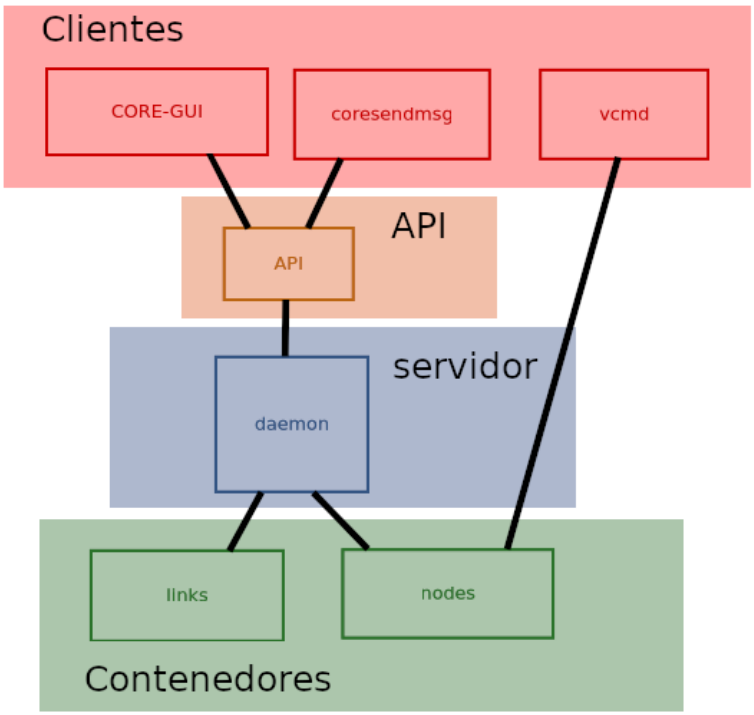

Figura 4.3: Arquitectura de CORE.

4.2.1. CORE-daemon

El *daemon* está escrito en el lenguaje de programación Python, y es el eje central de CORE. Con el fin de que los clientes del emulador puedan funcionar, este debe iniciarse de forma manual mediante *systemd*, o cualquier otro gestor de servicios disponible en la distribución de Linux que se esté utilizando, a través de un emulador de terminal. En el caso de *systemd*, para arrancar el *daemon* se puede usar el comando "*systemctl start core-daemon*" como superusuario. Sin embargo, esto únicamente causará que el *daemon* esté activo durante una sesión. Para permitir que este se inicie automáticamente cada vez que se encienda el sistema se puede utilizar el comando "*sudo*

systemctl enable core-daemon".

El *daemon* de CORE dispone de una API que se utiliza para la interconexión entre este y los clientes de CORE. El *daemon* creará la red que se haya diseñado a través del cliente mediante la combinación de *Linux Namespaces* con *Linux Bridges*. Los diferentes nodos (routers, switches, hosts…) se crean utilizando *Linux Namespaces*, mientras que los enlaces de red se crean utilizando *Linux Bridges*. Una vez se haya formado la red, se pueden alterar las características de los enlaces de datos a través del *Network Scheduler tc*.

Gracias al modelo cliente-servidor de CORE, los clientes y el servidor pueden ejecutarse en máquinas distintas. Además, el emulador ofrece la posibilidad de enlazar múltiples instancias del *daemon*, y controlarlas mediante un único cliente, pudiendo de este modo diseñar una red de grandes dimensiones.

4.2.2. CORE-GUI

La interfaz gráfica que ofrece CORE está basada en Tcl/Tk. Actualmente se está desarrollando una nueva interfaz gráfica basada en Python, pero se encuentra en fase beta.

La GUI de CORE es uno de los puntos más favorables de este emulador frente a otros emuladores de redes, puesto que permite al usuario diseñar redes y poner en funcionamiento los distintos escenarios de una forma muy amigable. Dispone de una barra de herramientas en el lateral izquierdo con utilidades relativas al diseño y análisis de las redes, una barra de menú en la zona superior desde donde se puede configurar la apariencia del emulador, o bien acceder a ciertas herramientas, y un área de trabajo, situada en la zona central.

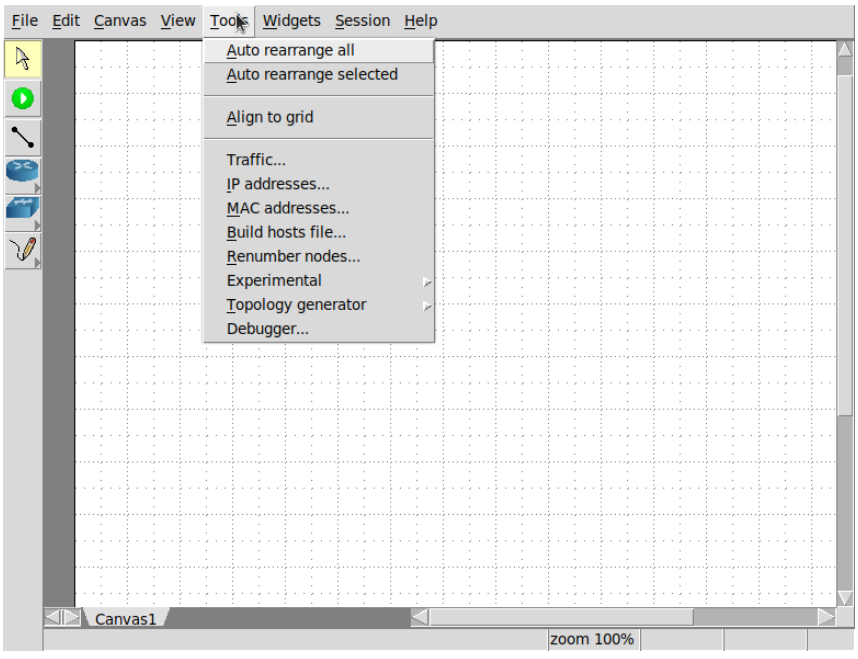

Figura 4.4: Interfaz gráfica CORE-GUI.

Desde la barra de menú se puede guardar una sesión o abrir una sesión que se haya guardado previamente. CORE permite guardar las sesiones tanto en formato IMN como en formato XML.

Asímismo, proporciona utilidades para configurar la direcciones IP y direcciones MAC de los nodos llamadas *IP addresses* y *MAC addresses* respectivamente, un generador de topologías, un generador de flujo de tráfico, y una utilidad para renombrar los nodos que se encuentren en el esquema de forma rápida, entre otras.

- IP addresses: Cuando se enlazan dos nodos, CORE asigna de manera automática una dirección IP a dichos nodos, que por defecto son de clase A. Mediante la utilidad *IP addresses*, situada en el menú de herramientas de la barra de menú, el usuario puede cambiar el tipo de clase de direcciones IP a asignar, pudiendo elegir entre las clases A, B o C. Una vez se hayan realizado los cambios deseados, cada vez que se haga una nueva conexión a una interfaz, se le asignará una IP de la clase seleccionada, de forma totalmente automática.
- MAC addresses: Del mismo modo, un enlace comenzará de forma predeterminada con la dirección MAC 00:00:00:aa:00:00. Esta utilidad permite cambiar el valor del primer dígito de las direcciones MAC, de forma que cada vez que se cree un nuevo enlace, este obtendrá una MAC que parte del valor introducido.
- Generador de topologías: Permite añadir el número de nodos deseado (por defecto routers, pero se puede cambiar al tipo de nodo que se prefiera) de forma inmediata al área de trabajo. Esta herramienta es útil si se quiere construir una red con un gran número de nodos interconectados entre sí, ya que permite automatizar esta tarea. Cuenta con patrones en forma de estrella o patrones circulares, entre otros.
- Generador de flujo de tráfico: Esta utilidad está situada junto a IP addresses y permite generar un flujo de tráfico tanto TCP como UDP siguiendo un patrón. Es sencillo de utilizar, sólamente hay que seleccionar el nodo de origen, el nodo de destino, y el patrón que seguirá el flujo que se quiera enviar. Una vez se haya configurado y la emulación esté en marcha, se puede iniciar el flujo de tráfico.
- Renumber nodes: Finalmente, desde aquí se pueden intercambiar los números de nodo entre dos nodos distintos, de forma muy sencilla. Una opción que puede resultar muy conveniente.

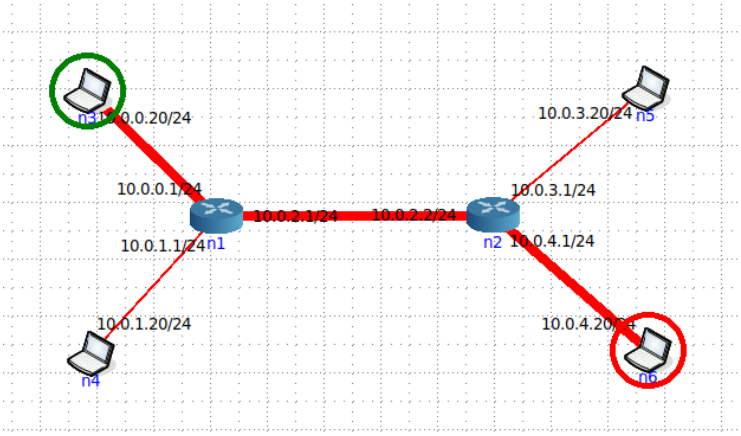

Figura 4.5: Flujo de tráfico generado entre los nodos n3 y n6.

La interfaz gráfica presenta dos modos de funcionamiento: el modo de edición y el modo de ejecución, y arranca en el modo de edición por defecto.

4.2.2.1. Modo de edición

El modo de edición incluye una barra de herramientas en la zona lateral con los diferentes nodos de los que dispone el *software*, así como la herramienta *link tool*, utilizada para establecer los enlaces entre dichos nodos. Además, se puede encontrar también la herramienta de selección, el botón de inicio de sesión, que pondrá en funcionamiento la emulación, y una herramienta para realizar anotaciones.

Figura 4.6: Barra de herramientas del modo edición de CORE-GUI.

Siguiendo el orden mostrado en la Figura 4.6, los elementos disponibles en el modo de edición son los siguientes:

1. La herramienta de selección es común a ambos modos de trabajo, y se utiliza principalmente para seleccionar y desplazar los distintos elementos a través del área de trabajo. Al hacer doble click sobre un nodo, se abrirá un diálogo de configuración para el mismo.
2. Una vez se haya terminado el diseño de la red, el botón de inicio de sesión dará la orden al *daemon* de crear los distintos nodos y enlaces, así como de poner en marcha la emulación.
3. La herramienta *link tool* permite enlazar dos nodos distintos al pulsar sobre uno de ellos y desplazar el cursor hacia el otro nodo.
4. A continuación se encuentran los distintos nodos de red que ofrece el emulador. Estos nodos vienen con servicios preconfigurados, aunque pueden cambiarse si fuera necesario. Los nodos de red que hay disponibles son los siguientes:
   - Router: Los routers de CORE funcionan gracias al software Quagga. Utilizan el protocolo OSPF por defecto, pero admiten también todos los protocolos que soporta Quagga, como RIP y BGP, entre otros. Además de Quagga, también pueden utilizar otros *software* de encaminamiento como XORP o BIRD.
   - Host: Los *hosts* emulan un servidor. Vienen con una ruta por defecto y el servicio de SSH activado de forma automática.
   - PC: Este nodo emula ordenadores con una ruta por defecto, pero no traen ningún servicio activo.
   - Router MDR: Similares al otro tipo de router de core, pero los routers MDR (*MANET Designated Router*) utilizan el protocolo OSPF-MDR. Este protocolo es una extensión de OSPF optimizada para redes ad hoc móviles, y viene de la mano de BOEING.

   Los servicios preconfigurados para cada nodo pueden cambiarse accediendo al menú de configuración del nodo correspondiente, y activando o desactivando el servicio deseado haciendo click en el nombre del mismo.
5. El siguiente elemento disponible en el modo de edición de CORE-GUI es el de los nodos de capa de enlace de datos. CORE ofrece los siguientes nodos: Hub, Switch, Wireless LAN, RJ45 y Tunnel.

