Estudio inicial del softroute XORP (eXtensible Open Route Platform) como herramienta de investigación en redes de comunicación.

Diego Augusto Pinzón Carreño.

Proyecto de grado para optar al título de Ingeniero Electrónico

Asesor
Pdo. Jhon Jairo Padilla Aguilar.
Coordinador Grupo de Investigación en Telecomunicaciones
Universidad Pontificia Bolivariana Seccional Bucaramanga.

Universidad Pontificia Bolivariana Seccional Bucaramanga.
Facultad Ingeniería Electrónica
2011

RESUMEN

TITULO: Estudio inicial del softroute XORP (eXtensible Open Route Platform) como herramienta de investigación en redes de comunicación.
FACULTAD: Ingeniería Electrónica y Administración.
DIRECTOR: Jhon Jairo Padilla
PALABRAS CLAVES: XORP, OSPF, BGP, RIP, VirtualBox, enrutamiento, virtualización.

El presente trabajo de grado realiza un estudio inicial que describe al *software* de enrutamiento XORP (eXtensible Open Router Platform) como herramienta para ser utilizada en la investigación de redes específicamente en el campo de enrutamiento.
La arquitectura Modular de XORP le permite ser Extensible. Característica que lo hace sobresalir de otras aplicaciones que compiten en el mismo espacio de enrutamiento tanto en *software* propietario como libre.

El autor del presente trabajo detalló la instalación no sólo desde el código fuente, sino desde otras herramientas que proveen los equipos de desarrollo XORP.ct y XORP.inc, como es el *CD-live*.

Se estudio XORP desde dos puntos de vista, debido a las capacidades de este *software* para ejecutar protocolos de enrutamiento avanzado. El primer punto de vista es el operativo; ejecutando protocolos de enrutamiento *Unicast* con herramientas de Virtualización. Como son *Oracle VirtualBox* y *Universal Driver* TUN/TAP, creando una mini red de dos enrutadores y un cliente por enrutador.

El segundo punto de vista del estudio de XORP fue el de desarrollador. Para ello, se buscó proveer los conceptos necesarios para que un usuario, con un perfil adecuado, se instruya en las especificaciones sobre los módulos de XORP, que tiene relación en la creación de nuevos procesos dentro de XORP, preparando así al lector para iniciar el proceso de desarrollo.

También se hace la descripción de cómo crear un proceso en XORP y el comportamiento de un Módulo cuando es activado. Estos son temas para tener en cuenta al desarrollar un proceso en XORP. Luego se presenta un resumen de documentos encontrados en internet acerca de la utilización de XORP en proyectos de grado, así como proyectos investigativos, modificando y extendiendo un módulo para un fin particular.

ABSTRACT

TITLE: Initial study of the XORP router software (eXtensible Open Router Platform) as a research tool in communication networks.
FACULTY: Electronics Engineering and administration**.**
DIRECTOR: Jhon Jairo Padilla
KEYWORDS: Soft Router, XORP, OSPF, BGP, VirtualBox, RIP, routing, virtualization.

This project developed an initial study describing the routing software known as XORP (eXtensible Open Router Platform) which is a tool used in network research specifically in the field of packet routing.
XORP's modular architecture allows it to be extensible. Feature that makes it stand out from other applications that compete in the same routing space in both proprietary and free software.

The author of this work not only details the installation from source, but from other tools that provide development teams XORP.inc and XORP.ct such as CD-live.

XORP was studied from two points of view, because the capabilities of this software to perform advanced routing protocols. The first view is the operation. To perform several tasks, several modules are necessary, such as Unicast routing protocols running with virtualization tools. Also, Oracle VirtualBox and Universal Driver TUN / TAP are used, creating a mini network between two routers and a client router.

The second point of view of the study is intended for XORP developers. Thus, we provide the necessary concepts for a user with a suitable profile, which is instructed in the specifications of the XORP modules; we describe how is the creation of new processes in XORP, preparing the reader to start the development process.

Description of how to create an XORP process and the behavior of a module when it is activated, are topics to consider for developing an XORP process. Following, there is a summary of documents found on the internet about the use of XORP projects such as research projects, and how to modify and extend a module for a particular purpose.

INTRODUCCIÓN

Para los desarrolladores principales de XORP, creadores de la compañía XORP.inc, fijaron como objetivo que, XORP sea utilizado como una herramienta para cubrir el espacio que hay en el proceso de desarrollo de nuevos algoritmos de enrutamiento o modificación de los existentes y tener la facilidad de probarlos en un ambiente de producción. Es por eso que iniciar un estudio a XORP es importante.
Describiéndolo en dos modos, uno operacional y otro desde el punto de vista del desarrollador.
En la descripción del modo operacional se ejecutaron protocolos de enrutamiento *Unicast.* Explicando paso a paso como ejecutar los mismos, sobre una plataforma de emulación en una red que costa de dos enrutadores cada uno con un cliente.
En el proceso de emulación de una red se utilizaron herramientas de Virtualización como Oracle VirtualBox y *Universal TUN/TAP Drivers.*

Para el modo de desarrollador, detallando el procedimiento para la creación de un nuevo proceso en XORP, describiendo el perfil del usuario que se encargaría de crear un proceso y conceptos que encierran la creación de un nuevo Módulo.

XORP es muy bien documentado, cada Módulo desarrollado en XORP tiene una literatura que especifica el funcionamiento, haciendo énfasis en una diagramación en bloque, detallando que librerías utilizar para extender el Módulo. Se escogió describir el funcionamiento de algunos de los Módulos de XORP, aquellos que tienen conceptos relacionados con la creación de un nuevo Módulo.

Durante el proceso de documentar la instalación y pruebas de protocolos de enrutamiento a XORP, se encontraron dos proyectos uno perteneciente a la compañía XORP.inc creada por los desarrolladores principales y otro perteneciente al desarrollador independiente Been Greear llamado XORP.ct.
Se decidió utilizar la versión 1.5 -XORP.inc, una versión anterior a la ultima liberación de la compañía para la ejecución de los protocolos enrutamiento *Unicast.*
Para la instalación se decidió documentar la versión de 1.8-XORP.ct, por que durante el proceso de transición y luego terminación de la compañía XORP.inc no había un conceso de quien o quienes iban a tomar el control del código y el mejor candidato para quedar a cargo de XORP es Been Greear. Este cambio de mantenedor del código podría definir que será del proyecto XORP en los próximos años, ya que, el desarrollador independiente podría tener otros objetivos con el proyecto.

La principal fuente de información para lograr ejecutar los protocolos de enrutamiento *Unicast* es el manual oficial de usuarios de XORP en la versión 1.6 y anteriores*.* Se decidió utilizar métodos de virtualización para emular las

capacidades de enrutamiento *Unicast* de XORP. Logrando capturar y describir los mensajes de comunicación que hacen parte de los protocolos *Unicast* de enrutamiento (Enrutamiento Estático, RIP, OSPF, BGP), por medio de la herramienta de auditoria de redes llamada *wireshark,* también se implementaron algunas políticas muy básicas para BGP.

Antes y durante el proceso de terminación de la compañía de XORP.inc, la documentación para lograr describir los pasos en la creación de un nuevo Módulo eran la versión oficial 1.6; aunque sin actualizar, presenta principios básicos en la creación de un proceso.
Además para las versiones de la compañía XORP.inc se encontró dos tesis que documentan la creación de un Módulo para un fin en particular.
Para las últimas versiones de XORP como son las versiones 1.8.3-XORP.ct se encuentra una página Web en formato Wiki, donde describen el procedimiento para la creación de un nuevo proceso.
Otro lugar muy importante a recurrir para obtener información de XORP, después de la documentación oficial es el servidor de correo electrónico a cual pertenecen usuarios y desarrolladores de XORP.
Después de estudiar las anteriores fuentes de información se creo un compendio en procedimientos para la creación de un Módulo, perfil del usuario encargado para la creación de un proceso, como opera un Módulo cuando es activado en XORP y literatura académica que utiliza a XORP como alternativa de investigación para redes.

Metodología

En el presente trabajo se tuvo encuentra que el proceso de estudio a un *software* de enrutamiento de licenciamiento GPL con lleva una serie de inconvenientes no por el tipo de licencia sino por que la mayoría del software libre en especial referente a redes tiene la falencia de pobre documentación. Pero no es el caso de XORP, que es su página principal WEB y dentro del paquete de instalación tiene la documentación necesaria para conocer este *software.*

La principal fuente de búsqueda de información para XORP es internet, exactamente la página Web oficial del proyecto XORP.

Se inicio con la recopilación de la información en formato pdf y una intensa búsqueda sobre documentos relacionados en el proceso de creación de un Módulo.

Se encontró dos documentos de trabajos académicos para titulación, relacionados con la creación de un Módulo, tomados en cuenta para describir el capitulo 4.

Otro lugar donde se puede recurrir a información es al servidor de correo del proyecto XORP. Este servidor tiene dos cuentas uno para usuarios y otro para desarrolladores.

Mayoría de proyectos de *software* libre tiene un foro o un servidor de correo para solicitar respuesta a posibles problemas o pedidos de mejoras al *software*. Allí se hizo una extensa búsqueda de datos relacionados por temas específicos.

Para documentar la instalación se hizo primeramente con la última versión de XORP la 1.6 de la compañía XORP.inc, en un fedora 10.

Lastimosamente se reporto a los desarrolladores una incompleta instalación.

Y como respuesta se dieron cuenta de la falta de interés del público hacia XORP, ya que la liberación del último código se había hecho hace seis meses desde que hice la primera instalación, nadie había documentado el error para la última versión Liberada.

Se modifico el *script* de instalación para realizar la instalación, por recomendación de un correo que se encontró el servidor de XORP citando el mismo error en versiones previas.

Ya instalado se ejecuto XORP con los mínimos requerimientos, sin presentar inconvenientes. Instalación 1.6-XORP.inc

Con la documentación adquirida de XORP.inc, más la que reside en el servidor de correo se logró entender y describir en formato texto, como funciona XORP en la

operación de comunicación entre Módulos y los archivos involucrados en el proceso. Fue lo que mas tomo tiempo.

Se recibieron noticias de los desarrolladores principales de XORP.inc, la terminación de la compañía y del soporte del código. Durante la transición y terminación de la compañía el desarrollador independiente creador de la compañía CANDELATECH y un buen contribuidor de código durante las liberaciones tempranas de XORP, le cedieron el proyecto XORP.inc.
Se procedió a documentar la instalación de 1.8XORP.ct, con un buen número de *bugs* pero resueltos.

Examinando la documentación oficial se encuentra un manual de usuarios de como ejecutar los protocolos de enrutamiento en XORP.inc. Fuente principal de consulta que se utilizó para las prácticas de XORP.
Las prácticas se utilizaron la versión 1.5 de XORP.inc. Se utilizaron técnicas de Virtualización para emular una pequeña red dos enrutadores y un cliente por enrutador. Capturando los paquetes de comunicación de los protocolos de enrutamiento *Unicast* con la utilidad de auditoria de redes *wireshark*. Todo esto en un portátil Dell Inspiron 640m con 1.5G bytes de RAM con procesador Intel core duo 7200, propiedad del autor del proyecto. El *software* que se utilizó para virtualización fue Oracle VirtualBox e Interfaces virtuales TUN/TAP. El licenciamiento de VirtualBox tiene algunas restricciones para usos comerciales pero para uso personal es totalmente libre y gratis. Para las interfaces Virtuales el *software* es código libre y gratis prácticamente sin restricciones de uso.

La documentación principal para entender como proceder en la creación de un nuevo Módulo se tuvo encuentra una página Wiki, creada dos meses antes después de la entrega del documento, actualmente propiedad de XORP.ct. También en la documentación de la desaparecida compañía XORP.inc se encuentra un pdf llamado introducción para la creación de un proceso en XORP. Esta documentación esta sin actualizar por que el proceso de compilación cambio entre las versiones XORP.inc y XORP.ct. Pero es importante leerla ya que muestran conceptos importantes para tener encuentra para el desarrollo de XORP.

CONTENIDO

MARCO TEORICO.

INTERFACES VIRUTALES TUN/TAP

Las interfaces Tun/tap son interfaces de red por *software*, significa que sólo existen en el núcleo de un sistema Operativo, actuando como interfaces de red físicas.

La diferencia entre una interface tap y tun es que la salida de la interface tap es dada totalmente en tramas *Ethernet*. En cambio para la tun la salida es dada en paquetes IP. La interface virtual puede ser temporal es decir creada, usada y destruida. Otra opción es hacer la interface persistente en ese caso es creada usando una utilidad especifica para esta tarea como *tunctl* o *openvpn -mktun*, luego los programas pueden ser atados a esta interface virtual.

Una vez la interface es creada, esta actúa como una interface física presente en el computador, significando que se le puede asignar una dirección IP, analizar trafico, reglas para el firewall, también pueden ser establecidas reglas de enrutamiento .

VIRTUALIZACIÓN

Virtualizacion es una tecnologia que particiona un computador en maquinas independientes, soportando varios sistemas operativos y aplicaciones.

El concepto de maquina Virtual surgio originalmente con el sistema VM/370 de IBM en 1972. Una VM es un duplicado de una maquina real, eficiente y aislado.
Aislado: Se pueden ejecutar varias maquinas virtuales sin interferencia de las

demas.
Eficiente: Deberia ejecutarse a una velocidad parecida a la de una maquina real.
Una de las mas grandes ventajas de esta tecnologia es de escojer un servidor que funciona a un 10% y llevarlo a la porcentajes de un 60% a 70% de trabajo, cargando multiples maquinas virtuales.
Actualmente la tecnologia de Virtualizacion esta presente a multiples niveles tanto para usuarios personales como para usuarios empresariales. Para aplicaciones personales se puede citar el caso de un desarrollador que necesita probar sus creciones a un ambiente especifico sin necesidad de requerir mas que una sola maquina.