- Hub y Switch: Se emplean para redirigir los paquetes recibidos a los nodos que tengan conectados. Los switch no disponen de ninguna opción de configuración en CORE, por lo que no será posible configurar redes de área local virtuales (VLAN) desde este emulador de redes.
- Wireless LAN: Al enlazar un router con este nodo, se mostrará una antena en el router indicando que se ha enlazado a una red inalámbrica.
- RJ45: El nodo RJ45 permite vincular la red emulada con una red física. Esta herramienta tiene un inconveniente, y es que necesita una interfaz de red física para cada conexión.
- Tunnel: Esta herramienta se utiliza para enlazar varias instancias del *daemon* de CORE entre sí. Utiliza para ello túneles GRE (*Generic Routing Encapsulation*). A diferencia de la herramienta RJ45, no hay que dedicar una interfaz física para CORE.

6. El último de los elementos disponibles en la barra de herramientas del modo de edición de la interfaz gráfica de CORE es la herramienta para realizar anotaciones. Esta herramienta tiene como objetivo ayudar a resaltar las distintas redes, y da la opción de añadir notas explicativas mediante la herramienta de texto.

Una vez esté listo el diseño de la red, el modo de edición permite configurar los distintos elementos. Como se ha mencionado anteriormente, se puede acceder a la configuración de dichos elementos desde la herramienta de selección haciendo doble click sobre un elemento. Al abrir la configuración de un nodo, se mostrará un diálogo que permitirá modificar las direcciones IP y MAC asignadas a cada una de las interfaces del nodo. Además, se podrá acceder a la lista de servicios que soporta dicho elemento, y activarlos o desactivarlos dependiendo del escenario que se quiera reproducir.

De la misma forma, los enlaces también pueden ser configurados, y se accede a su ventana de configuración de la misma forma que se accede a la de los nodos. Los enlaces permiten configurar parámetros tales como el ancho de banda, los milisegundos de retraso y *jitter*, o el porcentaje de paquetes que se pierden o que se envian por duplicado.

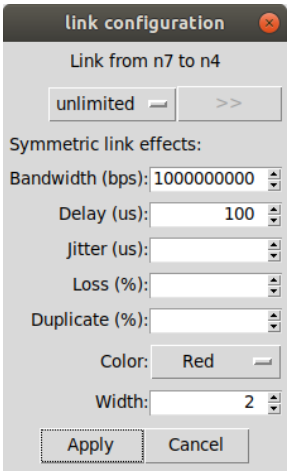

Figura 4.7: Ventana de configuración de enlaces.

4.2.2.2. Modo de ejecución

El segundo modo de funcionamiento de la GUI, el modo de ejecución, se alcanza una vez se ha presionado el botón de inicio en el modo de edición. Es posible iniciar CORE en el modo de ejecución directamente: Para ello hay que incluir el parámetro –start e indicar la ruta a un escenario IMN que se haya creado anteriormente, cuando se vaya a arrancar el software, mediante el uso de un emulador de terminal.

La primera diferencia con respecto del otro modo es que la barra de herramientas es distinta:

Figura 4.8: Barra de herramientas del modo de ejecución de CORE-GUI.

Una vez más, siguiendo el orden que aparece en la figura, los elementos son los siguientes:

1. Como se ha mencionado anteriormente, la herramienta de selección se encuentra disponible también en este modo. La principal diferencia es que esta vez, al hacer doble click sobre un nodo, se abre una ventana que permitirá enviar comandos de *shell* desde esa máquina. Además, al hacer click derecho sobre un nodo, aparece un diálogo desde el cual se pueden iniciar o detener los servicios asignados al nodo, o lanzar aplicaciones tales como Wireshark.

2. El siguiente elemento en la lista, el botón de parada de sesión, detiene la emulación que se esté ejecutando y devuelve al programa al modo de edición.

3. La herramienta de obervación invoca una ventana flotante sobre los nodos con información como los procesos activos para cada nodo, su correspondiente tabla de encaminamiento, o la información relativa a las interfaces de red. Además de los *widgets* que se incluyen, la herramienta da la opción al usuario de añadir *widgets* personalizados.

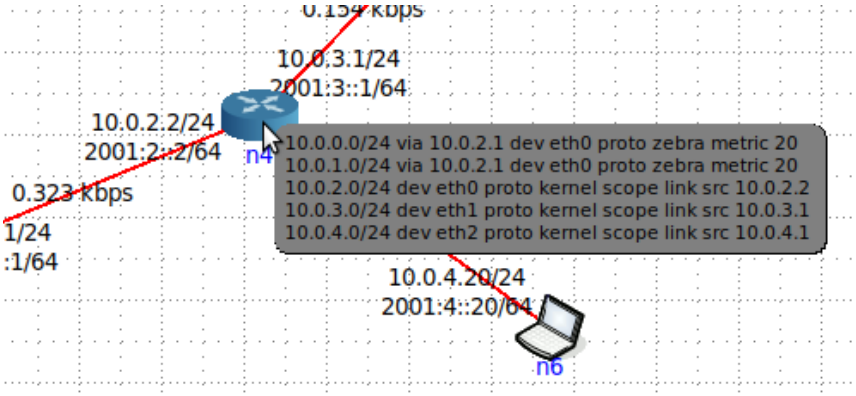

Figura 4.9: Tabla de encaminamiento de n4 utilizando un widget de la herramienta de observación.

4. A través de la siguiente herramienta, al seleccionar un enlace se genera un gráfico con el flujo de datos en tiempo real que circula por dicho enlace.

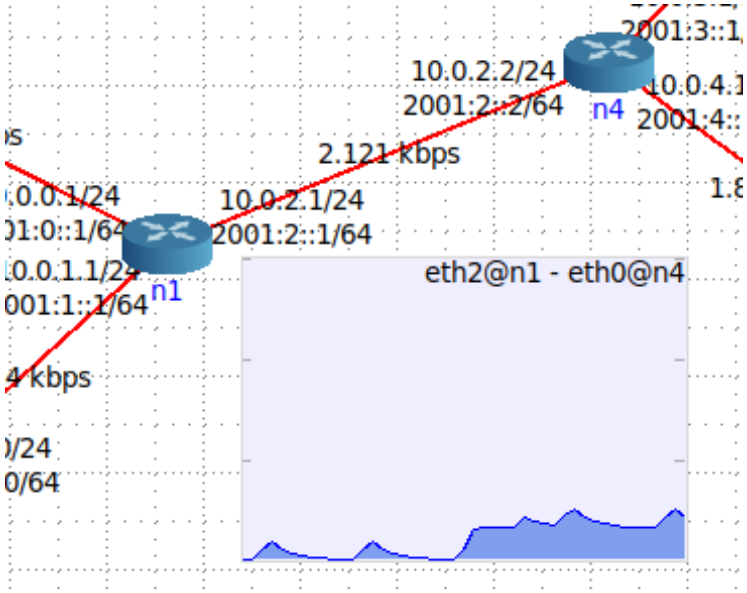

Figura 4.10: Gráfica con el flujo de datos en el enlace entre los nodos n1 y n4.

5. Este modo de trabajo incluye también la herramienta de anotaciones, aunque en este caso únicamente permite realizar anotaciones a mano alzada.

6. Otra de las utilidades accesibles a través del modo de ejecución es la *Two-node Tool*, o herramienta de dos nodos. La herramienta de dos nodos permite seleccionar de forma gráfica un nodo de origen y un nodo de destino y, posteriormente, permite ejecutar o bien un *traceroute* entre esos dos nodos, o bien un *ping* que continuará hasta que el usuario lo detenga. El hecho de que funcione por medio de la interfaz gráfica hace que sea muy sencilla de utilizar, y es una herramienta muy útil para comprobar que la red está funcionando de forma correcta.

   Para poder utilizar la funcionalidad de *traceroute* de CORE, es necesario tener la herramienta *traceroute* instalada. En el caso de Ubuntu, existen dos paquetes que proporcionan esta herramienta: *inetutils-traceroute* y *traceroute*. El primero de ellos es una implementación más tradicional, mientras que el segundo es más moderno, y por lo tanto ofrece un mayor número de funcionalidades.

   Esta información es relevante debido a que el comando de *traceroute* utilizado por CORE hace uso de una de las funcionalidades exclusivas de la segunda implementación, y como consecuencia será recomendable la instalación de esta última para evitar el tener que realizar modificaciones en el comando que proporciona CORE cada vez que se quiera hacer uso de esta herramienta.

7. Por último, la herramienta *Run Tool* permite ejecutar un comando sobre uno o varios nodos de manera rápida. La única condición es que el comando tiene que terminar por sí solo, ya que el comando ejecutado no puede cancelarse y la ventana espera a que este termine para mostrar los resultados. Esto quiere decir que si se ejecuta por ejemplo un *ping* sin indicar un número máximo de mensajes, la ventana de *Run Tool* no mostrará ningún resultado ya que el comando seguirá ejecutándose y no podrá cancelarse.

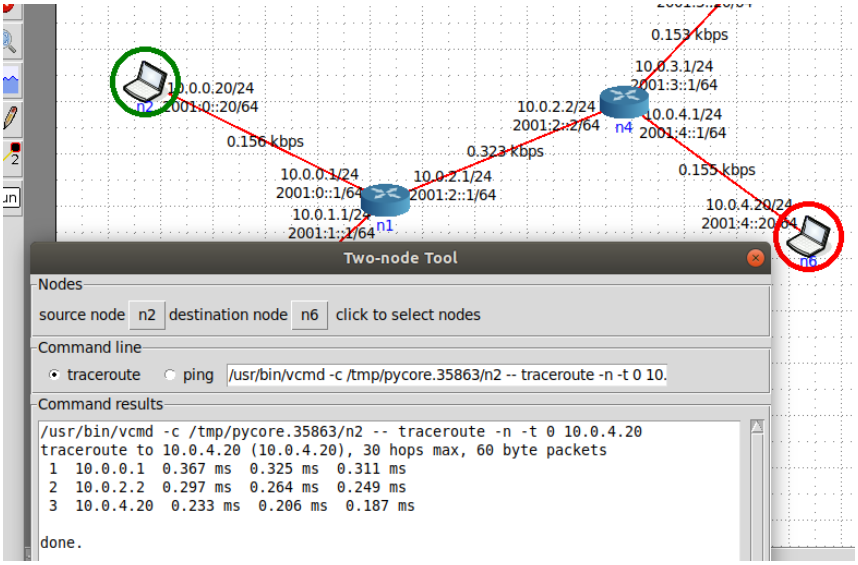

Figura 4.11: Traceroute entre los nodos n2 y n6 mediante la herramienta de dos nodos.

4.2.3. Coresendmsg y vcmd

4.2.3.1. Coresendmsg

CORE pone a disposición del usuario el cliente coresendmsg. Se trata de un cliente que funciona a través de CLI (*Command Line Interface*), sirviendo como una alternativa al uso de la interfaz gráfica.

Al funcionar a través de la línea de comandos, es mucho más complicado de utilizar que la GUI, pero la gran ventaja que tiene es que permite automatizar algunas tareas mediante el uso de *scripts*.

```
Usage: coresendmsg [-h|-H] [options] [message-type] [flags=flags] [message-TLVs]

Supported message types:
  ['NODE', 'LINK', 'EXECUTE', 'REGISTER', 'CONFIG', 'FILE', 'INTERFACE', 'EVENT', 'SESSION', 'EXCEPTION']
Supported message flags (flags=f1,f2,...):
  ['ADD', 'DELETE', 'CRI', 'LOCAL', 'STRING', 'TEXT', 'TTY']

Options:
  -h, --help            show this help message and exit
  -H                    show example usage help message and exit
  -p PORT, --port=PORT  TCP port to connect to, default: 4038
  -a ADDRESS, --address=ADDRESS
                        Address to connect to, default: localhost
  -s SESSION, --session=SESSION
                        Session to join, default: None
  -l, --listen          Listen for a response message and print it.
  -t, --list-tlvs       List TLVs for the specified message type.
  --tcp                 Use TCP instead of UDP and connect to a session
                        default: False
```

Figura 4.12: Mensaje de ayuda con las distintas opciones de coresendmsg.

No obstante, a esta herramienta aún le queda un largo camino por delante, ya que dispone de muchas menos funcionalidades que la interfaz gráfica de CORE.

4.2.3.2. Vcmd

Cuando se crea un nuevo nodo, o lo que es lo mismo, cuando se crea un nuevo espacio de nombres, CORE asigna una instancia del demonio *vnoded* a dicho espacio de nombres, donde se ejecutará con un ID de proceso 1. Esto ocurre de forma automática, y se repetirá para cada uno de los nodos que han de crearse.

El programa vcmd establece una conexión con el demonio *vnoded* a través de un canal de control que funciona por medio de *sockets*, y permite ejecutar comandos en el espacio de nombres. Mediante vcmd se puede obtener información del sistema, realizar un ping desde un nodo a cualquier otro para comprobar el estado de la red, mostrar información sobre los interfaces de red, mostrar las tablas de encaminamiento del nodo, entre muchas otras cosas.

Vcmd necesita dos argumentos para poder arrancar, y estos son el nombre del canal de control, y el comando que se va a ejecutar. El nombre del canal de control será de la forma "/tmp/pycore.nnnnn/NN", donde n equivale al identificador de sesión actual de CORE, y N equivale al nombre del nodo desde el que se quiere ejecutar el comando.

Para acceder a vcmd desde la GUI, simplemente basta con hacer doble click sobre uno de los nodos mientras haya una emulación activa, y se abrirá un emulador de terminal preparado para ejecutar comandos desde ese nodo. Para utilizarlo mediante coresendmsg, se emplea el tipo de mensaje EXECUTE junto a COMMAND con el comando que se quiera ejecutar desde el nodo.

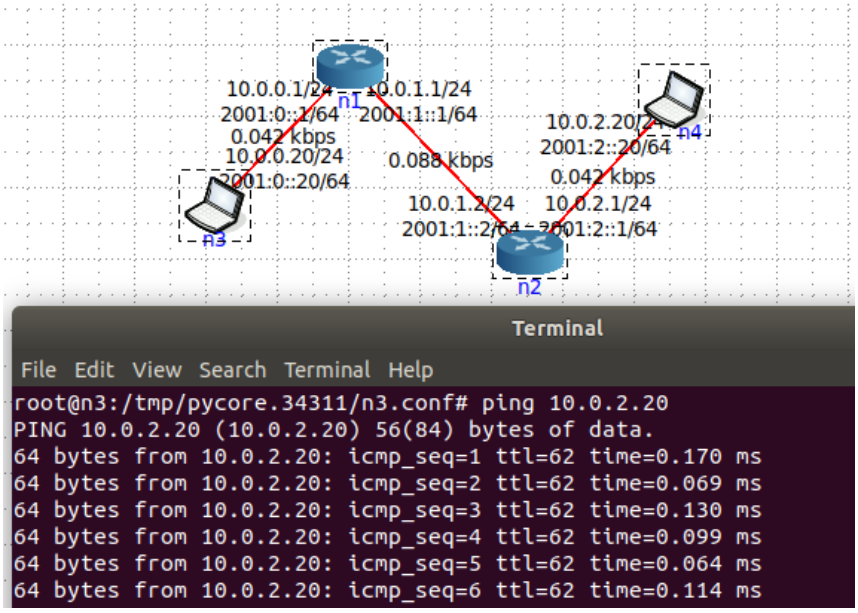

Figura 4.13: Ping desde n3 hasta n4 utilizando vcmd a través de la línea de comandos.

Capítulo 5

Despliegue y configuración de redes

5.1. Introducción

Tal y como se ha mencionado en los anteriores capítulos, CORE soporta múltiples servicios y protocolos. En este capítulo se configurarán algunos de estos servicios, concretamente RIP, OSPF, BGP y VPN, y se profundizará en el funcionamiento del emulador mediante una serie de escenarios donde se explicarán sus capacidades y limitaciones. Finalmente se realizará el análisis del tráfico que circula por los escenarios para ayudar a determinar el comportamiento de los distintos protocolos dentro de CORE.

CORE funciona de manera oficial en las distribuciones de Linux Ubuntu y CentOS, aunque funcionará también en cualquier distribución de Linux que disponga de un *kernel* con la funcionalidad de *namespaces* activa, lo cual sucede en la gran mayoría de los casos hoy en día. Para este Trabajo de Final de Grado se ha instalado la versión 7.2.0 del emulador CORE en una máquina virtual con la distribución de Ubuntu 20.04 LTS.

Desafortunadamente, CORE no se encuentra disponible en los repositorios de paquetes de las distintas distribuciones, algo que facilitaría en gran medida el proceso de la instalación, y por lo tanto se deben instalar las dependencias del *software*, y realizar la compilación del código del emulador de forma manual. Para facilitar esta tarea, desde el proyecto se proporciona un *script* cuyo objetivo es el de automatizar la instalación. La desventaja es que si se está utilizando una distribución de Linux distinta a las recomendadas, es muy posible que haya que modificar dicho *script* para ajustarlo a la distribución escogida.

Algo a destacar es que CORE utiliza *software* externo para cumplir con ciertas de sus funcionalidades, pero este *software* no tiene por qué estar instalado en el *host*, y depende del usuario instalarlo o no dependiendo de las necesidades que se requieran, otorgando por lo tanto cierto grado de flexibilidad.

BIRD, FRR o Quagga son algunos de los programas soportados por CORE para permitir el funcionamiento de los routers emulados. Para este TFG se ha elegido Quagga ya que soporta un mayor número de protocolos que el resto y posee una sintaxis de configuración similar a la de los routers de CISCO.

5.2. Configuración de servicios

5.2.1. Routing Information Protocol

En el primer escenario se ha diseñado una red que dispone de tres routers, los cuales han sido configurados para utilizar el protocolo de encaminamiento RIP. CORE asigna por defecto los servicios OSPFv2 y OSPFv3 a los routers cuando son creados, por lo que hay que acceder a la ventana de configuración de servicios de cada router, desactivar los servicios de OSPF y activar el servicio de RIP.

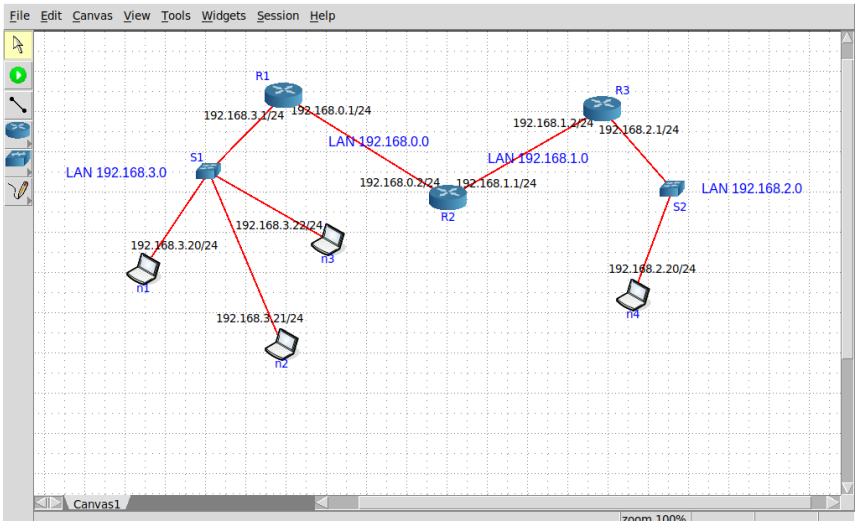

Figura 5.1: Esquema de red basado en RIP.

Para el correcto funcionamiento de RIP, será necesario dejar activos los servicios RIP, zebra e IPForward para cada router. El servicio IPForward permite que el nodo (router en este caso), envíe los paquetes recibidos de un puerto a otro, siempre y cuando esté indicado en su tabla de encaminamiento. Por otro lado, el servicio zebra aloja la configuración del protocolo RIP, y es necesario para que dicho protocolo funcione a través del *software* Quagga.

Dentro del servicio zebra, Quagga proporciona un esquema de configuración para RIP, el cual habrá que modificar para adaptarlo acorde a la topología de red que se ha diseñado. Para la configuración de RIP, hay que especificar para cada uno de los routers las interfaces en las que el protocolo estará activo, y esto se puede realizar mediante el comando *network*. La configuración final para el router R2 será la mostrada en la figura 5.3, y de forma similar, se repetirá el proceso para los routers R1 y R3, teniendo en cuenta que sus interfaces serán ligeramente distintas.

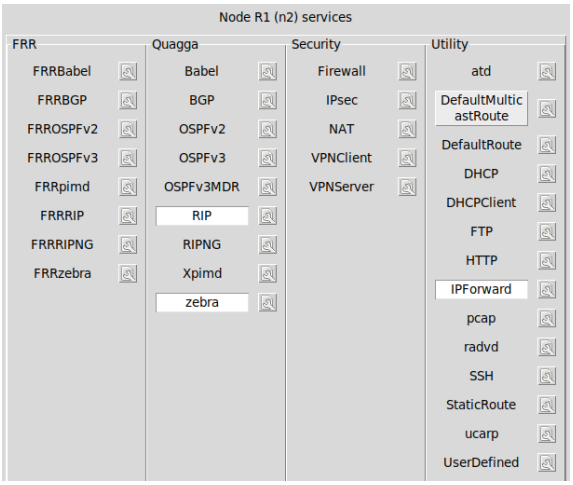

Figura 5.2: Lista de servicios activos en los routers RIP.

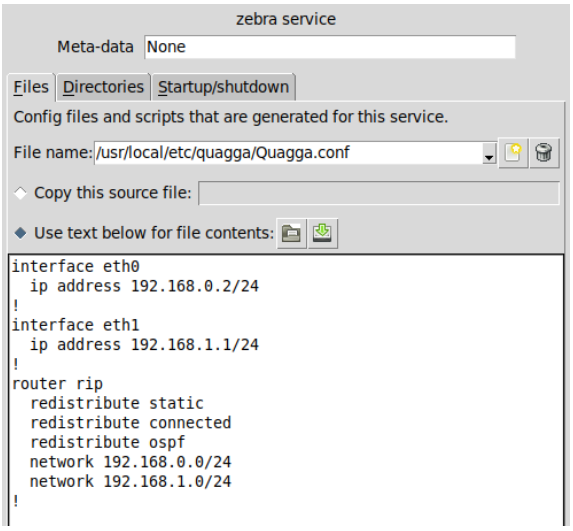

Figura 5.3: Configuración de RIP para el router R2.

Una vez iniciada la emulación, CORE proporciona ciertas herramientas que permiten interactuar con dicha emulación. Por una parte, tcpdump, tshark y Wireshark dan la posibilidad de capturar y analizar el tráfico que circula por las interfaces de red. Un detalle a tener en cuenta es que por defecto, en Linux, Wireshark únicamente permite realizar capturas si se dispone de permisos de superusuario, o si el usuario se encuentra en el grupo wireshark. Si CORE se ejecuta a través de un usuario corriente, no se dispone de la opción de ejecutarlo como superusuario, por lo tanto será recomendable añadir al usuario al grupo wireshark. Para hacer esto, se puede utilizar el comando *gpasswd*, especificando el usuario y el grupo pertinentes, de la siguiente forma: "*sudo gpasswd -a user wireshark*".

Volviendo al emulador, la herramienta *vtysh* ofrece al usuario la posibilidad de interactuar con el *software* Quagga mediante comandos, pudiendo de este modo obtener información muy útil como las tablas de encaminamiento. Se puede acceder a estas utilidades haciendo click derecho sobre uno de los nodos a través de la interfaz gráfica.

Desde la ventana de *vtysh*, al ejecutar el comando "*show ip route*", Quagga responderá con la tabla de encaminamiento. Como se puede apreciar en la figura 5.4, para el caso del router R1, tanto las redes 192.168.0.0 como 192.168.3.0 están directamente conectadas, mientras que las entradas para las redes 192.168.1.0 y la 192.168.2.0 se han obtenido a través del protocolo RIP, y se puede acceder a ellas por medio de la interfaz eth0 del router R2. La tabla muestra además la distancia administrativa para estas dos redes (120 en el caso de RIP), y la métrica o número de saltos: 2 en el caso de la primera, y 3 en el caso de la segunda.

```
Hello, this is Quagga (version 1.2.4).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

R1# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, P - PIM, A - Babel, N - NHR
       > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.0.0/24 is directly connected, eth0
R>* 192.168.1.0/24 [120/2] via 192.168.0.2, eth0, 00:01:57
R>* 192.168.2.0/24 [120/3] via 192.168.0.2, eth0, 00:01:52
C>* 192.168.3.0/24 is directly connected, eth1
R1#
```

Figura 5.4: Tabla de encaminamiento de R1.

A continuación, utilizando Wireshark, se ha realizado una captura en la interfaz eth1 (con dirección 192.168.1.1) del router R2 (Figura 5.5).