Para el caso de empresariales la utilizacion de tecnologias de este tipo como en un datacenter, que pueden tener multiples usos en:

- Virtualización para la red.
- Virtualización para almacenaje.
- Virtualización del hardware.
- Virtualización de aplicaciones.

Se encuentran herramientas para virtualizacion tanto para software libre como software propietario. A continuacion algunas de estas tecnologias son: UML (User Mode Linux), Quemu, Vmware, VirtualBox, Xen, KVM.

EMULACIÓN

La emulación de *software* es la más fácil implementación y la menos eficiente de los tipos de virtualización. Esta forma de virtualización permite ejecutar un sistema operativo, sin modificación, como invitado en una plataforma alterna o sistema operativo. Los emuladores recrean totalmente el ambiente, incluido todas las instrucciones de CPU, dispositivo I/O, usando software. Esto permite emular diferentes arquitecturas CPU, como un *software* de Windows ejecutado sobre un no-Intel MAC

VIRTUALBOX

Como Oracle VirtualBox se define[1], en su manual oficial, como una aplicación de virtualización multiplataforma; que puede ser instalada en arquitectura de computadores INTEL o AMD, en cualquier de los siguientes sistemas operativos, Windows, Linux, MAC y Solaris, seguidamente puede extender las capacidades

1 Oracle VM,VirtualBox User ManualVersion 3.2.6.de Internet: www.virtualbox.org/manual/UserManual.html. Pag:9

del sistema operativo base, de modo que puede correr múltiples sistemas operativos al mismo tiempo. Por ejemplo en un sistema operativo Windows puede correr Linux o MAC o también puede correr Windows Server 2008 en una maquina Linux Server. Se puede instalar y correr tantas maquinas virtuales pueda, el único limite es el espacio en disco duro y la memoria RAM.
En la Figura 1 se muestra el GUI de VirtualBox.

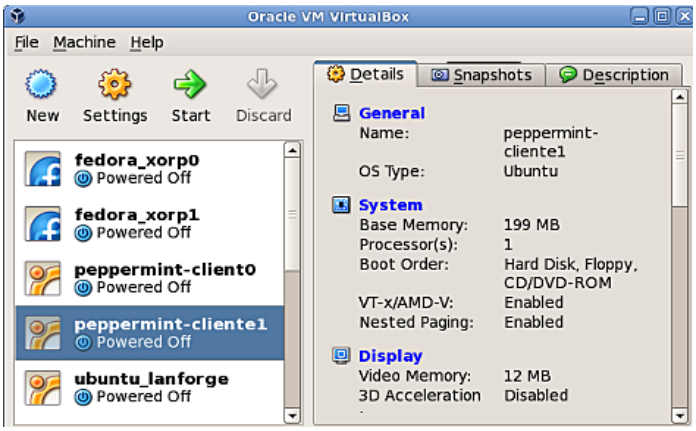


Figura 1. Interfaz grafica de VirtualBox fuente: Autor

Escenarios donde VirtualBox podría ser útil:
• Ejecutando múltiples sistemas operativos al mismo tiempo:
VirtualBox permite ejecutar múltiples sistemas al mismo tiempo, de esta forma podría un desarrollador tomar su aplicación y probarla en un Linux, MAC, Windows sin necesidad de reiniciar. También VirtualBox podría ejecutar un sistema operativo viejo como DOS ya que VirtualBox por software adapta el hardware requerido para ejecutar el sistema operativo viejo.
• Instalación de software fácilmente:
Los vendedores de software pueden utilizar toda una imagen de una maquina virtual para empacar su software evitando que el usuario tenga que instalarlo. Esto es útil cuando las instalaciones son procedimientos arduos y toman mucho tiempo.
• Pruebas y recuperación de desastre:
Una vez instalados una maquina virtual, sus discos virtuales son considerados contenedores que pueden ser congelados, activados, copiados y transportados entre clientes.
Con el uso del “snapshots” una vez establecido un estado de un disco virtual este puede ser de vuelto a un estado previo. Así el usuario puede probar instalaciones de software y caso de algo error puede volver al estado del “shapshots” establecido.

TERMINOLOGÍA

Es recomendable que un usuario de VirtualBox se sienta cómodo con términos relacionados a la virtualización de VirtualBox:

Sistema Operativo Anfitrión (Host OS): Es el sistema operativo donde se encuentra la maquina Virtual.

Maquina Virtual (VM): Es un ambiente especial que VirtualBox crea para el sistema operativo anfitrión mientras es ejecutado. Es decir que ejecuta su sistema operativo invitado dentro de una VM.

Adiciones al Invitado (guest aditions): Se refiere a software especial que es empacado en la instalación de VirtualBox, sólo que esta diseñado para el sistema invitado, con el fin de aumentar el desempeño y añadir nuevas características.

OPERACIONES EN LA RED PARA VIRTUALBOX

VirtualBox ofrece varios modos de realizar las operaciones de red, dentro de la maquina invitada[2], una de estas es *Bridges networking,* por definición utiliza el driver de la tarjeta de red del sistema huésped, en este caso una o mas *interfaces TUN* , creadas previamente, permitiendo interceptar datos de la "red física" e inyectar los datos al sistema operativo invitado, por medio de un driver llamado *netfilter,* el driver actuaría como un *switch virtua*l, habilitado por el "puente" entre la tapxx y la tarjeta de red en la maquina invitada.

En el manual de VirtualBox expone que el uso de interfaces virtuales para habilitar funciones de red en el sistema invitado ya no es necesario, pero se pueden habilitar para tener funciones de red avanzadas.[2]

"Note: Even though TAP is no longer necessary on Linux with bridged network- ing, you can still use TAP interfaces for certain advanced setups, since you can connect a VM to any host interface which could also be a TAP interface."

LINUX EN LA RED.

Las operaciones y aplicaciones de Linux en la red, son tan variadas como la cantidad de sus distribuciones Linux. Operaciones basicas en Linux como definir una direccion IP se logran con digitar el comando *ifconfig eth0 {mi_IP} {netmask}.* Encontraste operaciones avanzadas como compartir el ancho de banda de manera justa, hacer enrutamiento basado en la direccion MAC, direccion IP de origen, puerto, tipo de servicio, hora del dia o contenido; requieren un mayor adiestramiento al encargado de la administración del sistema. Pero en definitiva son algunos comandos que permiten auditar el estado de una red, una vez se hallan terminado operaciones basicas o de alta complejidad.

---

2 Oracle VM,VirtualBox User Manual.Version 3.2.6.de Internet: www.virtualbox.org/manual: /UserManual.htmlPag:103-107

Los archivos que controlan la red IP en un sistema operativo linux estan ubicados segun distribución; aqui un compendio de la ubicación de archivos que controlan funciones de red:

•Red Hat y Mandrake

Para los dispositivos de red: /etc/sysconfig/network-scripts/ifcfg-*

Para nombre del Host y definicion del Gateway pre-determinado : /etc/sysconfig/network

Para definicion de rutas estatica: /etc/sysconfig/static-routes

• Suse para versiones mayores a 8.

Para los dispositivos de red: /etc/sysconfig/network/ifcfg-*

Para definicion de rutas estatica: /etc/sysconfig/network/routes

• Debian

Definicion de interfaces de red y enrutamiento: /etc/network/interfaces

• Gentoo

Definicion de interfaces de red y enrutamiento: /etc/conf.d/net

• Slackware

Definicion de interfaces de red y enrutamiento: /etc/rc.d/rc.inet1

Los formatos para la configuración varian dependiendo de la distribucion pero las herramientas para su ejecución son las mismas.

A continuacion se mostraran algunos herramientas de auditoria de redes que pueden ayudar cuando se termina alguna operacion para la red de Linux:

• La herramienta ping diseñada para tomar ventaja del protocolo Internet Control Message Protocol (ICMP), A continuacion un ejemplo.

[Diegu@localy ~]$ ping -c 1 -n 192.168.99.254

PING 192.168.99.254 (192.168.99.254) from 192.168.99.35 : 56(84) bytes of data.

--- 192.168.99.254 ping statistics ---

1 packets transmitted, 0 packets received, 100% packet loss

PING 192.168.99.254 (192.168.99.254) from 192.168.99.35 : 56(84) bytes of data.

64 bytes from 192.168.99.254: icmp_seq=0 ttl=255 time=238 usec

• Cambiar la IP de una maquina y observar la tabla de enrutamiento local:

[Diegu@localy ~]$ **ifconfig wlan0 10.153.60.2**

[Diegu@localy ~]$ **route -n**

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.153.60.0	0.0.0.0	255.255.254.0	U	2	0		wlan0
0.0.0.0	10.153.60.1	0.0.0.0	UG	0	0		wlan0

[Diegu@localy ~]$

• Tambien con el comando ip route show muestra el autor del creador de la ruta, para el caso del ejemplo fue el kernel el autor:

[Diegu@localy ~]$ **ip route show**

10.153.60.0/23 dev wlan0 proto kernel scope link src 10.153.61.59 metric 2

default via 10.153.60.1 dev wlan0 proto static

[Diegu@localy ~]$

• Apagar una interface de red especificando el nombre:

[root@localy ~]$ **ifconfig eth0 down**

• Activar una interface de red ethernet y configurar la mascara de red:

[root@localy ~]$**ifconfig eth0 192.168.99.14 netmask 255.255.255.0 up**

[root@localy ~]$ **ifconfig eth0**

eth0 Link encap:Ethernet HWaddr 00:80:C8:F8:4A:53

inet addr:192.168.99.14 Bcast:192.168.99.255 Mask:255.255.255.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:100

RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)

• Definir una ruta por defecto:

[root@localy ~]$ **route add default gw 192.168.99.254**

• La utilidad de red arping puede generar mensajes ARP (Address Resolution Protocol), con las opciones -q -c 3 -U -I en la salida de la interface eth0 para la direccion IP 192.168.99.35

[root@localy ~]$ **arping -q -c 3 -U -I eth0 192.168.99.35**

Se utiliza tcpdump para capturar los mensajes de la utilidad de red arping.

[root@localy ~]$ **tcpdump -c 3 -nni eth2 arp**

tcpdump: listening on eth2

06:28:23.172068 arp who-has 192.168.99.35 (ff:ff:ff:ff:ff:ff) tell 192.168.99.35

06:28:24.167290 arp who-has 192.168.99.35 (ff:ff:ff:ff:ff:ff) tell 192.168.99.35

06:28:25.167250 arp who-has 192.168.99.35 (ff:ff:ff:ff:ff:ff) tell 192.168.99.35

• Muestra informacion sobre la tabla local ARP creado por el sistema Linux.

[Diegu@localy ~]$ **arp -na**

? (10.153.60.1) at 40:01:c6:68:5f:01 [ether] on wlan0

[Diegu@localy ~]$ **ip neighbor show**

10.153.60.1 dev wlan0 lladdr 40:01:c6:68:5f:01 STALE

• Para el comando netstat con las opciones – untap muestra el estado de los puertos y aplicaciones que utilizan operaciones sockets:

[Diegu@localy ~]$ **netstat -untap**

(Not all processes could be identified, non-owned process info

will not be shown, you would have to be root to see it all.)

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	127.0.0.1:631	0.0.0.0:*	LISTEN	-
tcp	0	0	::1:631	:::*	LISTEN	-
udp	0	0	0.0.0.0:46802	0.0.0.0:*	-	
udp	0	0	0.0.0.0:5353	0.0.0.0:*	-	
udp	0	0	0.0.0.0:631	0.0.0.0:*		

UTILIDAD DE RED NETCAT

Netcat es conocido como el TCP/IP "*Swiss Army knife*" por buenas razones. El propósito básico de esta utilidad es leer y escribir información a través de conexiones en la red. El código fuente de NetCat se ubica en http://netcat.sourceforge.net/.

Con la opción -e puede ejecutar un programa con los argumentos correctos para la ejecución con el fin de utilizarlo como puerta trasera en un cliente remoto.

Con la opción -l para que escuche el puerto 26.

[root@localy Diegu]# nc -l 26

Para iniciar comunicación con un cliente se necesita la dirección IP y el puerto.

[Diegu@localy ~]$ su root

Password:

[root@localy Diegu]# nc 26

usage: nc [-46DdhklnrStUuvzC] [-i interval] [-p source_port]

[-s source_ip_address] [-T ToS] [-w timeout] [-X proxy_version]

[-x proxy_address[:port]] [hostname] [port[s]]

[root@localy Diegu]# nc 127.0.0.1 26

DEFINICION SOTFWARE DE ENRUTAMIENTO

El procesamiento computacional ha aumentado a tal punto que usuarios comunes pueden tener acceso a un *software* que anteriormente era ejecutado en plataformas de *hardware* muy especiales tanto en costo como en procesamiento y adaptarlo a las necesidades de un mercado.
Actualmente en el espacio de enrutamiento por *software* existen múltiples plataformas, entre algunas están: quagga, zebra, gated, BIRD, freesco y XORP. Cada una con características de desarrollo y de diferentes capacidades de enrutamiento.

Los enrutadores por software toman ventaja de las actuales velocidades de procesamiento de computadores personales, que es necesario para ejecutar protocolos de enrutamiento. Programas llamados “demonios” de enrutamiento tienen la tarea de intercambiar información con otros sistemas para crear la tabla de enrutamiento. Durante la comunicación utilizan el procesamiento de paquetes IP que trae el sistema huésped.

1. XORP (EXTENSIBLE OPEN ROUTE PLATFORM)

Los enrutadores de software basados en hardware de PC y herramientas *Open Source*, poseen ventajas tales como la relación costo vs capacidad de procesamiento, además de una plataforma de desarrollo que les permite ser extensibles por programación y adaptables por *software* a diferentes situaciones; por el contrario, los enrutadores con hardware y software propietario, son más costosos y no permiten modificaciones del hardware y el software que los compone. Es allí donde XORP hace su aporte.