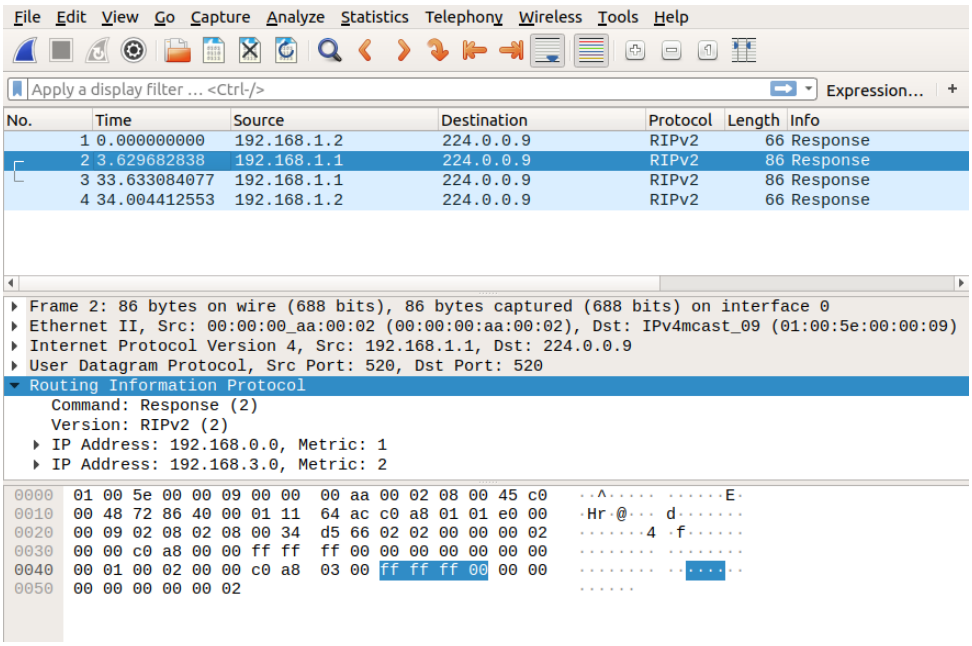

Figura 5.5: Captura de las tramas de respuesta de RIP.

En la captura se muestran dos respuestas diferentes: La primera, cuyo tamaño es de 66 Bytes, corresponde a la respuesta enviada por el router R3, e indica la información acerca de la red 192.168.2.0. La segunda respuesta, de 86 Bytes de tamaño, muestra información de las redes 192.168.0.0 y 192.168.3.0. Como se puede observar, las respuestas son tramas UDP que van sobre el puerto 520, el puerto que utiliza RIP por defecto. Otra de las características de RIP, y que puede apreciarse en la figura mostrada anteriormente, es que la información de actualización se envía a la dirección 224.0.0.9 de multicast, ya que se está utilizando la versión 2 del protocolo. Por último, la captura muestra también el uso de los temporizadores a la hora de enviar actualizaciones. Por defecto RIP utiliza un temporizador de 30 segundos sumado a un tiempo aleatorio para enviar la información. Es por esto que la primera trama ha tardado 34 segundos en repetirse, pero la segunda trama sólo ha tardado poco más de 30 segundos.

5.2.2. Open Shortest Path First

En el segundo escenario, se va a proceder a la configuración del protocolo OSPF. Este protocolo es el que utiliza por defecto CORE cuando se crean nuevos routers. El esquema de configuración proporcionado por Quagga asigna por defecto todos los routers al área 0, por lo tanto se tendrán que realizar modificaciones al fichero de configuración si se diseña una topología de red con más de un área.

Para el diseño de esta red se ha optado por una topología con tres áreas. Tal y como se muestra en la figura 5.6, el área 0, o área de *backbone*, está compuesta por los routers R1, R2, R3 y R4, siendo R2 y R4 ABR (*Area Border Routers*). Del mismo modo, el área 1 estará formada por los routers R4, R5 y R6, y el área 2 estará compuesta por los routers R2, R7 y R8.

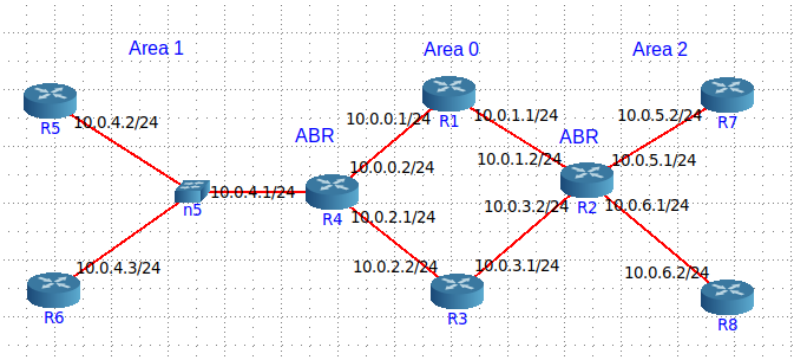

Figura 5.6: Esquema de red basado en OSPF.

El siguiente paso es el de activar los servicios correspondientes para cada uno de los routers. En este caso, los servicios de interés serán OSPFv2, zebra e IPForwarding (Figura 5.7). A continuación, a través del servicio zebra, se modifica la configuración de OSPF ya que en el esquema definido hay más de un área, y como se ha dicho anteriormente, CORE considera a todos los routers parte de la misma área por defecto. Para realizar la configuración del protocolo OSPF, se utiliza de nuevo el comando *network*, seguido de la subred a la que pertenece la interfaz sobre la que se va a activar OSPF, y finalizando por último con el área a la que pertenece dicha interfaz.

El router R4 tiene una interfaz perteneciente al área 1; la 10.0.4.1/24, mientras que las dos restantes, 10.0.0.2/24 y 10.0.2.2/24, pertenecen al área 0. De este modo, se tendrá que realizar la

modificación del fichero de configuración tal y como se muestra en la figura 5.8. Este proceso deberá repetirse para cada uno de los routers de la red.

CORE también proporciona una utilidad para visualizar de forma clara el proceso mediante el cual se establecen las adyacencias entre los routers vecinos. Se trata de la herramienta de adyacencia, que muestra enlaces de distintos colores según el estado en el que se encuentre cada interfaz. La herramienta actualiza los enlaces en tiempo real, lo que quiere decir que el color de dichos enlaces cambiará a medida que se vaya formando la adyacencia.

Esta herramienta resultará muy útil en la siguiente sección, cuando se profundice en la creación de adyacencias.

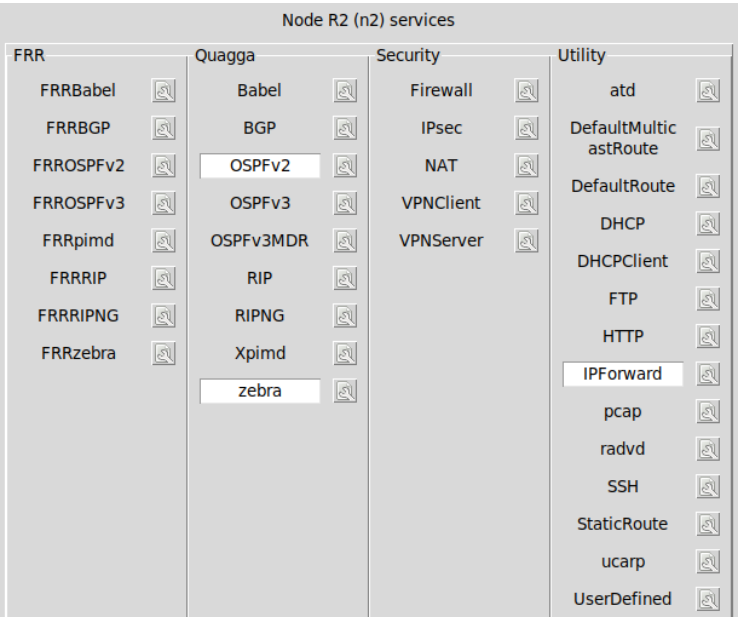

Figura 5.7: Lista de servicios activos en los routers OSPF.

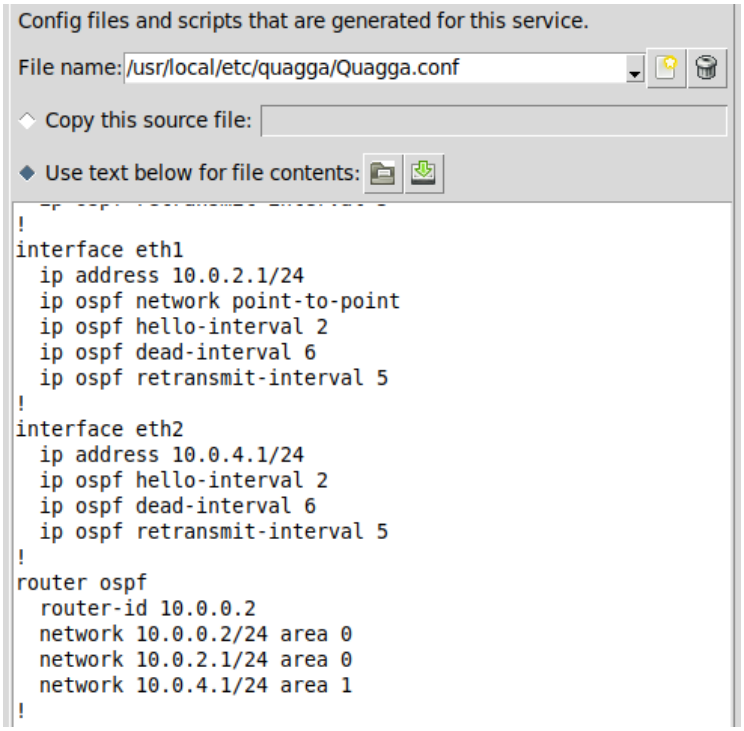

Figura 5.8: Configuración de OSPF para el router R4.

A continuación se inicia la emulación de la red que se ha diseñado, y se va a tratar de explicar, con ayuda de Wireshark y la herramienta de observación de CORE, el proceso mediante el cual se establecen las adyacencias entre los routers. En concreto, se va a analizar la adyacencia formada entre los routers R4 y R5:

- Estado down a estado init: En esta primera fase, para averiguar si hay vecinos OSPF en el mismo enlace, los routers envían paquetes HELLO por cada una de las interfaces en las que OSPF esté activado. En este caso, el router R4 envía el HELLO con su identificador de router a R5. La herramienta de adyacencias representa el estado init con el color amarillo.

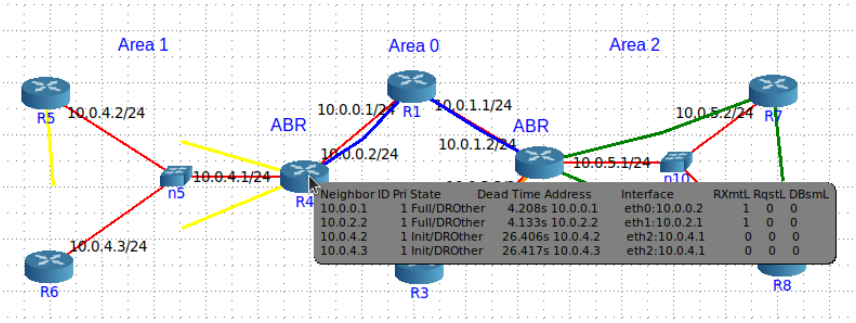

Figura 5.9: Estado de adyacencia Init.

- Estado init a 2-way: En la segunda fase, el router recibe el paquete HELLO con el identificador del primer router, y si este identificador no se encuentra en su lista de vecinos, lo

añade y responde con otro paquete HELLO que contendrá esta vez tanto el identificador del segundo router como el del primero. A continuación, el primer router recibe el HELLO, añade al segundo router a la lista de vecinos, y como el primer router ha recibido un paquete que contiene su propio identificador de router, este pasa al estado 2-way. En la figura 5.11, tanto R4 como R5 se han añadido mutuamente a la lista de vecinos, y han pasado al estado 2-Way. La herramienta de adyacencias representa este estado con el color verde.

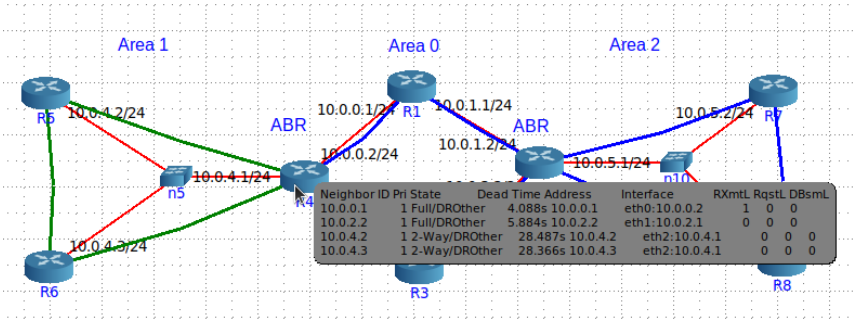

Figura 5.10: Estado de adyacencia 2-Way.

- Elección del DR: Como los routers pertenecientes al área 1 no están conectados mediante enlaces punto a punto, se debe escoger un router designado (DR), y un router designado de respaldo (BDR). Para elegir a un router designado, se tiene en cuenta la prioridad configurada para cada uno de ellos, y si todos tienen la misma prioridad, entonces se elige el router con un identificador mayor. En este caso no se ha configurado la prioridad y todos tienen la misma, por lo tanto el router designado será el de mayor ID, es decir, el router R6. El router designado de respaldo sería, por lo tanto, R5, ya que es el router con un ID mayor de entre los restantes.

  Una vez se han elegido estos routers, se pasa a la fase de sincronización de bases de datos.

- Estado ExStart: En el estado ExStart se determina cual de los routers será maestro y cual será esclavo. Esta decisión se toma eligiendo al router con mayor ID, que actuará como maestro. Como se muestra en la figura 5.12, es el router R6 con identificador 10.0.4.2 el que se comporta como maestro.

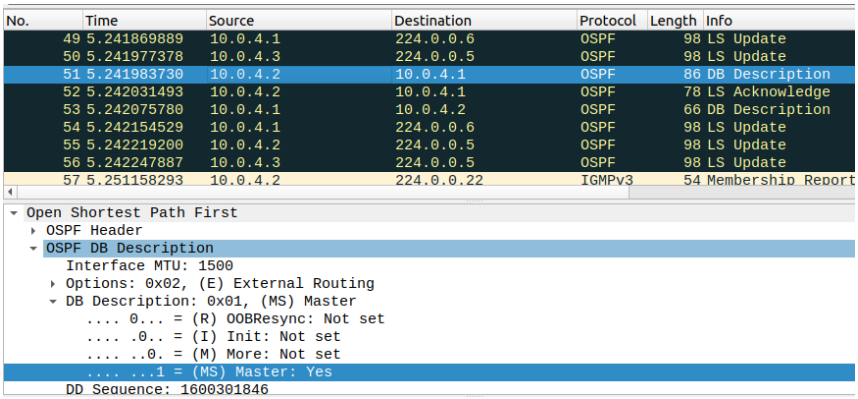

Figura 5.11: Elección de router maestro y esclavo.

- Estado Exchange: En este estado, los routers sincronizan las entradas de sus LSDB (*Link-State Database*). Comienza entonces el intercambio entre paquetes DBD (*Database Description*) que contienen información sobre las entradas de LSA (*Link State Advertisement*) de los routers. Una vez se han sincronizado todas las entradas de los routers, el enlace pasa a estado Full y la adyacencia se considera terminada (Figura 5.12).

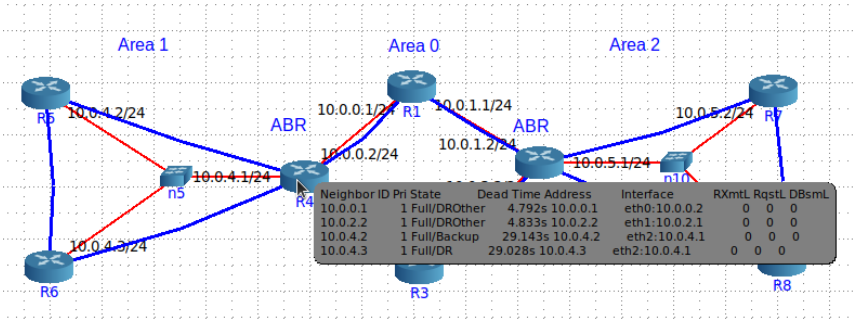

Figura 5.12: Estado final de las adyacencias.

Volviendo a la configuración de OSPF, es posible cambiar la configuración de un router para asignarle prioridad a una de sus interfaces, haciendo que se escoja como router designado incluso aunque su identificador no sea el más alto. En el escenario anterior, el DR ha sido R6 ya que su ID (10.0.4.3) era mayor que el del resto de routers del área, pero ahora se va a incrementar la prioridad de R4, con ID 10.0.0.2, para que salga como DR en lugar de R6. El cambio de prioridad se realiza mediante "*ip ospf priority N*", donde N es el nuevo valor de prioridad que se va a otorgar, en este caso 2 ya que el resto tiene prioridad 1 (Figura 5.14).

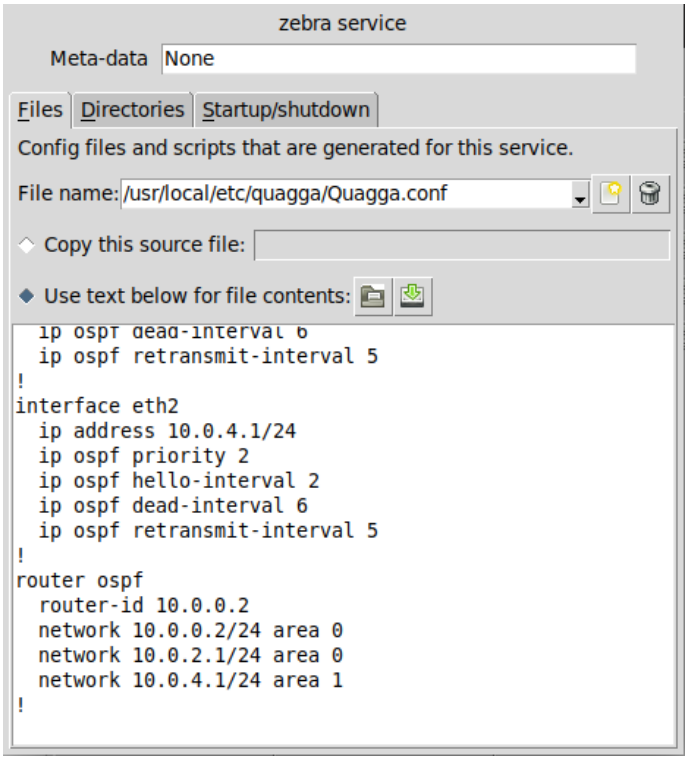

Figura 5.13: Configuración especificando el cambio de prioridad de R4.

Para comprobar que el resultado sea el esperado, esta vez se va a utilizar la herramienta *vtysh*, junto con el comando "*show ip ospf neighbor*", desde R5. La tabla obtenida muestra al router con ID 10.0.0.2, es decir, R4, como router designado, mientras que R6 será esta vez el router de respaldo (Figura 5.15).

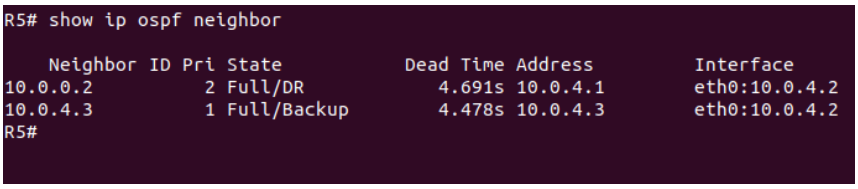

Figura 5.14: R4 como Router Designado.

Finalmente, también a través de *vtysh*, se puede obtener la tabla de encaminamiento de los routers. La tabla muestra si las distintas subredes están directamente conectadas, o si por el contrario han sido aprendidas por medio de OSPF. También muestra la distancia administrativa (110 en el caso de OSPF), así como el coste para alcanzar los distintos destinos (Figura 5.16).

```
R4# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, o - OSPF6, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O   10.0.0.0/24 [110/10] is directly connected, eth0, 00:15:17
C>* 10.0.0.0/24 is directly connected, eth0
O>* 10.0.1.0/24 [110/20] via 10.0.0.1, eth0, 00:15:05
O   10.0.2.0/24 [110/10] is directly connected, eth1, 00:15:17
C>* 10.0.2.0/24 is directly connected, eth1
O>* 10.0.3.0/24 [110/20] via 10.0.2.2, eth1, 00:15:05
O   10.0.4.0/24 [110/10] is directly connected, eth2, 00:15:17
C>* 10.0.4.0/24 is directly connected, eth2
O>* 10.0.5.0/24 [110/30] via 10.0.0.1, eth0, 00:15:05
                         via 10.0.2.2, eth1, 00:15:05
C>* 127.0.0.0/8 is directly connected, lo
R4#
```

Figura 5.15: Tabla de encaminamiento de R4.

5.2.3. Border Gateway Protocol

Para el tercer escenario, se dispone de tres AS (*Autonomous System*) conectados entre sí. Para este ejemplo se utilizará el protocolo de puerta de enlace de frontera (BGP) tanto para las conexiones internas dentro de un mismo sistema autónomo (iBGP), como para las conexiones entre distintos sistemas autónomos (eBGP). El AS 1 está compuesto por los routers R1, R2 y R3, y tanto R3 como R2 se encargarán de anunciar los prefijos 10.0.1.0/24, 10.0.2.0/24 y 10.0.3.0/24 al resto de vecinos. Por otra parte, al AS 2 lo forman los routers R4 y R5, que anunciarán los prefijos 10.0.8.0/24 y 10.0.9.0/24. Por último, son los routers R6, R7 y R8 los que componen el AS 3, y se encargarán de anunciar los prefijos 10.0.5.0/24, 10.0.6.0/24 y 10.0.10.0/24 al resto de *peers* (Figura 5.17).

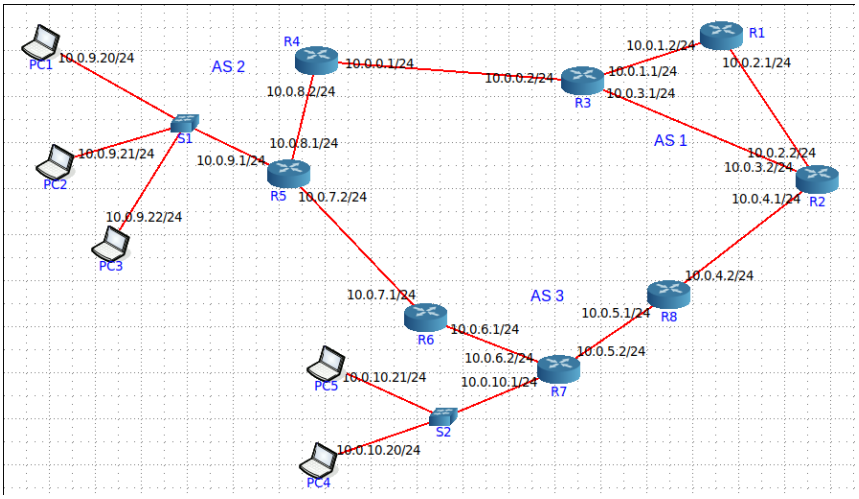

Figura 5.16: Topología de red basada en BGP.

Los servicios que se han activado para permitir que BGP funcione en los routers son los servicios de BGP, zebra e IPForward como se muestra en la figura 5.18).

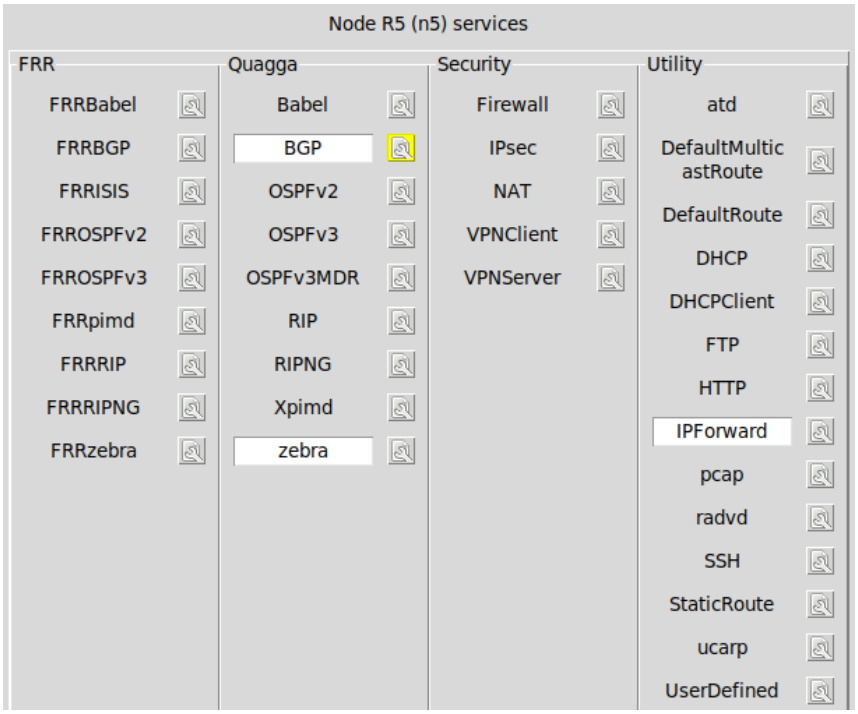

Figura 5.17: Servicios activados para el funcionamiento de BGP.