En la página Web de los desarrolladores del proyecto de XORP, se presenta una corta introducción de lo que ofrece XORP como herramienta de investigación[1]:

> “XORP (eXtensible Open Route Plattaform) es un soft route de código libre, único en la industria de enrutamiento con la característica de tener una plataforma extensible. La arquitectura modular de XORP permite una rápida incorporación de nuevos protocolos, características y

1 Ben Greear www.candelatech.com/-XORP.ct/

funcionalidades, incluyendo soporte para hardware y software en el plano de envió”

Otra ventaja de los enrutadores basados en Linux, es el bajo costo del equipo de *hardware* del PC a utilizar, escogiendo a comodidad las piezas que más se ajusten a las necesidades de capacidad de la red y su futura ampliación.

Empresas como Juniper o Cisco y otras relacionadas con el negocio de redes, tienen productos de arquitectura cerrada y no están motivadas para darle apoyo a la investigación académica, ni mucho menos ofrecer servicios de redes que se adapten a un nicho especifico diferente a las redes de uso general.

Otra gran ventaja del software libre es la adquisición de fragmentos de código o de funcionalidades de un producto determinado para utilizarlas en otro proyecto. Es el caso de VYATTA, una empresa que escogió como modelo de negocio el de Linux Red Hat, el cual provee dos productos; el primero de ellos es su producto estrella (Linux Red Hat) y el segundo producto que es desarrollado por la comunidad de usuarios, donde se prueban nuevas funcionalidades que podrían ir al producto estrella de la empresa, en el que se cobra por el soporte del servicio de documentación y entrenamiento; en este caso, el producto estrella es un Router basado en software que integra enrutamiento avanzado, conocido como VYATTA. Este producto ha incorporado y mejorado, la parte de la línea de comando de XORP, conocida como el xorpsh[2].

XORP inició su desarrollo en llegará del 2002 con la versión 0.1[3]. Desde entonces, ha tenido un ciclo de mejoras hasta febrero del 2010, donde se detuvo el desarrollo en la versión 1.6-XORP, que soporta Windows Server y macOS X. Esto significa que, si un usuario de este producto llegará a encontrar algún error de código (bug) o un funcionamiento errático, al reportarlo no tendría respuesta de los encargados del mantenimiento del software. Pero existe la posibilidad de que el mismo usuario pueda modificar el código para solucionar el mal funcionamiento, esta opción existente en proyectos de código libre que permite una constante interacción entre el desarrollador y el usuario probador, permitiendo encontrar soluciones para el operador final. Afortunadamente, la empresa Candelatech creó una nueva rama de XORP llamada 1.8. XORP-CT, creado y mantenido por Ben Greear, quien es el responsable del código fuente (ubicado en http://github.com/greearb/XORP.ct) y también se encarga de aceptar la integración de nuevos parches que pudieren llegar al código.

Los objetivos de este desarrollador son: disminuir el tamaño de la instalación de XORP-ct a 8MB, y además, la posibilidad de trabajar con procesadores ARM.

XORP opera con los protocolos BGP, OSPF, RIP, OSLR y PIM-SM. Sin embargo, el protocolo SNMP fue removido del código base de la versión 1.6-XORP. XORP

---

2 Allan Leinwand ,*30 -03-2006, Vyatta***,** XORP-users@XORP.org disponible desde internet en: http://mailman.icsi.berkeley.edu/pipermail/XORP-users/2006-March/001165.html

3 .XORP team, disponible desde internet en:www.XORP.org/news.html

también soporta la nueva versión del núcleo (kernel) de Linux 2.6.34 y compila en Linux x86_32, x86_64 y variantes de BSD[1]. También, XORP cuenta con paquetes de instalación rpm para fedora 12 y 13, además cuenta con un CD-live (contiene instalador y programa ejecutable de XORP) que ejecuta una distribución de Linux Ubuntu.

1.1 INSTALACIÓN Y PUESTA EN FUNCIONAMIENTO DE XORP

XORP es un enrutador por *software*, una vez instalado, toma el *hardware* de la maquina con sistema operativo base Linux o bsd, de un computador personal, para otorgarle facultades de operación en enrutamiento avanzado, ejecutando protocolos *Unicast (OSPF, BGP entre otros)* y *Multicas (PIM-SD),* además el diseño de la arquitectura modular de XORP permite con ayuda de una API (Application Programming Interface) crear, modificar procesos internos en XORP.

La instalación de la versión 1.8-XORP-CT se puede lograr en dos formas, una recurriendo al CD-live, que provee el equipo desarrollador de XORP-CT en la página web: http://www.candelatech.com/XORP.ct/livecd.html. Instalación descrita en el anexo B.
La segunda forma de instalación es hacerla sobre una distribución de Linux o una variante de BSD ya previamente instalada, sirviendo como huésped a XORP. Instalación descrita en el anexo A.

La ventaja que tendría la primera instalación, surge cuando se tiene una máquina para utilizarla como enrutador, pero esta no tiene una distribución Linux o una variante de BSD como sistema operativo base. Con este tipo de instalación se ahorraría mucho tiempo, ya que se tendría en un sólo paquete en forma de CD-live al sistema operativo base y la aplicación XORP, además de la posibilidad de instalar programas de auditorías para redes como wireshark, tcpdum.

Como tercera posibilidad, para ejecutar XORP esta la utilización de la versión del CD-live de la desaparecida compañía XORP.inc, la cual trabaja directamente desde el Cd-ron sin opción de instalación. Interactuando en modo texto como lo hacen los enrutadores modernos.

1.1.1 Configuración del CD-LIVE XORP.INC.

Como se menciono en la introducción de XORP, la versión XORP-1.6 propiedad de la desaparecida compañía XORP.inc, formada por los desarrolladores

---

1 Team XORP. 6-2-2010.XORP-CT is realese XORP-users@XORP.org está disponible en Internet en http://mailman.icsi.berkeley.edu/pipermail/XORP-users/2010-June/003963.html

principales también incluye una versión Cd-live (Cd-Room auto ejecutable), a diferencia de la versión XORP.ct, este corre con un *kernel* modificado y su medio de interactuar con el usuario es vía consola. La documentación actual para este *software*, hace referencia a la página electrónica principal www.xorp.org que es aplicable para la versión 1.6.
El conocimiento de esta herramienta provee al especialista de red, la posibilidad de probar la línea de comandos de XORP y la forma de configuración de los protocolos de enrutamiento, sin necesidad de instalar el *softroute* en el disco duro en la maquina anfitrión de XORP.

A continuación se describirá paso a paso la configuración y comandos para interactuar con el Cd live XORP-1.6.
Al dar ejecución al Cd live, este primero verificara si se encuentra el archivo de configuración del enrutador que tiene una extensión .boot. Generalmente busca en memoria usb y medio de almacenamiento extraíbles del computador. Si se encuentra un archivo .boot pre configurado, este programa revisara la configuración y la aplicara. Si no hay usb o no hay archivo de configuración, el *software* le dará una advertencia acerca que no podrá guardar los datos de la configuración. Exponiéndose a la perdida de datos, en caso de falla eléctrica o reinicio de la maquina anfitrión.

Seguido de esta advertencia, se pasa a la autenticación del súper usuario y del usuario XORP, para iniciar cambios en el archivo de configuración .boot. Ver figura 2,3 respectivamente (siguiente pagina). El ambiente grafico de la interacción en la línea de comando, es idéntico al funcionamiento y visualmente a la que se presenta en la versión escritorito de XORP-ct.

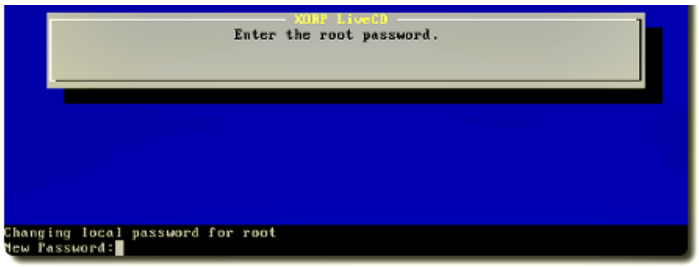


Figura 2. Configuración de la identidad del súper usuario.
Fuente: Autor

Figura 3. Creación del usuario XORP. *Fuente: Autor*

Seguidamente, se entra a la configuración de las interfaces de red a habilitar. Ver figura 4.
Allí el programa encontrara las tarjetas de red, a las cuales se validaran en la configuración del archivo de extensión .boot, simplemente se tiene que señalar, marcando las casillas a las cuales se les dará implementación, en el caso del ejemplo, incluye *loopback interface*, una interface virtual y la interface Ethernet.
Siguiente, se pasa al aviso de confirmación, al completar la configuración. Ver figura 4.

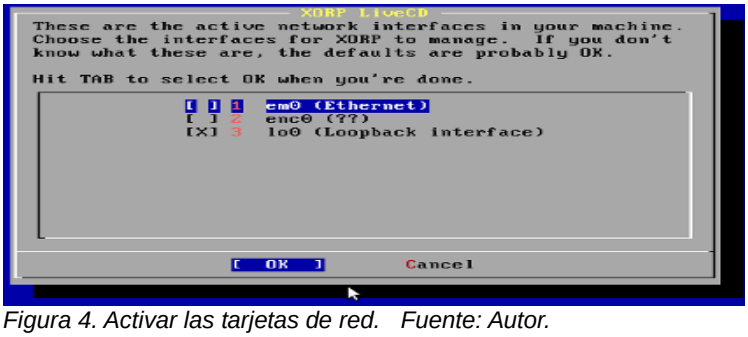


Figura 4. Activar las tarjetas de red. *Fuente: Autor.*

Cabe destacar el mensaje en consola *"Starting XORP router processes: xorp_rtrmgr":* Como se puede ver en la figura 5. El primer proceso en dar inicio en XORP es el *router manager*, el cual utiliza la configuración realizada en la figura 4 para preparar a XORP e inicializar los protocolos configurados. (siguiente Página)

Figura 5. Ventana de confirmación e inicialización del primer proceso de XORP. Fuente: Autor

Después del procesamiento del *router manager* este le presenta al usuario información del software XORP, por ejemplo como se salvara la información y como dar inicio al xorpshell (Módulo para iniciar línea de comandos). La información presentada en la figura 6 puede tener un sentido de precaución en caso de utilizar este software para un ambiente de producción.

```
FreeBSD/i386 (xorpcd.local) (ttyv0)

login: xorp
Password:
No home directory.
Logging in with home = "/".
Copyright (c) 1992-2008 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.

FreeBSD 7.0-RELEASE (LIVECD) #0: Thu Jan  8 10:13:08 PST 2009

Welcome to the XORP LiveCD!
Build date: Thu Jan  8 10:13:50 PST 2009

Use the "xorpsh" command to enter the XORP shell.

Once inside xorpsh, after configuring the router
using "configure", use "save /etc/local/xorp.conf"
to save the configuration, and "usb save" to back
up all configuration to a preformatted USB key.

Backing up may also be done from this shell, using
"usb_save". Do not unplug the key during a backup.
$
```

Figura 6. Inicio de CLI(Command Line Interface) Fuente: Autor

Al digitar en consola “xorpsh”, se dará al usuario de XORP la facultad de interactuar por medio de la línea de comandos con el softroute.
Una vez haya entrado al dominio de xorpsh, aparecerá en consola una frase de bienvenida, esto le asegura que ha entrado exitosamente a la línea de comandos de XORP. Ver figura 7 (siguiente página).

```
Welcome to the XORP LiveCD!
Build date: Thu Jan  8 10:13:50 PST 2009

Use the "xorpsh" command to enter the XORP shell.

Once inside xorpsh, after configuring the router
using "configure", use "save /etc/local/xorp.conf"
to save the configuration, and "usb save" to back
up all configuration to a preformatted USB key.

Backing up may also be done from this shell, using
"usb_save". Do not unplug the key during a backup.
$ xorpsh
Welcome to XORP on xorpcd.local
xorp@xorpcd.local>
```

Figura 7. Pantallazo de bienvenida a usuario "XORP" fuente: Autor

A partir de este momento, ya tiene una representación fiel de lo que sería el xorpsh en una maquina con una instalación completa de XORP (anexo B).
Para explorar las opciones de comandos que habilita xorpsh, con digitar el símbolo "?", la consola le responderá con una serie de alternativas con un respectivo *label* ,ver figura 8.

```
$ xorpsh
Welcome to XORP on xorpcd.local
xorp@xorpcd.local> ?
Possible completions:
  configure           Switch to configuration mode
  exit                Exit this command session
  help                Provide help with commands
  ping                Ping a hostname or IP address
  ping6               Ping an IPv6 hostname or IPv6 address
  quit                Quit this command session
  show                Display information about the system
  traceroute          Trace the IP route to a hostname or IP address
  traceroute6         Trace the IPv6 route to a hostname or IPv6 address
  usb                 -- No help available --
xorp@xorpcd.local>
```

Figura 8. Comando de ayuda y opciones del menú fuente: Autor

El xorpshell además genera una lista de los posibles comandos, como opción cuando se tiene un error en la ejecución de un comando, como se ve en la figura 20, al tratar de mostrar las interfaces de red habilitadas.

```
$ xorpsh
Welcome to XORP on xorpcd.local
xorp@xorpcd.local> show Interfaces
                        ^
syntax error, expecting `host', `interfaces', `route', or `version'
xorp@xorpcd.local> show interfaces
em0/em0: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500 speed 1 Gbps
        physical index 1
        ether 08:00:27:84:46:b4
em1/em1: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500 speed 1 Gbps
        physical index 2
        ether 08:00:27:65:aa:a9
lo0/lo0: Flags:<ENABLED,MULTICAST,LOOPBACK> mtu 16384 speed unknown
        physical index 4
xorp@xorpcd.local>
```

figura 9. Comando "show interfaces" fuente: Autor

Para configurar los procesos de enrutamiento, digitamos "*configure*", como se

muestra en la figura 10. Como confirmación en el modo de configuración, la consola le responderá con un mensaje "*Entering configuration mode*".