A diferencia de OSPF, BGP no establece las relaciones de forma automática con los routers vecinos, y se debe, por lo tanto, especificar dicha información en el fichero de configuración de los distintos routers. Para ello, una vez más, se deberá modificar la plantilla de configuración de Quagga, dentro del servicio Zebra.

Para la configuración de BGP en un router, en primer lugar se especifica el número del sistema autónomo al que pertenece dicho router, seguido de su ID de router, y finalmente se establecen las relaciones entre los *peers*. Para configurar los *peers* en BGP, primero se debe especificar la palabra *neighbor*, seguido de la dirección IP de los interfaces de otros routers que estén conectadas al router que se está configurando, y por último se indica el sistema autónomo al que pertenece el vecino. En el caso de que los routers se encuentren en el mismo AS, se utiliza iBGP y el número del sistema autónomo por lo tanto coincidirá con el especificado al inicio de la configuración de BGP.

Tomando como ejemplo a R3, se deben establecer tres entradas acorde con el diseño de red que se ha realizado: dos de ellas establecerán una conexión iBGP con R1 y R2, que existen dentro del AS 1, y una última entrada establecerá una conexión eBGP con R4, que se encuentra en el AS 2. Para las dos primeras entradas, se indicarán las direcciones 10.0.1.2 y 10.0.3.2, y el AS al que pertenecen, es decir, el AS 1, mientras que para la última se indicará la dirección 10.0.0.1 y su pertenencia al AS 2 (Figura 5.19).

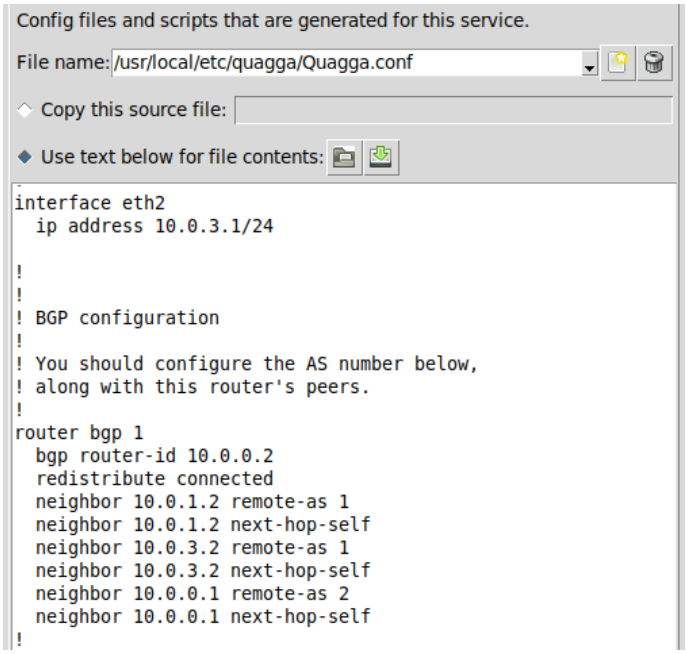

Figura 5.18: Configuración de BGP para R3.

En el caso de que un router esté conectado a una red remota, es conveniente añadir una entrada adicional con el prefijo de dicha red, para que de ese modo se anuncie al resto de vecinos. Esto se puede hacer con el comando *network*. Por ejemplo, para el router R7 se debería añadir la entrada "*network 10.0.10.0 mask 255.255.255.0*".

Una vez se ponga en funcionamiento la emulación, se puede comprobar el estado de la red realizando un *traceroute* a través de la herramienta de dos nodos. Este método permite observar de forma gráfica que la red funciona de forma correcta, puesto que el *traceroute* realizado entre los nodos PC3 y PC4, pertenecientes a distintos AS, se ha completado con éxito, tal y como se puede observar en la figura 5.20.

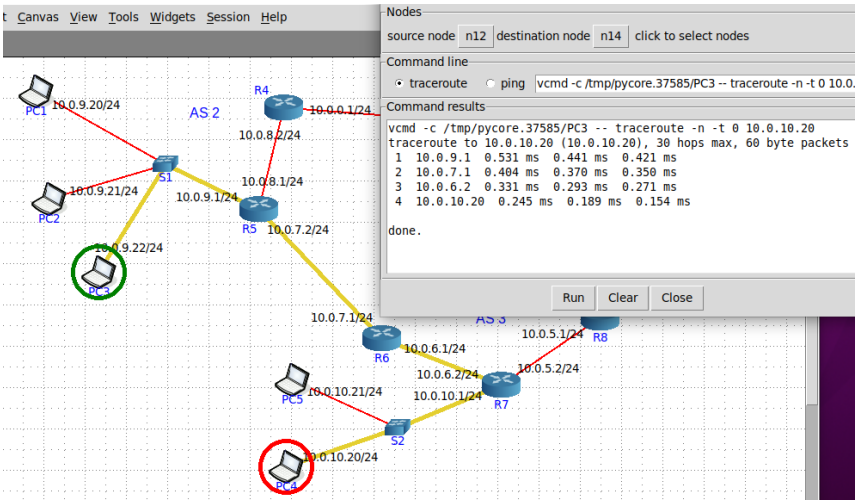

Figura 5.19: Traceroute entre PC3 y PC4.

A continuación, mediante la herramienta vtysh se puede, una vez más, consultar la tabla de encaminamiento resultante en uno de los nodos mediante el comando "*show ip route*". En este caso se ha esogido a R5 (Figura 5.21).

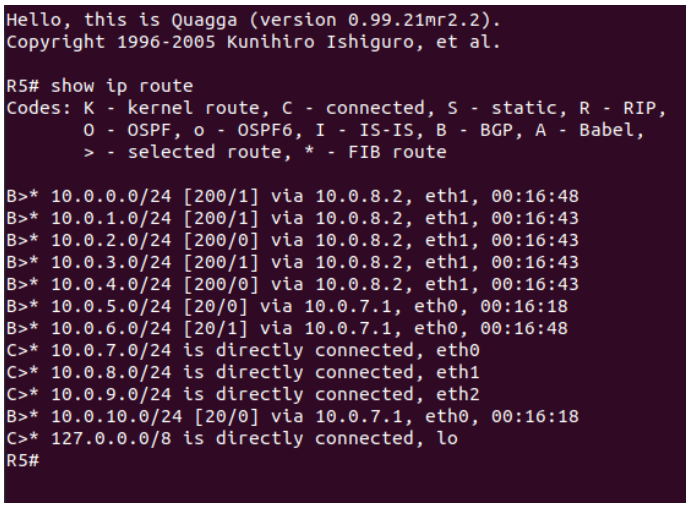

Figura 5.20: Tabla de encaminamiento de R5.

En la tabla se pueden encontrar las direcciones que están directamente conectadas a R5 señaladas con la letra C, así como las que se han aprendido mediante el protocolo BGP, que comprenden el resto de direcciones de la red. Se muestra además la distancia administrativa, que en este caso es de 200 si se ha utilizado BGP interno, o por el contrario 20 si se ha utilizado BGP externo.

A diferencia de lo que ocurría en el caso de OSPF, donde se podía observar la información de los routers vecinos mediante *widgets*, CORE no proporciona ningún tipo de *widget* para obtener información acerca de BGP de manera predeterminada. Para obtener información acerca de los *peers* de un router, puede utilizarse el comando "*show ip bgp summary*" desde la ventana de vtysh. De nuevo se ha elegido al router R5 para la demostración (Figura 5.22).

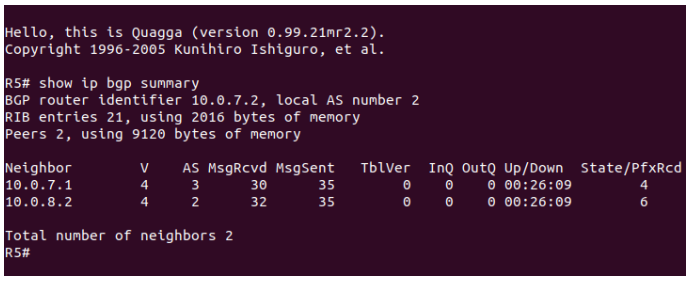

Figura 5.21: Información de los peers de R5.

Como se observa en la figura, la información obtenida es la esperada: el router 5 pertenece al sistema autónomo 2 y tiene dos *peers*, que son el router 4 y el router 6. Uno de ellos, el router 6, se encuentra en el sistema autónomo 2, mientras que el otro se encuentra en el sistema autónomo 3.

Aunque CORE no disponga de un *widget* para observar el estado de los *peers* BGP, sí que permite el diseño de *widgets* personalizados, por lo que, una vez comprobado que tanto el comando

como la red funcionan del modo esperado, se va a proceder a la creación de un nuevo *widget* (Figura 5.23).

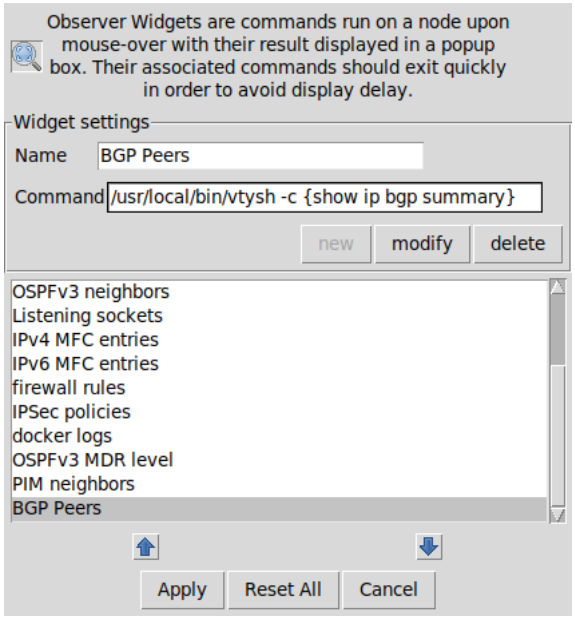

Figura 5.22: Creación del widget para BGP.

El nuevo *widget*, al que se ha llamado "*BGP Peers*", permite observar de forma sencilla el estado de todos los peers de la red, como se demuestra en la Figura 5.24, donde se pueden visualizar los tres *peers* de R2.

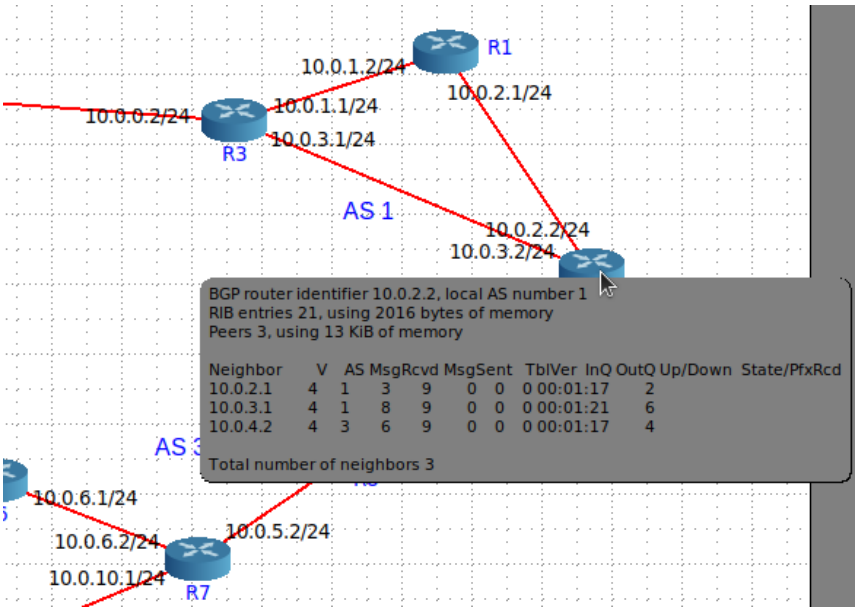

Figura 5.23: Estado de los peers de R2 mediante el uso de un widget personalizado.