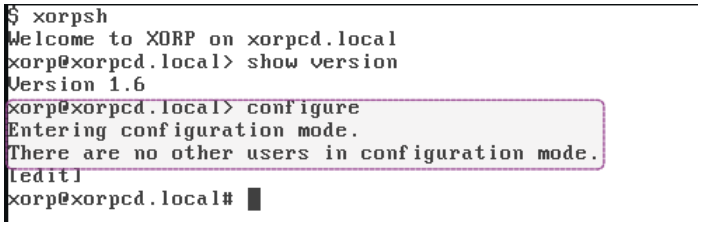


Figura 10. Modo configurar procesos Fuente: Autor.

Para configurar xorp se emplea un concepto de nodos y jerarquías, donde el criterio de cambiar algún parámetro es entrar al nodo principal que tendría la jerarquía mas alta como ejemplo sería "protocols" o "interfaces". A partir de allí el salto al siguiente sud nodo sería con un espacio y el nombre del sud nodo.
Para ver la configuración actual del enrutador, digitamos en consola "show", como se observa en la figura 11, muestra la configuración previa de las interfaces habilitadas y protocolos existentes configurados.

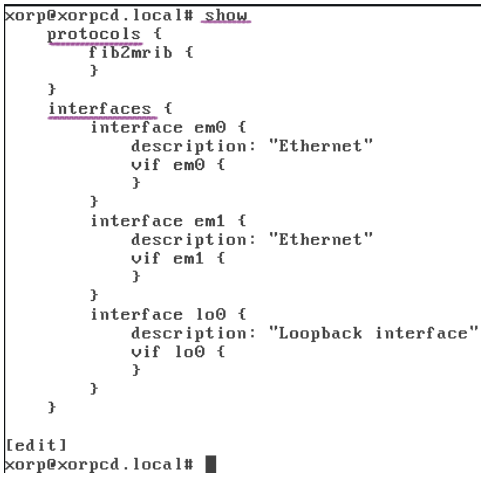


Figura 11. Comando configure y show
Fuente: Auto

Los comandos que se encuentran habilitados para la parte de configuración se pueden encontrar digitando en consola "?" . ver figura 12. (siguiente página)

```
xorp@xorpcd.local# ?
Possible completions:
  commit                Commit the current set of changes
  create                Alias for the "set" command (obsoleted)
  delete                Delete a configuration element
  edit                  Edit a sub-element
  exit                  Exit from this configuration level
  help                  Provide help with commands
  load                  Load configuration from a file
  quit                  Quit from this level
  run                   Run an operational-mode command
  save                  Save configuration to a file
  set                   Set the value of a parameter or create a new element
  show                  Show the configuration (default values may be suppressed)

  top                   Exit to top level of configuration
  up                    Exit one level of configuration
xorp@xorpcd.local#
```

Figura 12 Opciones para la configuración del archivo .boot *Fuente: Autor*

Para iniciar la configuración debida de los procesos de enrutamiento tenemos que digitar en consola “edit protocols”, esto permitirá hacerles cambios al nodo *protocols*, para así incluir un nuevo proceso. Para el caso del ejemplo de la figura 13, sería RIP (Router Information Protocol), con el comando “set RIP”. Para verificar que la configuración tuvo efecto, se digita el comando “show”.

```
$ xorpsh
Welcome to XORP on xorpcd.local
xorp@xorpcd.local> configure
Entering configuration mode.
There are no other users in configuration mode.
[edit]
xorp@xorpcd.local# edit protocols
[edit protocols]
xorp@xorpcd.local# show
    fib2mrib {
    }

[edit protocols]
xorp@xorpcd.local# set rip
[edit protocols]
xorp@xorpcd.local# show
    fib2mrib {
    }
>   rip {
>   }

[edit protocols]
xorp@xorpcd.local#
```

Figura 13 Comandos configure, edit, set, show
fuente: Autor

Para salvar los cambios a los nodos, se utiliza el comando “commit”, también está presente el comando “delete” que permite borrar nodos y “help”, como se observa en la figura 14 (siguiente pagina).

```
[edit protocols]
xorp@xorpcd.local# commit
OK
[edit protocols]
xorp@xorpcd.local# help delete

The "delete" command is used to delete a part of the configuration tree.
All configuration nodes under the node that is named in the delete command
will also be deleted.  For example, if the configuration contained:

   protocols {
       bgp {
          peer 10.0.0.1 {
             as: 65001
          }
   }

then typing "delete protocols bgp", followed by "commit" would cause bgp
to be removed from the configuration, together with the peer 10.0.0.1, etc.
In this case this would also cause the BGP routing process to be shut down.

See also the "edit" and "set" commands.

xorp@xorpcd.local#
```

Figura 14. Comandos delete, commit y help Fuente: Autor

Terminado la configuración y salvado los cambios se proceden a salir del dominio del usuario XORP, con el comando “exit”. Ver figura 15.

```
xorp@xorpcd.local# up
[edit]
xorp@xorpcd.local# exit
xorp@xorpcd.local> exit
$ exit

FreeBSD/i386 (xorpcd.local) (ttyv0)

login:
```

Figura 15. Comando "exit" Fuente: Autor

Para la utilización del Cd auto ejecutable en un ambiente de producción, estaría sujeto al soporte que tiene la detección de las tarjetas de red, la arquitectura del procesador, la memoria RAM en su velocidad y el parámetro L2 de la cache. Las tarjetas de red que puede detectar están presentes en un archivo, ubicado en /boot/defaults/ loader.conf aquí algunas de ellas:

National Semiconductor DS8390/WD83C690, Intel(R) PRO/1000 Gigabit Ethernet, Midway-based ATM interfaces, 3Com Ether link III (3c5x9), Intel Ether Express Pro/10 Ethernet, Fujitsu MB86960A/MB86965A based Ethernet, Intel Ether Express PRO/100B (82557, 82558), Sun GEM/Sun ERI/Apple GMAC.

Como recomendación, en el proceso de instalación de XORP-1.6 y versiones previas, el gran nivel de utilización en recursos de procesamiento de la maquina, por ello sería favorable mejorar la ventilación o el ambiente de instalación que sea preferiblemente a 14 C o menor.

2 INTRODUCCIÓN A XORP

En este capítulo se dará introducción a XORP, exponiendo las funcionalidades y arquitectura de este *software*. Se presenta una base inicial de conocimiento, acerca del engranaje funcional relacionados con conceptos necesarios para la creación de un nuevo proceso en XORP, como son la comunicación entre módulos, secuencia de inicio de XORP, descripción del Módulo xorpsh y el Módulo FEA (Forwarding Engine Abstrantion). Este capítulo tiene como base de información los documentos oficiales de la versión XORP-ct[1].

2.1 GENERALIDADES DE XORP

XORP (eXtensible Open Route Plattaform), es un software de enrutamiento avanzado que utiliza un conjunto de protocolos, a los cuales es posible extender por medio de una API(Application Programming Interface) totalmente escrito en c++, incorporando herramientas para su configuración. Estos protocolos trabajan a nivel del usuario en un sistema operativo Linux, el cual provee todo el andamiaje en el plano de control del enrutador, reemplazando, borrando, creando, rutas de enrutamiento, coordinando toda la base de información del enrutador, de acuerdo con los parámetros y directrices del protocolo configurado o adaptado por el usuario. XORP no tiene su propio motor de envió de paquetes, por tanto utiliza una serie de mensajes para comunicarse con el núcleo del sistema operativo Linux, el cual pone a disposición un amplio espectro de posibilidades, tales como control de tráfico, filtrado, modificación de cabeceras IP y desde luego el envió de paquetes en la capa de red. Ver figura 16. Una alternativa para el envió de paquetes y la modificación de estos es la utilización del soft ware CLICK[2], que tiene soporte para las versiones 1.5 y 1.6 de XORP.inc.

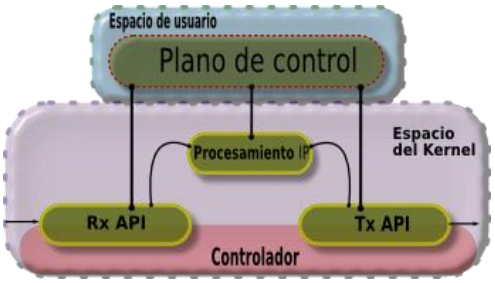


Figura 16. Separación del plano de envió y de control Fuente: Autor

CLICK es un software experimental que realiza operaciones típicas de un enrutador en el área de tratamiento de paquetes, trabajando a nivel del usuario y/o

1 XORP-ct. Internet: www.candelatech.com/XORP.ct/releases/1.8-CT/docs/
2 CLICK modular router. www.read.cs.ucla.edu/click/

a nivel del kernel. A CLICK se le puede cargar la tabla de enrutamiento que XORP ha creado, por medio de un *script*. CLICK fue escrito por Eddie Kohler quien describió detalladamente este proyecto en su tesis doctoral.

XORP tiene cuatro procesos principales que deben iniciar su ejecución, cuando se da inicio, al enrutador, estos son:

xorp_rtrmgr (Router Manager Process)
- xorp_fea (Forwarding Engine Abstraction Process)
- xorp_rib(Routing Information Base Process)
- xorp_finder

Cada uno de estos procesos está presente en el sistema Linux y cada uno es accesible al usuario a excepción del Finder. A cada proceso le corresponde un PID (Process Identification). En la figura 17 se observan los diferentes procesos que se ejecutan bajo XORP.

```
[root@localxorp rtrmgr]# ps -U root  |  grep xorp
 2820 pts/0    00:00:08 xorp_rtrmgr
 2821 pts/0    00:00:03 xorp_fea
 2822 pts/0    00:00:02 xorp_rib
 2823 pts/0    00:00:00 xorp_policy
 2824 pts/0    00:00:04 xorp_rip
 2825 pts/0    00:00:01 xorp_ospfv2
[root@localxorp rtrmgr]#
```

Figura 17. PID de XORP *Fuente: Autor*

XORP es desarrollado en un modelo de arquitectura modular asíncrono, donde cada Módulo es ejecutado independientemente pero con posibilidad de comunicación con otros Módulos. Un Módulo puede representar ya sea un protocolo de enrutamiento (*Unicast*, *multicast*), mecanismos de comunicación entre módulos (Finder), Módulo para habilitar la línea de comandos (xorpsh), Módulo para almacenar las rutas ganadoras (RIB), Módulo para comunicación de *socktes* I/O y manejo de las interfaces de red (FEA) y el Módulo que gerencia e inicializa a los procesos internos de cada Módulo, llamado *Router Manager* (rtrmgr) todos presentes en la figura 18.

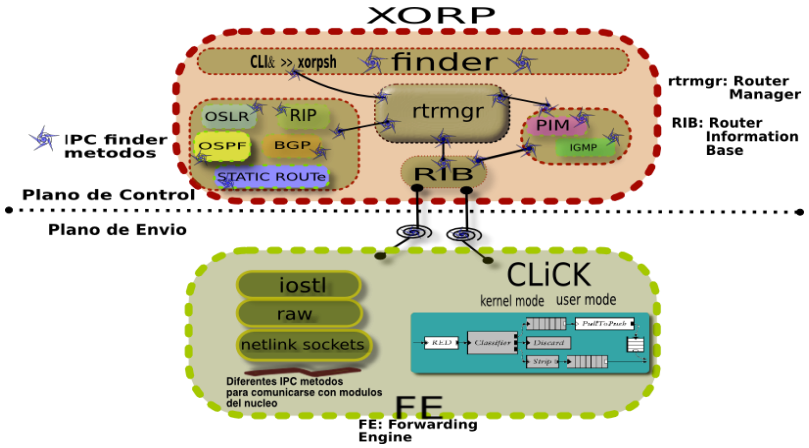


Figura 18 Arquitectura modular de XORP *fuente : Autor*

2.2 MÓDULO FINDER E IPC (INTER PROCESS COMUNNICATION) METODOS

El módulo FINDER ejecuta su tarea a nivel interno de XORP, el Finder se encarga de ser el facilitador de la comunicación entre los diferentes módulos, convirtiéndose en un enrutador virtual de las solicitudes de comunicación de los procesos. Para lograr esta meta, el finder, se soporta en la construcción de un método de direccionamiento, que debe ser común, sencillo, confiable y accesible a la comprensión humana, con el fin de proveer acceso a los métodos por parte de procesos ajenos, incluyendo también variables. De este concepto nace el XRL (Xorp Resource Locator). Un XRL es un mecanismo que habilita métodos de comunicación IPC (Inter-Process-Comunnication) utilizados por el finder y los módulos de xorp.

```
[root@localxorp rtrmgr]# netstat --inet -p -e --numeric-host
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State       User       Inode      PID/Program name
tcp        0      0 127.0.0.1:19999             127.0.0.1:60945             ESTABLISHED root       14388      2878/xorp_rtrmgr
tcp        0      0 127.0.0.1:52815             127.0.0.1:38360             ESTABLISHED root       14409      2879/xorp_fea
tcp        0      0 127.0.0.1:60945             127.0.0.1:19999             ESTABLISHED root       14384      2879/xorp_fea
tcp        0      0 127.0.0.1:60952             127.0.0.1:19999             ESTABLISHED root       14436      2880/xorp_rib
tcp        0      0 127.0.0.1:54364             127.0.0.1:40368             ESTABLISHED root       14411      2879/xorp_fea
tcp        0      0 127.0.0.1:40380             127.0.0.1:54364             ESTABLISHED root       14471      2882/xorp_static_ro
tcp        0      0 127.0.0.1:60960             127.0.0.1:19999             ESTABLISHED root       14467      2882/xorp_static_ro
tcp        0      0 127.0.0.1:54364             127.0.0.1:40371             ESTABLISHED root       14439      2879/xorp_fea
tcp        0      0 127.0.0.1:45568             127.0.0.1:35900             ESTABLISHED root       14441      2880/xorp_rib
```

Figura 19 Registro de la comunicación socket del finder fuente: Autor

En la figura 19 encontramos, un registro de la comunicación mediante *sockets*, de llamadas locales, realizadas entre los módulos xorp_fea, xorp_static_route, xorp_rib y xorp_rtrmgr. Este proceso de comunicación entre *sockets tcp*, ocurre un instante de tiempo después de haber dado inicio a XOR, producto de la necesidad de comunicación de los Módulos. En la figura se destaca el puerto 19999, el cual tiene como cliente y servidor al *finder* .