Para finalizar con BGP se va a describir, con ayuda de Wireshark, el proceso mediante el cual

se intercambia la información de encaminamiento entre nodos. Para ello se va a capturar y analizar el tráfico transmitido entre los routers R3 y R4.

- Estado Connect: Este estado se inicia con un *handshake* en el cual se envían tres paquetes TCP con dirección de puerto 179 entre los dos routers. Una vez se ha completado el *handshake*, el router 3 envía un mensaje OPEN al router 4. El mensaje OPEN incluye información como el AS al que pertenece el router, la versión de BGP utilizada, o el ID del router. En el ejemplo realizado se muestra que R3 pertenece al AS 1, tiene como ID 10.0.0.2, y la versión de BGP utilizada es la 4 (Figura 5.25).

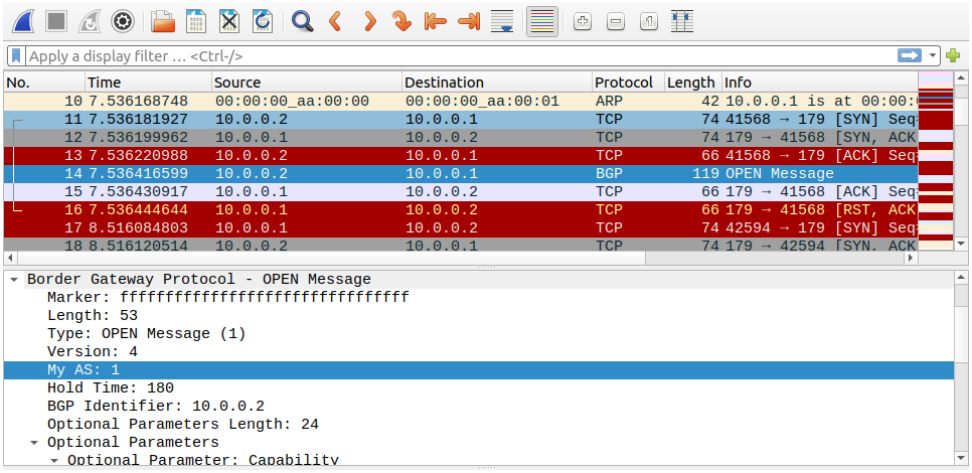

Figura 5.24: Mensaje BGP OPEN.

- Estado Active: En este estado se realizará otro *handshake* de tres paquetes TCP, pero esta vez en sentido contrario. Es decir, si antes era el router 3 el que envió el primer paquete, esta vez será el router 4 el que inicie el *handshake*. Una vez terminado, el router 4 enviará el mensaje OPEN al router 3 y se pasará al estado OpenSent.

- Estado OpenSent: En el estado OpenSent se compara información de los mensajes OPEN tales como la versión de BGP utilizada para comprobar si los routers son compatibles. Una vez se comprueba la compatibilidad, el estado pasa a OpenConfirm.

- Estado OpenConfirm: En este estado, el router espera hasta recibir un mensaje KEEPALIVE. Una vez se recibe un KEEPALIVE (Figura 5.26), se pasa al estado Established, donde queda establecida la sesión BGP.

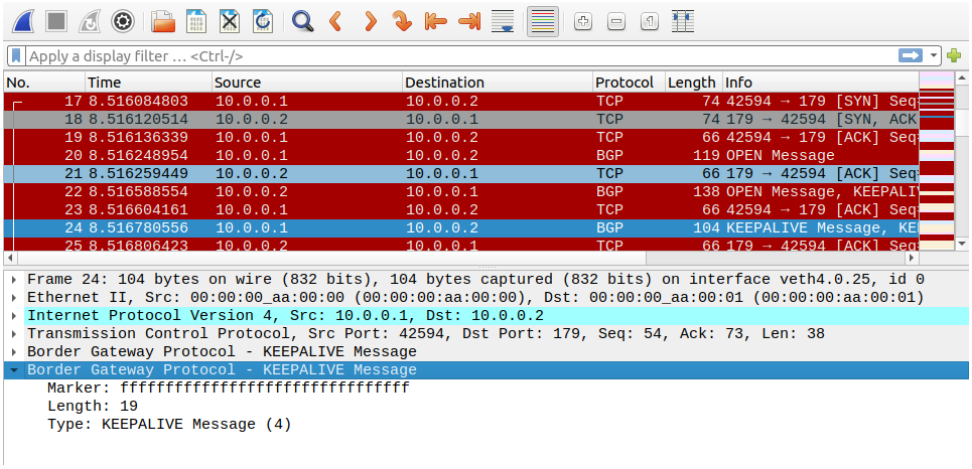

Figura 5.25: Recepción del mensaje KEEPALIVE.

- Estado Established: En este estado se procede al envío de mensajes UPDATE, los cuales incluyen información como los prefijos anunciados. En la Figura 5.27 se muestra un mensaje UPDATE con origen del router 4 hasta el router 3. En este mensaje aparecen los prefijos 10.0.0.0/24 y 10.0.8.0/24 anunciados.

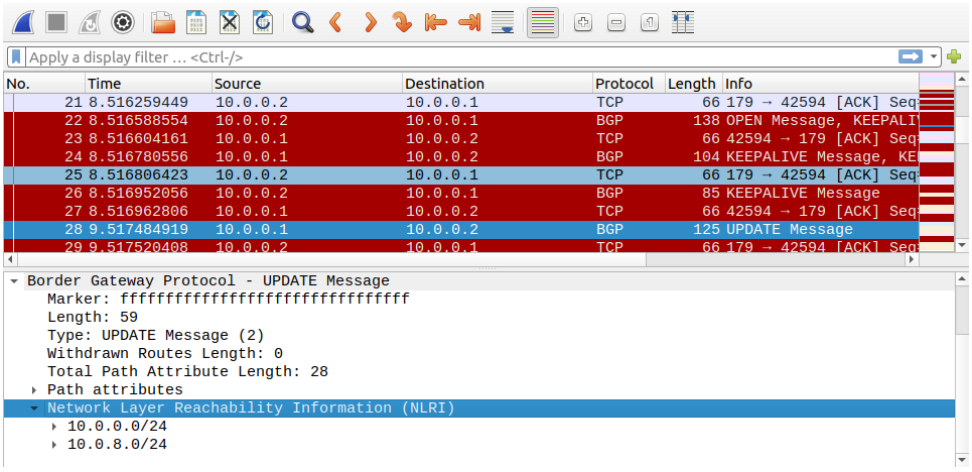

Figura 5.26: Mensaje UPDATE.

5.2.4. Virtual Private Network

Para el siguiente escenario se va a configurar una red privada virtual (VPN) utilizando OpenVPN. La VPN permitirá que el tráfico transmitido o recibido viaje a través de un túnel donde los paquetes irán encapsulados y encriptados, evitando de este modo que se pueda acceder a su contenido por terceros.

Para la configuración de este servicio se ha empleado uno de los diseños de red proporcionados por CORE, que consiste en un cliente VPN conectado a una red privada por medio de un servidor VPN (Figura 5.28).

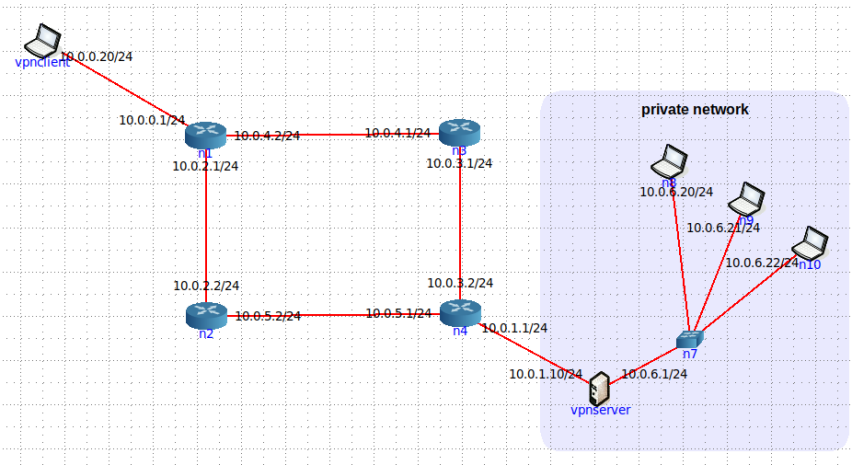

Figura 5.27: Diseño de red de VPN.

En este ejemplo el cliente y servidor tienen activado el servicio de VPNClient y VPNServer respectivamente, que a su vez contienen ficheros de configuración que se modificarán más adelante.

En primer lugar, y para hacer uso de este servicio, se debe generar una infraestructura de clave pública (PKI), sumado a una serie de certificados y claves que son los siguientes:

- Un certificado público de la Autoridad de Certificación (al que se ha llamado ca-cert.pem).
- Un fichero de parámetros Diffie-Hellman (dh2048.pem).
- Un certificado público y una clave privada tanto para el servidor como para el cliente (vpnclient.key, vpnclient.pem, vpnserver.key y vpnserver.pem).

Para generar el par de claves y certificados se ha empleado *easy-rsa*, una utilidad de línea de comandos.

Una vez se hayan generado los archivos mostrados en la figura 5.29, se debe especificar el directorio donde se hayan almacenado en el fichero de configuración del servidor y del cliente VPN. En concreto donde se indican los parámetros de *certdir* y *keydir* en dicho fichero.

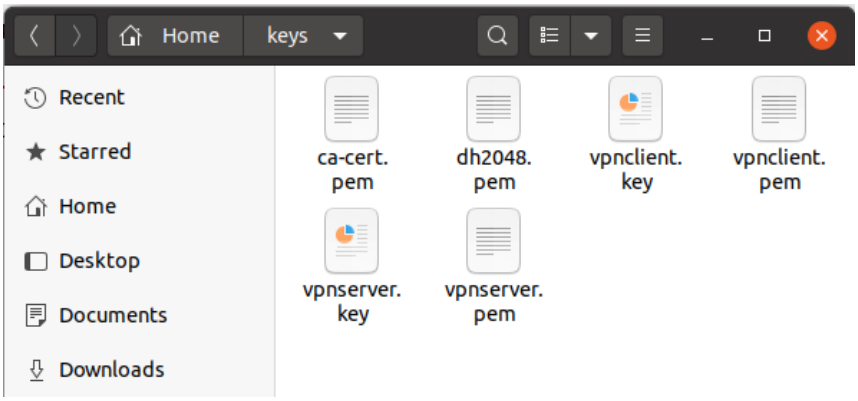

Figura 5.28: Lista de claves y certificados.

Después de realizar los cambios en los ficheros de configuración, el servicio ya está listo para su funcionamiento, sin embargo, al iniciar la emulación se observa mediante el *widget* de procesos de la herramienta de observación que, mientras que el servicio de openvpn está activo en el cliente VPN, el servicio del servidor VPN no ha sido inicializado, y por lo tanto la red no funciona (Figura 5.30).

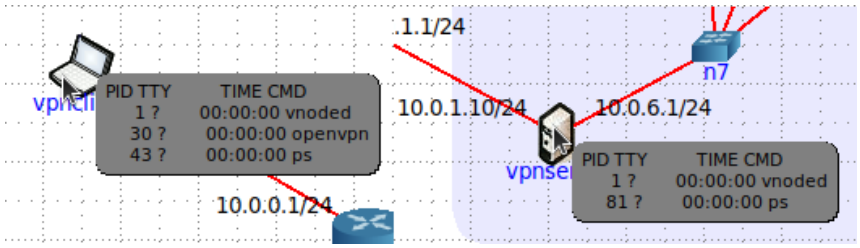

Figura 5.29: Estado del servicio openvpn en el cliente y en el servidor.

Después de realizar varias pruebas con los scripts de configuración del cliente y del servidor, se llega a la conclusión de que existe un error en dos de las líneas del *script* correspondiente al servidor (vpnserver.sh), que se incluye con el escenario de ejemplo de CORE 7.2.0.