De otra parte, en la figura 20 se tienen las instrucciones de código de la librería FINDER_CONSTANTS_hh, utilizada para definir constantes y variables del módulo FINDER. En este código se destaca la función pública estática “uint16”, que tiene como retorno el valor 19999, valor encargado de establecer el puerto de recepción y envió para la comunicación con el FINDER.

```
“#ifndef __LIBXIPC_FINDER_CONSTANTS_HH__
 00025 #define __LIBXIPC_FINDER_CONSTANTS_HH__
 00026  class FinderConstants {
 00027public: static uint16_t FINDER_DEFAULT_PORT()  {return (19999);}
 00028  static const IPv4 FINDER_DEFAULT_HOST() { return
 00029     IPv4::LOOPBACK(); }
 00030   #endif // __LIBXIPC_FINDER_CONSTANTS_HH__”
```

Figura 20 Librería FINDER_CONSTANTS fuente: Autor

Un módulo de XORP debe tener una Interface[1] XRL, la cual permite al Módulo

1.XORP, Inc.XRL Interfaces: Specifications and Tools.2009.1p.Internet:www.candelatech.com /XORP.ct/Sreleases/1.8-CT/docs/libxipc/xrl_interfaces.pdf

tener un enlace sobre las operaciones y métodos (de programación) para poder interactuar con este. Para lograr el direccionamiento de alguna Interface en particular, los desarrolladores de XORP crearon lo que se llama una XRL (Xorp Resource Locator) que es una dirección que indica la ubicación de los métodos de un módulo en particular. Su sintaxis es similar a la de una dirección *url* de internet, pero con la diferencia de que el direccionamiento lo hace directamente dentro de un archivo (.xif se pronuncia punto-ziff) ubicado en el computador, en el cual se han definido los métodos públicos accesibles para el usuario y a otros módulos.

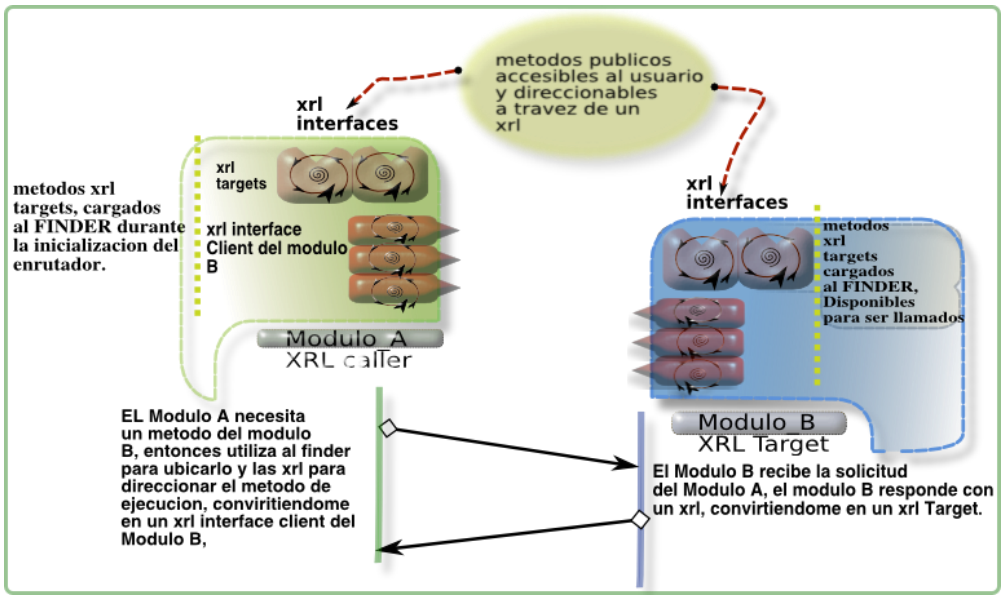


Figura 21 Conceptos de comunicación entre Módulos fuente: Autor

La comunicación entre procesos utilizando la interfaz XRL, sin tener en cuenta la intervención del finder, se observa en la figura 34 y se describe a continuación:

El Módulo que inicia la solicitud de un método hacia otro Módulo se llama *XRL caller* (Módulo_A), a diferencia del proceso que genera la respuesta a la solicitud se llama *XRL target* (Módulo_B).

En la figura 21 se presenta un pequeño ejemplo de como un Módulo *router manager* (rtrmgr) utiliza al finder para ubicar a otro Módulo (static_route) para activar una variable.

El *xrl Target* es el Módulo static_route, el cual es llamado por el Módulo *router manager* (rtrmgr). Todo el proceso ubicado en la figura 35 tiene como fin activar una variable *enable_estatic_route* (que indica que se activa el Módulo). En este instante de tiempo, inicia internamente en el *static_route* una serie de pequeños eventos programados en el Módulo.

Primeramente hay que tener en cuenta que el Módulo *router manager* (rtrmgr)

tiene que comunicarse con cada uno de los módulos (Protocolos, interfaces, firewall) que han sido declarados en el archivo de ejecución (.boot) para inicializarlos, para ello utiliza el finder y los métodos de direccionamiento (XRLs). Se debe considerar también que cada Módulo durante el proceso de inicialización

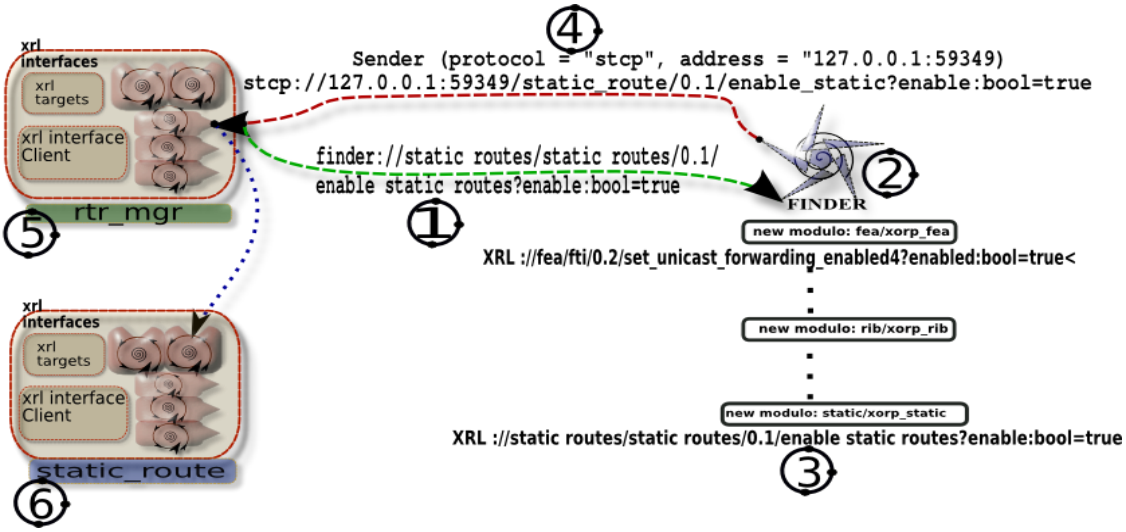


Figura: 22 Descripción detallada en la comunicación de Módulos donde interviene el finder fuente: Autor

del enrutador, registra sus *xrl interfaces* al finder para sean direccionales.

A continuación se hará una explicación más detallada de la secuencia de comunicación entre módulos, específicamente el Módulo rtrmgr y la inicialización del Módulo static_route, descrita en la figura 22.

1. El Módulo rtrmgr necesita saber la ubicación del Módulo static_route, ya que en el archivo tipo .boot donde se configuran los procesos de enrutamiento esta declarado el Módulo static_route. Para lograrlo utiliza un método que describe el procedimiento de direccionamiento para que el *finder* lo resuelva.

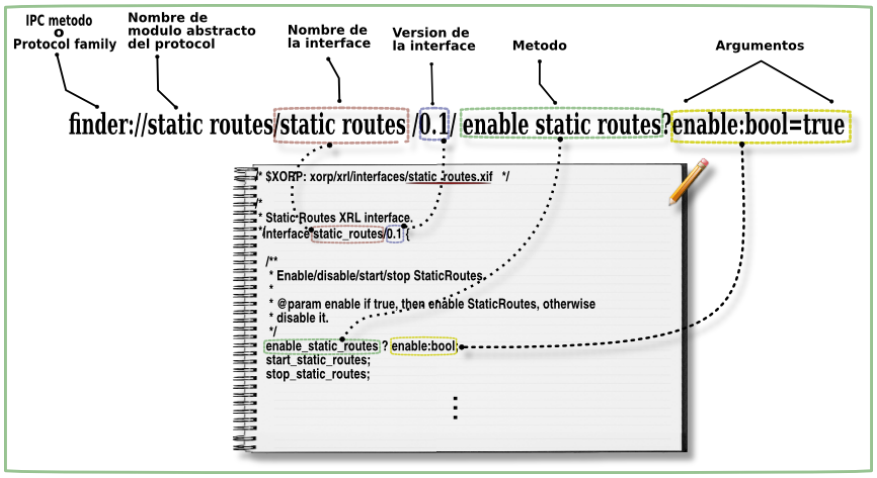


Figura 23 Descripción de los segmentos que hacen parte de un xrl genérico fuente: Autor

En la figura 23, se encuentran los diferentes segmentos que van indicando progresivamente como ubicar al método (enable_static_route), dentro del archivo tipo .xif, formando un xrl genérico. En la documentación de XORP-ct, este tipo de xrl[1] es definido por los desarrolladores de XORP como la versión no resuelta de una llamada, caracterizado por que el método de comunicación es el finder. La descripción de cada segmento es la siguiente:

Familia de Protocolos o método IPC: Es el mecanismo por el cual los XRL son transportados de un componente a otro en el presente sistema, definidos por el equipo de XORP[2] se encuentran tres como las principales familias de protocolos de comunicación: TCP, UDP y también se encuentra la familia del protocolo finder que permite al finder ser direccionable a través de un xrl. El mas utilizado es TCP. En la figura 33 se puede observar el protocolo tcp de comunicación que utilizan los diferentes módulos al ser ya direccionales gracias al finder

Nombre del Módulo: Este nombre es declarado en una variable (static_route.targetname) de un archivo “plantilla”, por ahora lo que se debe saber, es que este nombre debe ser conocido y declarado para que el finder lo valide como un XRL target y pueda ubicarlo.

Nombre de la Interface: En este punto del segmento se entra a la identificación de un archivo de extensión .xif, ubicado en el computador donde tiene como

---

1 Eddie Kohler.XORP: An Open Platform for Network Research.ICSI Center for Internet Research,Berkeley, California.2003.Internet:research.microsoft.com/ apps/pubs/default.aspx?id=73275.p-56.

2 Pavlin Radoslavov, Designing Extensible IP Router Software, University College, London:UCLA, International Computer Science Institute, 2005..Internet:www.candelatech.com/XORP.ct /papers/XORP-nsdi.pdf. p-197

cabecera la cadena “interface static_routes/0.1 {...”, en este archivo se declaran todos los
Métodos que puede habilitar el Módulo, siguiendo ciertas reglas en la declaración.

Versión de la Interface: Este número es establecido por el desarrollador, registra los cambios hechos a código de la interface o del Módulo.

Método: Es el método, que ejecutara el xrl Target, llamado por el *XRL caller.* Para el ejemplo, el método es habilitar el inicio de static_route.

Argumentos: Es el tipo de parámetros, que fueron expuestos en el archivo principal de configuración de XORP de extensión .boot. Datos necesarios para que el método ejecute la tarea a la que fue programado.

2. El FINDER definido como la entidad que resuelve la ubicación de los módulos con un procedimiento de direccionamiento llamado xrl.

3. Cada Módulo previamente configurado, carga sus respectivos xrls al finder (Metodos xrl Target según definición realizada en el archivo de extensión .tgt). Luego al iniciar el softroute este hará una validación y retornara la información requerida. Este proceso se detallara más adelante con la descripción de las funcionalidades del rtrmgr (router manager).

4. El finder presenta la versión resuelta de un xrl, para el caso del ejemplo, es enviada al Módulo rtrmgr, con la información de ubicación del Módulo static_route. La dirección IP (127.0.0.1), el puerto (59349) y el protocolo de comunicación (stcp). La librería encargada de darle resolución al XRL es llamada Library *XRLIPC* y esta ubicada en la carpeta xorp/libipc del código

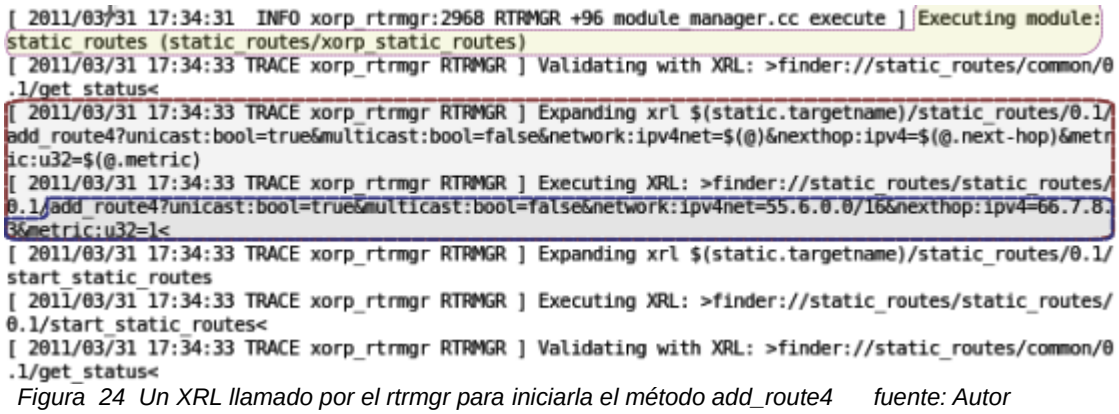


Figura 24 Un XRL llamado por el rtrmgr para iniciarla el método add_route4 fuente: Autor

fuente.

En la figura 24 se muestra la ejecución del Módulo static_route en consola, donde es posible rastrear al xrl que se ha disparado, allí en la salida de consola se puede observar el xrl con el protocolo de comunicación “finder”, para ejecutar el método “add_route” (en azul), para crear una ruta estática.

5. El Módulo *router manager* (xorp_rtrmgr), encargado de inicializar a los demás procesos, es identificado como un *xrl Interface Client* que apunta a la interface del Módulo static_route/0.1/{...}.

6. El Módulo static_route, encargado de crear rutas estáticas, identificado como el xrlTarget para el ejemplo.

La tarea de ubicación de los módulos se realiza una vez, ya que la ubicación es almacenada como cache para futuras consultas, descrito por uno de los desarrolladores de XORP[1]
Si la comunicación sufre algún error, la cache es limpiada, la aplicación es notificada, y el proceso responde con un reporte programado, descrito en la librería de control de errores de XORP.

2.3 ROUTER MANAGER PROCESS (rtrmgr)

El *router manager* es el primer proceso que se ejecuta en el *softroute* e inicializa a los demás procesos configurados ofreciendo la lógica requerida para su inicio. Para cumplir la tarea para la cual fue programado, toma la configuración descrita en el archivo tipo .boot y la contrasta con la información declarada en un archivo llamado "template" o plantilla, el cual sirve de enlace entre el archivo descriptor de los métodos XRLs (el .xif) y el archivo de configuración del softroute (.boot).

1 Pavlin Radoslavov, Designing Extensible IP Router Software, University College, London:UCLA, International Computer Science Institute, 2005,197-p. Internet:www.candelatech.com/XORP.ct /papers /XORP-nsdi.pdf

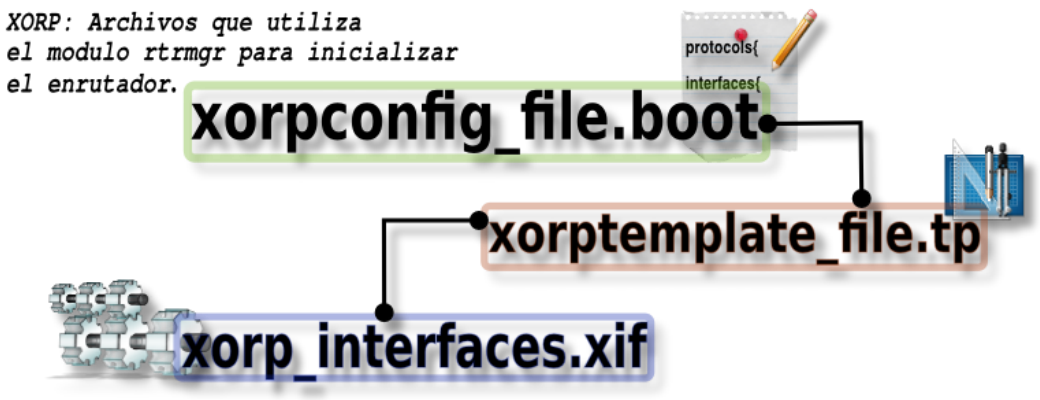


Figura 25 Archivos involucrados en la definición de un proceso fuente: Autor

Esta comparación es realizada por que en la construcción del archivo “Template” o plantilla se detalla cómo debe ser configurado un Módulo en particular. Por cada proceso existente en XORP hay un archivo *template y* un *.xif.* En la figura 25 se muestran los archivos que utiliza el *router manager* para inicializar al enrutador estos son los archivos Plantilla y el .xif (pronunciado punto-ziff).

Cuando el *rtrmgr* contrasta la información declarada en el archivo de extensión .boot con el archivo *template*, este hace una rigurosa auditoria en la forma como se ha declarado un proceso en particular, el rtrmgr compara el tipo de datos de los parámetros necesarios para la ejecución, la fiabilidad del xrl, el orden especifico de configuración de los nodos con sus dependencias correctamente declaradas.

El archivo plantilla es creado en cualquier procesador de texto y durante la construcción de este archivo se pueden distinguir dos partes, la primer parte muestra el orden jerárquico de los nodos y el tipo de datos que debe contener un nodo en un proceso en particular, mientras que la segunda parte describe los comandos o acciones que van a tomar los diferentes xrls, dispuestos en cada nodo, para ser disparados por el Módulo xorp_rtrmgr, ver figura 26. Este archivo está ubicado en */XORP/etc./templetes* para la versión 1.6 y anteriores.

Para la figura 26, representa un fragmento del archivo plantilla *static_route.tp,* en el cuadro verde indica los tipos de datos de los parámetros que se necesitan dar a unas dependencias en particular al Módulo para que sea exitoso la ejecución. Al mismo tiempo lograr que la configuración en su diseño tome una forma jerárquica, sobre niveles de definición de lo más general a lo más específico.

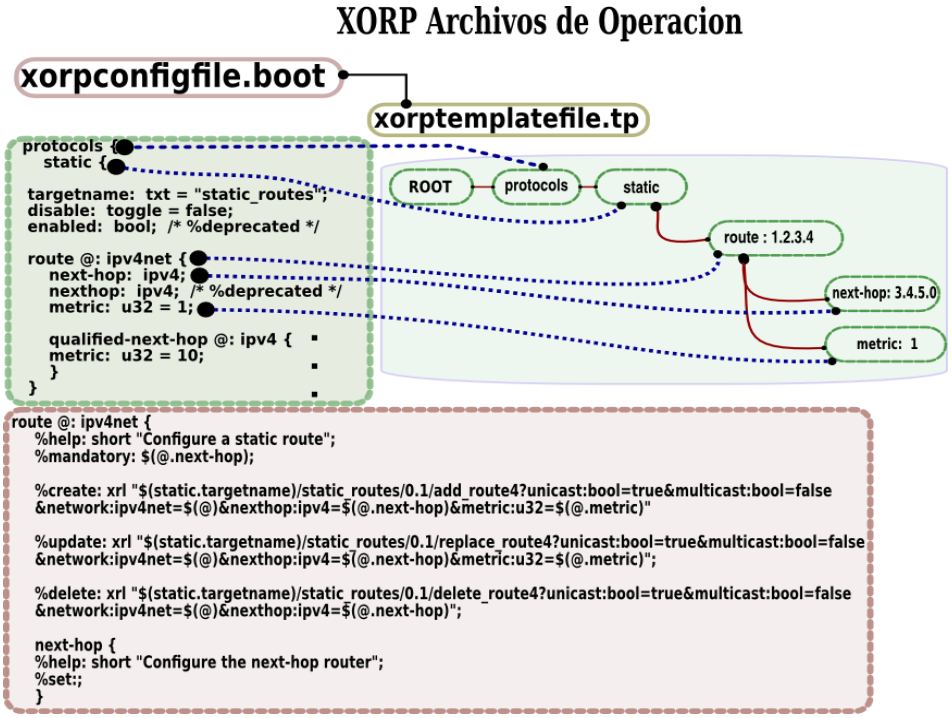


Figura 26 Fragmento de un archivo plantilla y estructura de nodos fuente: Autor

Al finalizar la creación de un archivo plantilla, se logra visualizar un esquema de tipo nodo. El tipo de los valores permitidos para ser declarados son descritos por los desarrolladores de XORP con el nombre de *el tipo de valores de los nodos en el árbol plantilla,*[1] los valores que pudiera tomar un nodo serían: sin signo entero de 32 bits(u32), rango de valores sin signo de 32 bits (u32range), dirección ipv4 en formato decimal etc. Para la figura 38 el nodo “protocols statu route net-hop” tiene el tipo “ipv4”.

Siguiendo con las observaciones de la figura 26 se tiene, un cuadro de color rojo se destaca la implementación de los comandos por nodo para implementar en el código de c++; Identificados con el carácter ”%” induciendo a una forma temprana de un xrl. Cada comando tiene una función muy particular afectando al xrl y al método. En el capitulo que tratara la creación de un proceso se detallara la creación de los archivos descriptores (.xif y .tgt).

1 XORP Router Manager Process (rtrmgr).XORP.inc.de Internet:http://www.candelatech.com/ XORP.ct/ releases/1.8-CT/docs/rtrmgr/rtrmgr.pdf. P-7

El *router manager* debe realizar una serie de pasos cuando inicializa el *softroute*, en ese lapso de tiempo el rtrmgr hace un uso extensivo de los archivos de configuración (.boot), de los *template* y los .xif. A continuación estos son los pasos descritos por el equipo de desarrolladores de XORP:[2]

1. El rtrmgr lee todos los archivos “plantilla”. El archivo plantilla describe, como se debe declarar un proceso en el softroute, sirviendo de pegamento entre la parte funcional (.xif) y declaratoria (.boot) de un proceso. Después de leer los archivos plantilla, el softroute dispone de la información actualizada de que procesos puede soportar, almacenando esa información en un árbol plantilla. Con la información leída, actualizada y almacenada en un árbol plantilla, es de nuevo leída directamente en el árbol plantilla, en busca de errores. El proceso rtrmgr terminaría si se encontrara algún error.

En la figura 27, se tiene una representación grafica de lo que sería un árbol plantilla, donde se manejan el tipo de variables (pinte, u32, ...) y comandos disponibles para cada nodo. Algunos comandos del árbol plantilla son: %set, %delete, %créate.

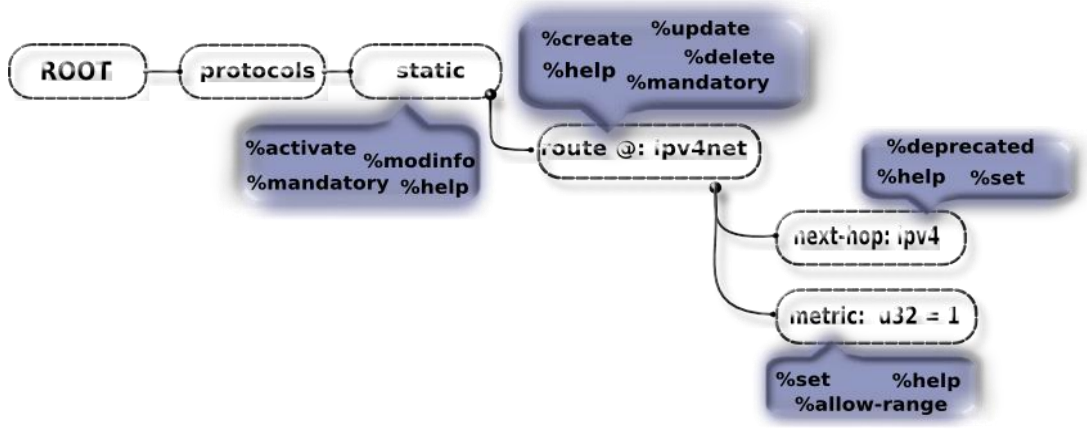


Figura 27. Representación grafica de un Árbol plantilla con el tipo valores de los nodos y comandos fuente: Autor

2. El rtrmgr lee el contenido de todos los XRLs en el directorio que esta ubicado en *XORP/xrl/,* para descubrir que procesos de comunicación están validados para cada Módulo. Exponiendo en cada archivo de tipo .xif, los métodos que soporta un proceso configurado. Estos XRL son contrastados con la versión del xrl árbol plantilla.

3. El rtrmgr lee el archivo de configuración .boot. Todas las opciones de

2 XORP Router Manager Process (rtrmgr).XORP.inc.de Internet:http://www.candelatech.com/ XORP.ct/ releases/1.8-CT/docs/rtrmgr/rtrmgr.pdf. P-2

configuración expuestas en el archivo .boot deben coincidir con las descritas en el archivo *template*. Ya leído el archivo de configuración este es almacenado en un archivo conocido con el nombre de árbol de configuración. Este árbol es muy parecido *árbol template* sólo que este tiene los valores ya dichos de las variables declaradas en el archivo .boot.

En la figura 28 se observa una representación grafica de lo que sería un árbol de configuración.[1]

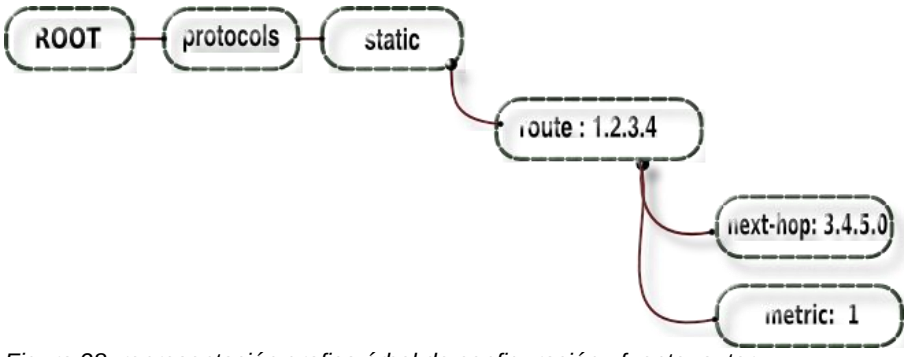


Figura 28. representación grafica árbol de configuración fuente: autor

4. El rtrtmgr lee el archivo del árbol de configuración, para descubrir la lista de procesos que necesitan ser inicializados y proveer las funcionalidades requeridas para su inicio.

5. El rtrmgr atraviesa de nuevo el árbol plantilla para descubrir el orden de inicio de los comandos requeridos para tener la funcionalidad del proceso configurado.