El error se encuentra en las líneas que incluyen el comando "*push*" de configuración de OpenVPN. Los argumentos del comando "*push*" deben ir entrecomillados, sin embargo los de la plantilla de configuración no utilizan comillas. Una vez que se han realizado los cambios pertinentes, mostrados en la figura 5.31, la emulación arranca correctamente y los servicios de VPN se activan tanto en el cliente como en el servidor.

```
# openvpn server config
local $vpnserver
server $vpnsubnet 255.255.255.0
push redirect-gateway def1
EOF
)> $PWD/server.conf

# add routes to VPN server private subnets, and push these routes to clients
for privatenet in $privatenets; do
    if [ $privatenet != "" ]; then
        net=${privatenet%%,*}
        nexthop=${privatenet##*,}
        if [ $checkip = "0" ] &&
           [ "$(sipcalc "$net" "$nexthop" | grep ERR)" != "" ]; then
            echo "ERROR: invalid vpn server private net address \
$net or $nexthop " >> $PWD/vpnserver.log
        fi
        echo push route $net 255.255.255.0 >> $PWD/server.conf
    fi
done
```

```
# openvpn server config
local $vpnserver
server $vpnsubnet 255.255.255.0
push "redirect-gateway def1"
EOF
)> $PWD/server.conf

# add routes to VPN server private subnets, and push these routes to clients
for privatenet in $privatenets; do
    if [ $privatenet != "" ]; then
        net=${privatenet%%,*}
        nexthop=${privatenet##*,}
        if [ $checkip = "0" ] &&
           [ "$(sipcalc "$net" "$nexthop" | grep ERR)" != "" ]; then
            echo "ERROR: invalid vpn server private net address \
$net or $nexthop " >> $PWD/vpnserver.log
        fi
        echo push \"route $net 255.255.255.0\" >> $PWD/server.conf
    fi
done
```

Figura 5.30: Modificación del script del servidor VPN (Antes y después).

Si se observan las interfaces de red del cliente VPN, se muestra la interfaz tun0, indicadora de que el túnel se ha establecido con éxito (Figura 5.32).

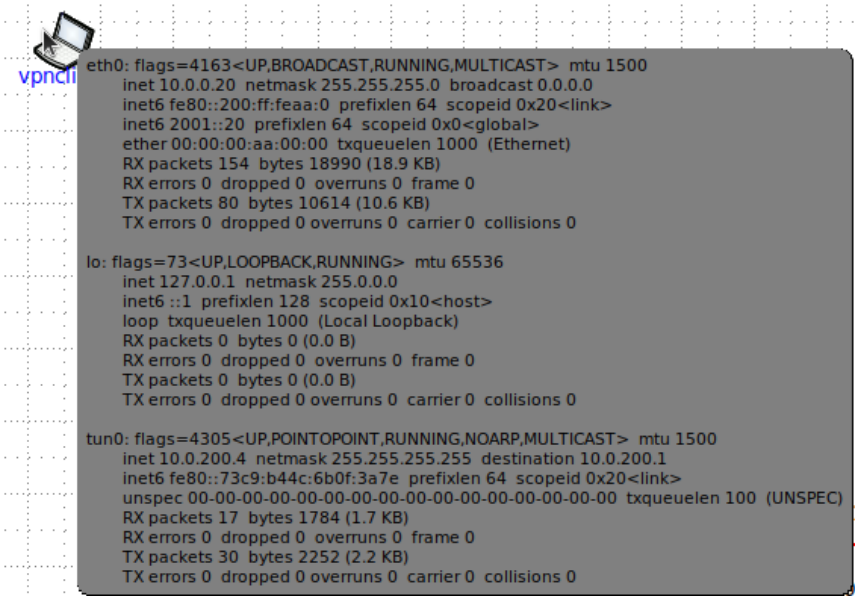

Figura 5.31: Interfaces de red del cliente VPN.

Para finalizar con las redes privadas virtuales, se ha añadido un nuevo servidor HTTP a la red. Dicho servidor contiene un documento HTML que muestra ”Hola mundo!” cuando se accede a él. Con ayuda de Wireshark, se van a realizar dos ejemplos: en el primero se apagará el servicio de VPN y se accederá al documento del servidor HTTP, mientras que en el segundo se volverá a poner en funcionamiento el servicio de VPN y se procederá de forma similar al primer ejemplo.

- Primer ejemplo: Para este caso, gracias al navegador web con interfaz de línea de comandos w3m, se ha accedido a la página web alojada en el servidor HTTP situado en la red pública. Cuando se accede al contenido de la página, Wireshark muestra una serie de paquetes TCP y HTTP y permite al usuario observar la información que se ha transmitido, ya que viaja en texto plano por la red (Figura 5.33).

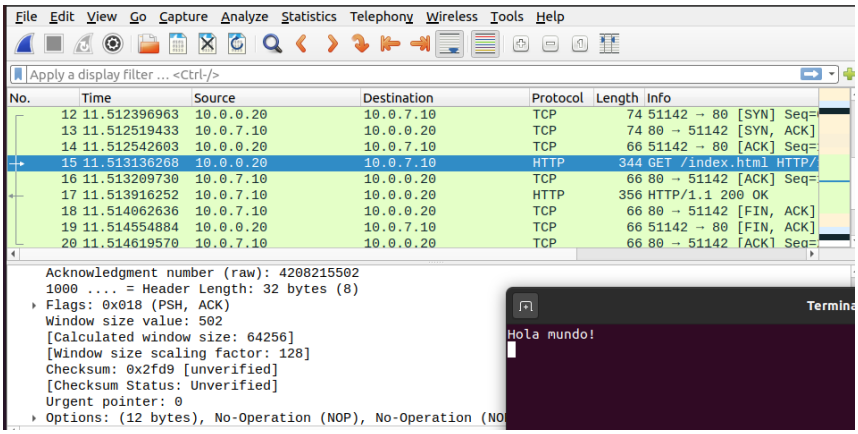

Figura 5.32: Intercambio de paquetes TCP y HTTP.

- Segundo ejemplo: En este caso, se realiza la conexión entre el cliente y un servidor HTTP

que se encuentra dentro de la red privada. La conexión con el servidor se realiza de manera satisfactoria, y el contenido de la página es mostrado en la ventana del terminal, pero al observar la captura de Wireshark, esta muestra paquetes OpenVPN en lugar de los HTTP del ejemplo anterior. Los paquetes OpenVPN no muestran ningún tipo de información relacionada con el servidor web, y por lo tanto se demuestra que efectivamente los paquetes van encriptados y encapsulados (Figura 5.34).

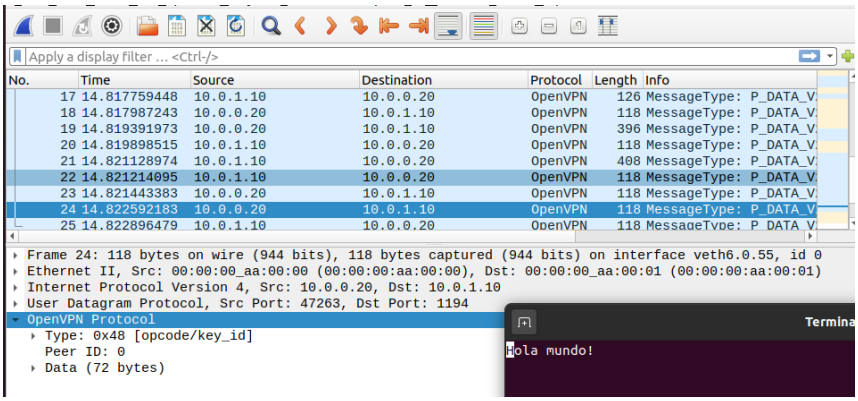

Figura 5.33: Captura de paquetes del servidor HTTP privado.

Capítulo 6

Conclusión y Líneas futuras

6.1. Conclusión

En este proyecto se ha estudiado el emulador de redes CORE, desde su arquitectura hasta cada una de las herramientas que incluye para la manipulación de la red.

Para realizar este estudio, se ha comenzado en primer lugar por las bases y los casos de uso de los emuladores de redes, con el fin de ayudar a comprender la necesidad de los mismos. Además, se han listado algunas de las diferencias entre un emulador y un simulador de redes, términos que a menudo se confunden.

A continuación, se han estudiado los distintos elementos que componen al emulador CORE: tanto las utilidades que no forman parte realmente del emulador, pero que utiliza para ampliar su repertorio de posibilidades, como las herramientas que se encuentran disponibles a través de su interfaz gráfica y el resto de clientes.

Finalmente se han configurado unos pocos de los servicios que CORE pone a disposición del usuario, y se ha comprobado el correcto funcionamiento de dichos servicios, demostrando por lo tanto que se trata de un *software* capaz, y que supone una muy buena alternativa si se quiere disponer de un emulador de redes para un sistema operativo Linux.

No obstante, hay algunos de los aspectos que se podrían mejorar. CORE destaca por su interfaz gráfica de sencillo manejo, y si bien incluye una herramienta capaz de gestionar los nodos desplegados a través de una interfaz de línea de comandos, su funcionalidad es mucho más limitada que la que ofrece la interfaz gráfica. Además, esta herramienta carece de una opción para construir los enlaces entre los nodos. De este modo, se debe recurrir al uso de la interfaz gráfica a la hora de diseñar una nueva topología de red, y si bien este método funciona de maravilla cuando se necesita crear una topología de red pequeña, si se desea construir en su lugar una topología con cientos de nodos, realizar esta tarea a través de la GUI podría resultar un tanto tedioso.

Otro de los aspectos negativos de CORE es que algunos de sus elementos, como por ejemplo el switch, carecen de opciones de configuración, y por lo tanto se debe acceder a herramientas externas al emulador para realizar tareas tales como la creación de VLAN.

Pese a esto, CORE sigue siendo un programa con mucho potencial, flexible ya que deja al usuario la opción de escoger los servicios que se desean activar permitiendo, o no, la instalación

del software pertinente en función de cada necesidad, y capaz de replicar un enorme número de escenarios de red gracias a la gran cantidad de servicios que soporta de forma nativa. Este emulador sin duda podría estar perfectamente a la altura si se comparase con otros emuladores de redes alternativos.

6.2. Líneas futuras

Este trabajo se ha centrado en la configuración de algunos de los servicios de los que dispone el emulador mediante el uso de topologías de red pequeñas. De aquí derivan dos posibles caminos: En primer lugar podría continuarse con el estudio de los servicios que no se han tratado en esta memoria, explorándose además la opción de crear nuevos servicios utilizando para ello el lenguaje de programación Python.

En segundo lugar se podría desplegar una red de gran tamaño, donde se podría más adelante hacer mediciones tales como el consumo de recursos del sistema utilizados por el emulador en función del número de nodos que se haya desplegado. Aquí se podría explorar también la herramienta Tunnel de CORE, que permite enlazar múltiples instancias del programa, donde cada una de ellas poseería una parte de la red.

Además, debido a la naturaleza del *software*, con el paso del tiempo se irán añadiendo nuevas funcionalidades, dando pie a nuevos trabajos de investigación que se puedan centrar en dichas funcionalidades.

Bibliografía

[1] Hangbin Liu. "Introduction to Linux Interfaces". En: (2018). URL: https://developers.redhat.com/blog/2018/10/22.

[2] Zak Cole. "Network Simulation or Emulation?" En: (2017). URL: https://www.networkworld.com/article/3227076/network-simulation-or-emulation.html.

[3] Damian Garros. "Introduction to Network Emulation". En: (2019). URL: https://blog.networktocode.com/post/Network_Emulation/.

[4] Debian Wiki. "Network Bridging". En: (2020). URL: https://wiki.debian.org/BridgeNetworkConnections.

[5] "Quagga vs. BIRD vs. ExaBGP". En: (2018). URL: https://www.bizety.com/2018/09/04/bgp-open-source-tools-quagga-vs-bird-vs-exabgp/.

[6] Paul Menage. "Control Groups". En: (2004). URL: https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html.

[7] The Linux Foundation. "Linux Kernel Networking". En: (2016). URL: https://wiki.linuxfoundation.org/networking/start.

[8] Kunihiro Ishiguro. "Quagga Routing Suite Documentation". En: (2017). URL: https://www.quagga.net/.

[9] Arch Linux Wiki. "Linux Containers". En: (2020). URL: https://wiki.archlinux.org/index.php/Linux_Containers.

[10] Michael Kerrisk. "Linux Man Pages Project". En: (2021). URL: https://man7.org/linux/man-pages/index.html.

[11] Jeff Ahrenholz. "Common Open Research Emulator Documentation". En: (2020). URL: https://coreemu.github.io/core/.

[12] CISCO Support. "OSPF Neighbor States". En: (2014). URL: https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/13685-13.html.