6. El rtrmgr empieza el primer proceso de la lista para ser inicializada.

7. Si no ocurre ningún error, el rtrmgr atraviesa el árbol de configuración, para construir la lista de comandos, que se necesitan para ejecutar la configuración de los procesos descritos en el archivo .boot. Estos comandos son llamados uno tras otro, y con el desarrollo exitoso de cada comando va pasando el siguiente hasta lograr el proceso completo.

8. Si no hay ningún error en la configuración, el siguiente proceso es iniciado y configurado, hasta que todos los procesos descritos en el archivo tipo .boot se hayan completado.

---

1 XORP Router Manager Process (rtrmgr).XORP.inc..de Internet:http://www.candelatech.com/ XORP.ct/ releases/1.8-CT/docs/rtrmgr/rtrmgr.pdf. P-21

9. En este punto el enrutador está listo y trabajando. Seguidamente el rtrmgr esta disponible para recibir conexión con el *xorpshell*, para operaciones interactivas con el usuario por medio de la línea de comandos con el terminal de Linux.

Como se pudo constatar en las paginas anteriores la mayor parte del software esta relacionado al plano de control del *softroute*, donde el gran director de esta filarmónica de software es el Módulo rtrmgr, incluyendo como la varita del director al finder y los métodos xrls. XORP automatiza a nombre del usuario-xorp, la tabla de enrutamiento que es obteniendo por algoritmos estándares, dentro del sistema operativo Linux por medio de *routing daemons,* para luego aplicar esta tabla con

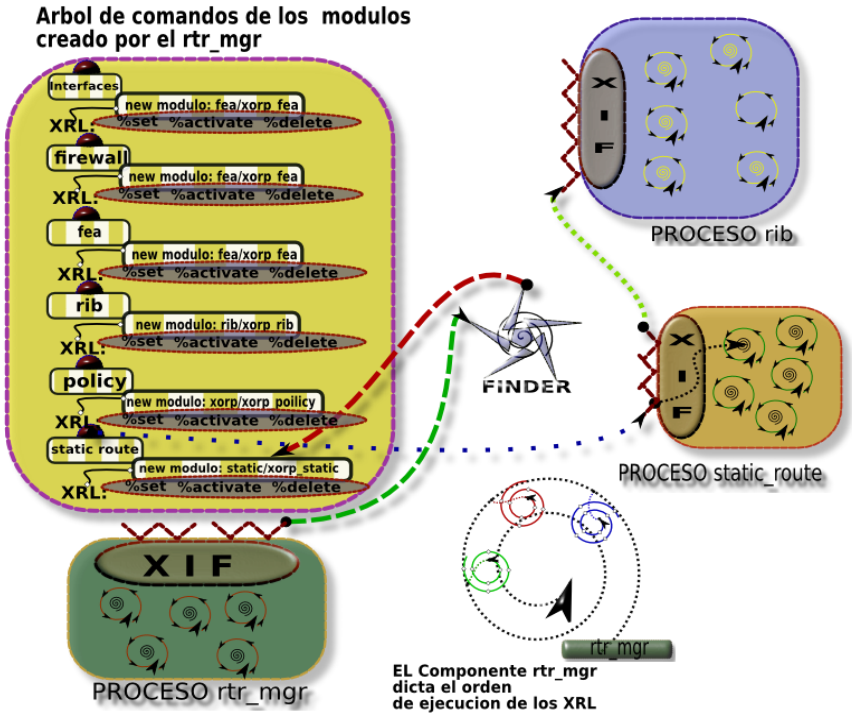


Figura 29. Pre inicialización de XORP por medio del Módulo rtrmgr fuente: Autor

llamadas internas propias del sistema huésped.
A continuación detallara el proceso del *rtrmgr,* desde la creación del árbol plantilla hasta el envió y recepción del xrl para el proceso de llamar al Módulo *static_route*
En la figura 29, se muestra, el árbol de comandos (el cuadro amarillo) ya inicializado y ordenado por el Módulo rtrmgr, haciendo un llamando al finder (la estrella azul), para resolver la solicitud (flecha verde) en este caso para la grafica, la solicitud de direccionamiento al Módulo *static_route*, una vez resuelta (flecha roja) pasa a que se ejecute el comando que le corresponde al xrl, (flecha punteada azul) activado por el mecanismo de xrltarget(static_route), generando un evento

dentro del Módulo receptor, para luego llamar a una función programada dentro del Módulo, en este caso enviar un dato al Módulo RIB (Router Information Base), que a su vez tendrá habilitados métodos xrl relacionado con el Módulo *static_route*, repitiendo así el ciclo de llamadas, a cada uno de los módulos y ejecutando en orden los comandos habilitados en el archivo plantilla de un Módulo en particular.

2.4 EL MÓDULO XORPSH

La configuración del enrutador, por vía consola CLI (*Comand Line Interface*), hace suponer, la relación entre el terminal y aquel proceso que tiene como domino la gerencia e inicialización de los módulos configurados, ya que el Módulo rtrmgrt es de vital importancia, los desarrolladores concluyeron en limitar el acceso a este Módulo por medio de llamadas xrl pre establecidas. Por ello la creación del Módulo xorpsh que informa sobre parámetros ya configurados, comandos de información de la tabla de enrutamiento e información concerniente de los protocolos de enrutamiento configurados, además el acceso a la posible configuración del archivo tipo .boot para modificación desde consola queda restringido a la autentificación del usuario perteneciente al grupo XORP. Ver figura 30.[1]

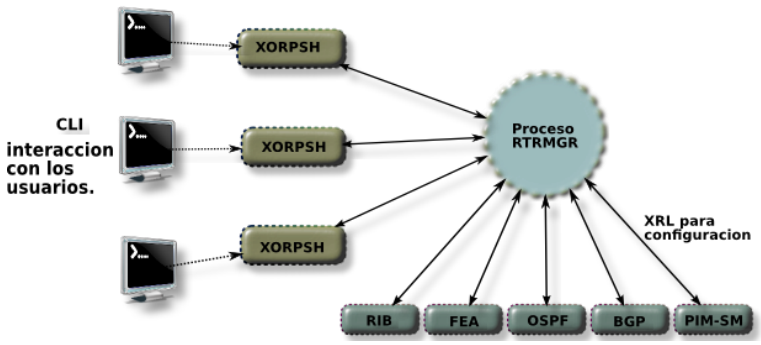


Figura 30 Mediación del Módulo xorpsh entre rtrmgr y la CLI fuente: Autor

2.5 FEA (XORP Forwarding Engine Abstraction)

Una de las tareas que tiene un enrutador es proveer al paquete que ya tiene una ruta establecida la forma de poder llegar a su destino. Esta logística recae sobre el plano de envió y mas exactamente en XORP en el Módulo de la FEA de XORP.

La FEA se define como un Módulo de XORP, en una plataforma independiente con características a un *software* de control para la parte del envió de paquetes. Además provee una separación funcional de la etapa de control en la ejecución de los procesos de enrutamiento a la etapa del envió de paquetes. En caso que la FE

1 XORP Router Manager Process (rtrmgr).XORP.inc.de Internet:http://www.candelatech.com/ XORP.ct/ releases/1.8-CT/docs/rtrmgr/rtrmgr.pdf.P-21.

(*Forwarding Engine)* no se encuentre en la misma maquina, es decir la parte física que hace posible el envió de los paquetes, sería la FEA la encargada de conducir la comunicación entre la parte de control de procesos de enrutamiento (que provee la ruta) y el plano físico de envió (que provee la recepción y envió de paquetes).
La FEA provee cuatro herramientas en *software* que son fundamentales para que la FE ejecute su tarea, estas herramientas están presentes y accesibles por medio de una API, que es ampliamente descrita por el equipo XORP en la documentación oficial:[2] gestión de las interfaces, gestión de la tabla de envió, *raw packet* I/O, *TCP/UDPsocket I/O.*
El Módulo FEA es sólo utilizado plenamente con algoritmos de enrutamiento *unicast*, para el caso de los protocolos de multidifusión los desarrolladores plantearon la creación de un Módulo MFEA (Multicast Forwarding Engine Abstraction), el cual es más complejo y trabaja en colaboración con la FEA.
A continuación definición de las cuatro herramientas que provee la FEA .

2.5.1 Gestión de las Interfaces de Red.

El *router manager* es la principal fuente de pedidos para la configuración de las interfaces por medio de la FEA, como se puede observar en figura xx la comunicación de *sockets tcp* entre los módulos xorp_rtrmgr, xorp_fea y el FINDER (escuchando en el puerto 19999).
El *route manager* dispone del archivo de configuración .boot para extraer la información concerniente a los parámetros de las interfaces. Después, le solicita a la FEA que habilite las interfaces utilizando los valores que ya fueron requeridos por el rtmrgr. Para iniciar el proceso de la configuración de las interfaces, no es distinto a los documentados anteriormente con la explicación de la comunicación entre los módulos finder, rtrmgr y static_route.

2 XORP Forwarding Engine Abstraction.XORP. Inc.Internet:http://www.candelatech.com/XORP.ct/releases/

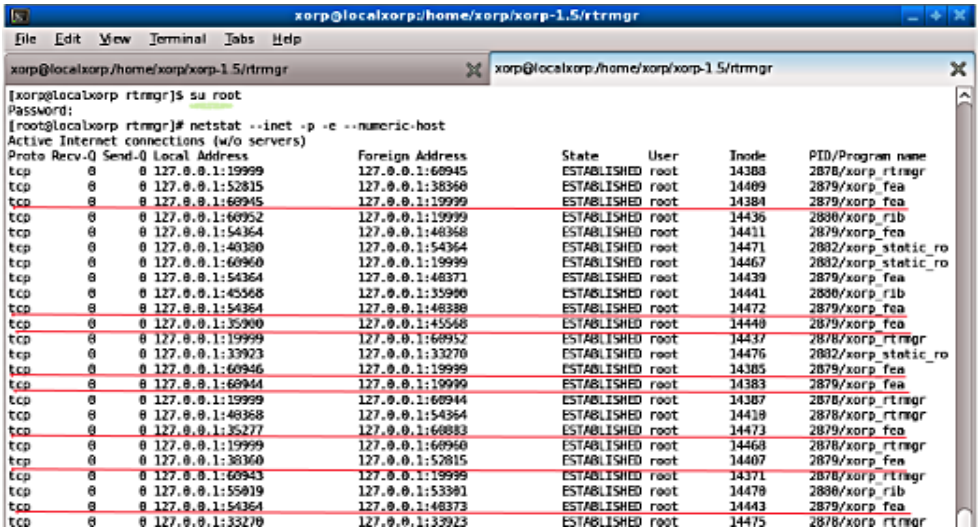


Figura 31. Respuesta y pedidos de la FEA a nivel de sockets fuente: Autor

Los desarrolladores de XORP definieron la forma de enviar notificaciones tipo *UPDATE*[1] a los procesos de enrutamiento, en caso de que si alguna interface de red ha sufrido un cambio, el protocolo de enrutamiento tomara acciones al respecto.

2.5.2 Manejo de la tabla de envío

La FEA primeramente recibe de la RIB (Routing Information Base), la tabla de enrutamiento, aquellos enrutadores ganadores especificados por los algoritmos de enrutamiento, para luego hacer los cambios necesarios como remover e insertar entradas de enrutamiento dentro al plano de envió. Para tener acceso a la tabla de envió se necesita utilizar *plug-ings*, que son específicos para el tipo de maquina que hospeda a XORP. Para el caso de Linux se debe utilizar NETLINK.

2.5.3 *Raw packet I/O*

Protocolos de Enrutamiento, como OSPF, necesitan poder recibir y enviar paquetes a interfaces de red especificas en el plano de envió, ya que OSPF utiliza su propio mecanismo de retransmisión y *acknowledgment*[1], *RAW sockets* permite adaptar la comunicación de tipo *sockets*, a las necesidades de la maquina huésped del plano de envió, para lograr cumplir con el intercambio de información de enrutamiento, estas operaciones I/O recaen sobre la FEA.

---

1 XORP Forwarding Engine Abstraction. XORP.Inc. Internet:http:// www.candelatech.com /XORP.ct/releases/1.8-CT/docs / fea/fea.pdf.P-1

1 Network Routing Algorithms, Protocols, and Architectures, 2007. autor: Deepankar Medhi, Karthikeyan Ramasamy.Pag 18.

2.5.4 TCP/UDP *sockets* I/O

Protocolos de enrutamiento como BGP o RIP necesitan enviar y recibir paquetes en direcciones IP especificas para establecer una conexión e intercambiar información de enrutamiento atreves del protocolos de transporte como TCP/UDP. Para el manejo desde XORP de este tipo de sockets, la API disponible de XORP permite extender las capacidades del sistema anfitrión de XORP UNIX TCP/UDP, logrando que las operaciones I/O se deleguen en la fabricación de mensajes xrls, en el dominio de la FEA, el mismo caso ocurre para *sockets* crudos.

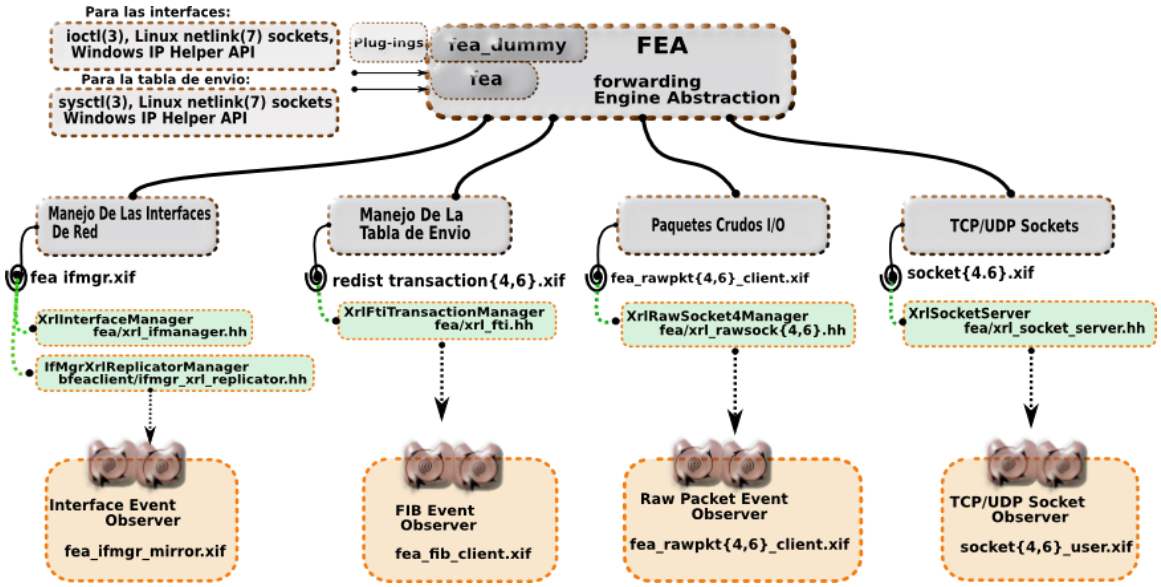


Figura 32. Organigrama de las librerías habilitadas por la FEA fuente : Autor

En la figura 32 se puede observar que para cada herramienta que soporta la FEA hay una serie de definiciones que involucran a archivos del tipo .xif, que detallan las funciones pertinentes a cada herramienta, además acompañados con las librerías habilitadas por los desarrolladores (en verde), que logran las operaciones de sockets para recibir y enviar paquetes, inserción y borrado de tablas de enrutamiento y el manejo de las interfaces de red, también se puede detallar el tipo de eventos que se activan para las operaciones I/O y de cambio de interfaces de red.(en naranja). En la parte superior de la figura 32 se definen los *plug-ings* perteneciente a cada SO, que permiten acceder a informacion y configuración de las interfaces de red y a la tabla de envio.

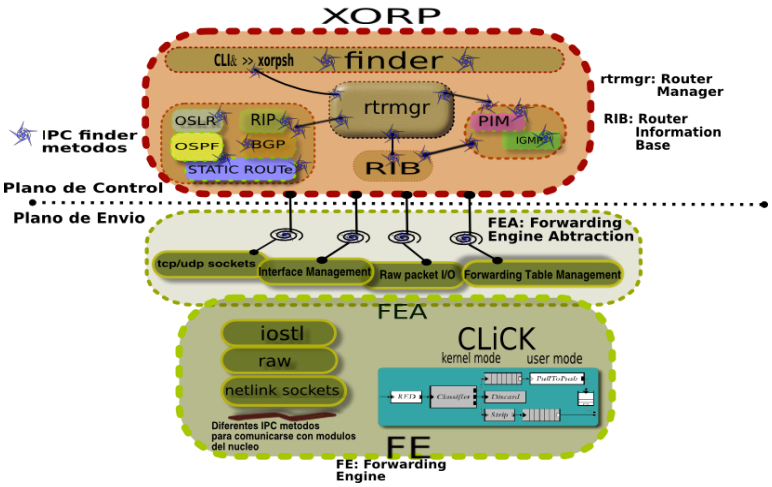


Figura 33. La arquitectura modular de XORP fuente: Autor

En la figura 33 se muestra la arquitectura modular de XORP, donde se aprecia al *router manager* como el integrador y gerenciador de los distintos procesos ya sean para protocolos de enrutamiento *unicast* como OSPF o para protocolos de multidifusión, diferenciando claramente dos funcionalidades programadas en XORP, que son el plano de control y el plano de envió. Para el plano de envió en XORP utiliza las facultades de red del sistema que alberga XORP, para lograr esto XORP utiliza una serie de mensajes propios de sistemas Linux soportados por los xrls y librerías propias de XORP (FEA) que involucran librerías llamadas *netlink* para comunicarse con el núcleo.

2.6 PREGUNTAS FRECUENTES SOBRE XORP

1.) Que es exactamente XORP ?
XORP actualmente implementa un conjunto de protocolos de enrutamiento, con una API de programacion para extender XORP y herramientas de configuración. Soporta protocolos BGP, RIP, OSPF, PIM-SM, IGMP/MLD y VRRP(Virtual Router Redundancy Protocol). IPV4 e IPV6 son ambos soportados.
Hasta la presentacion de este trabajo XORP no implementa su propio sistema de envio de paquetes. Este toma prestado el motor de envio del sistema operativo que hospeda a XORP.

2.) Cuales son los pros y cons de XORP ?
Pros:
● El codigo es libre bajo licencia gpl (GNU Lesser General Public License)
● El codigo es ejecutado facilmente en hardware de PC y en sistemas operativos Linux, unix BSD y parcialmente en Windows Server.

● El diseño Modular permite una separacion funcional de las entidades de un proceso, significando que la falla de un componente no compromete la totalidad de XORP.
● El diseño Modular permite habilitar la integracion el hardware para el plano de envio.
● Cada componente de XORP utiliza metodos IPC que son flexibles con la posibilidad de crear una arquitectura distribuida.
● Si requiere algun desarrollo especifico en algun area de XORP, el desarrolldor y mantenedor del codigo ofrece sus servicios por un costo.
● Se puede instalar XORP como enrutador para servicios en la nube.

Cons:
● El software del plano de envio limita la tasa de envio de paquetes.
● No hay un soporte explicito para QoS, pero puede ser desarrollado.
● La utilizacion de conceptos como Multiplexacion de Interfaces de C++ aplicados a multiple herencia, no es para el perfil del desarrollador promedio.

3.) XORP como se compara con otros software que proveen enrutamiento avanzado como Zebra y BIRD ?
XORP y Zebra/BIRD compiten por el mismo espacio, ya que proveen una solucion de software libre que implementa los mas conocidos protocolos de enrutamiento avanzado. Sin embargo se pueden diferenciar de varias formas. XORP hace enfasis en la capacidad de extencibilidad y esto es reflejado en la arquitectura modular.
Tambien sobresale en que el operador de XORP no tiene por que saber como esta implementado XORP internamente.
Desde el punto del desarrollador Zebra/BIRD es desarrollado en c , mientras que XORP es implementado en c++.
Desde el punto de vista del investigador de redes, XORP provee una gran lista de librerias en c++, que deberian simplificar el desarrollo en la investigacion. De hecho el principal objetivo de desarrollo de XORP es el campo de la investigacion.

4.) Hay posibilidad de llevar a XORP hacia un ambiente de producción ?
Si, es posible, de hecho la ultima version 1.8.3-XORP.ct permite adecuarlo a una medio embebido. Inclusive el desarrollador y mantenedor del codigo XORP.ct Been Greear ha estado trabajando activamente en la estabilidad de XORP; con la resolucion de cerca de 140 inconvenientes relacionados con la parte operativa de XORP.

3. PRUEBAS DE LOS PROTOCOLOS DE ENRUTAMIENTO DE XORP CON ORACLE VIRTUABOX E INTERFACES VIRTUALES

Al estudiar en el capitulo anterior como XORP opera a un bajo nivel, se pasara a ejecutar las operaciones de enrutamiento *unicast* que XORP es capaz de realizar, permitiendo describir a XORP desde un punto operacional de alto nivel.
Logrando incluir durante cada éxito de ejecución de XORP una explicación operacional.

La utilización de herramientas de Virtualización como Oracle VirtualBox para emular una mini red virtual de dos enrutadores y un cliente por enrutador, tiene como fin probar los distintos protocolos de enrutamiento y características de configuración que posee XORP, como la utilización del Firewall, línea de comandos CLI, CLICK, VLAN y *policy statement.*
En las pruebas se emplearon herramientas de virtualización de tarjetas de red, *interfaces* TUN/TAP, las cuales actúan como tarjetas de red en la capa de red 3. VirtualBox permite utilizar estas tarjetas de red en el sistema operativo invitado.

Para la implementación de la mini red con cuatro maquinas virtuales se describirán los pasos, que servirán de base para la ejecución de los protocolos de enrutamiento *unicast*. En cada ejecución se capturaran los mensajes de comunicación entre los dos enrutadores, por medio del *software wireshark.*

Las maquinas virtuales residen en un INSPIRON 640m con procesador Intel core 2 duo T7200, con una RAM de 1.5GB, en un Fedora 12 i386.

3.1 CONFIGURACIÓN DE LA MINI RED PARA LAS PRUEBAS DE XORP

El objetivo principal es observar los mensajes que generan los protocolos de enrutamiento, aplicando la configuración básica y necesaria en XORP para que puedan ejecutarse los protocolos de enrutamiento *unicast* y así describir el proceso que un operador de XORP debe tener en cuenta al ejecutar este *software* de enrutamiento.
Pero antes, se debe describir el procedimiento de configuración para establecer los enlaces entre las interfaces de red, dispuestas en la figura 34. Una vez probados los enlaces con algunos comandos de auditoria de red, el siguiente paso es ejecutar XORP y ejerza las operaciones de enrutamiento.

El proceso de creación de maquinas virtuales es bastante intuitivo, además de una muy buena documentación de parte de Oracle VirtualBox, en su manual oficial que describe las funcionalidades de este software, no hay que olvidar a los usuarios de VirtualBox que documentan por medio de *blogs* procedimientos de instalación.

Para la figura 34 se presentan cuatro maquinas Virtuales, dos con sistema operativo fedora 10 cada una tiene instalado 1.5-XORP.inc, además de dos

interfaces de red creadas por VirtualBox. Como cliente de cada enrutador se tiene a la distribución de Linux Peppermint con una interface de red también creada desde VirtualBox.

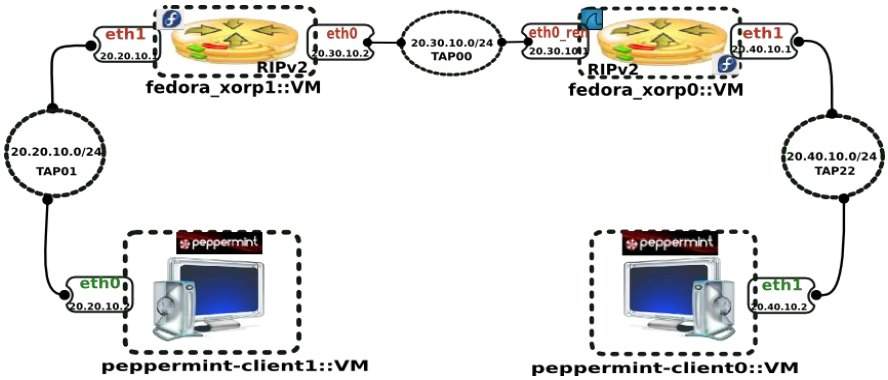


Figura 34. Estructura de la mini red a configurar Fuente: Autor

Pasos a desarrollar con el fin de obtener la configuración dispuesta en la figura 34, para obtener funcionalidades de enrutamiento con el protocolo RIP (Routing Information Protocol) :

1. Ejecutar el script *./config-taps.sh* como root, para crear, configurar y activar las interfaces de red TAP. Disponible en el anexo D.
Las direcciones IP establecidas son para configurar el protocolo de enrutamiento RIP (Router Information Protocol), basado en el esquema de la figura 34, donde se definen el espacio de las sudnets 20.xx.10.0, 20.xx.10.0, 20.xx.10.0.

```
tap00     Link encap:Ethernet  HWaddr F2:4D:A2:BA:9A:31
          inet addr:20.30.10.0  Bcast:20.30.10.255  Mask:255.255.255.0
          inet6 addr: fe80::f04d:a2ff:feba:9a31/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:2 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

tap01     Link encap:Ethernet  HWaddr 5E:5C:E3:D0:61:54
          inet addr:20.20.10.0  Bcast:20.20.10.255  Mask:255.255.255.0
          inet6 addr: fe80::5c5c:e3ff:fed0:6154/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:3 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

tap22     Link encap:Ethernet  HWaddr BA:AD:0B:63:F8:D4
          inet addr:20.40.10.0  Bcast:20.40.10.255  Mask:255.255.255.0
          inet6 addr: fe80::b8ad:bff:fe63:f8d4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:2 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Figura 35. Resultado del script para las interfaces taps
fuente: Autor

También se observo como la maquina huésped genera una tabla de enrutamiento. Figura 36.

```
[root@localy Diegu]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
20.30.10.0      0.0.0.0         255.255.255.0   U     0      0        0
20.20.10.0      0.0.0.0         255.255.255.0   U     0      0        0
20.40.10.0      0.0.0.0         255.255.255.0   U     0      0        0
[root@localy Diegu]# ip route show
20.30.10.0/24 dev tap00  proto kernel  scope link  src 20.30.10.0
20.20.10.0/24 dev tap01  proto kernel  scope link  src 20.20.10.0
20.40.10.0/24 dev tap22  proto kernel  scope link  src 20.40.10.0
[root@localy Diegu]#
```

Figura 36. Tabla de enrutamiento para el sistema huésped después de generar las TAPs Fuente: Autor

2.) Una vez creadas las interfaces virtuales, se ejecuta VirtualBox para inicializar la configuración de las tarjetas de red en la maquina virtual “fedora_xorp0”, seguido debe ir a, menú de Oracle VirtualBox, *settings* luego *Network*. Ver figura 37. Allí residen los parámetros para habilitar funciones de red a la maquina virtual.
Para la práctica se hizo uso de dos adaptadores de red para los enrutadores. El modo de habilitar los adaptadores fue ”Bridged Adapter” con puente a una primer interface tap00, el tipo de adaptador fue Intel PRO/1000 MT Desktop, con la dirección MAC08002704C30A, la segunda interface tap22, con el tipo de adaptador también Intel PRO/1000 MT Desktop, con la dirección MAC 080027189D59 para la maquina “fedora_xorp0”.

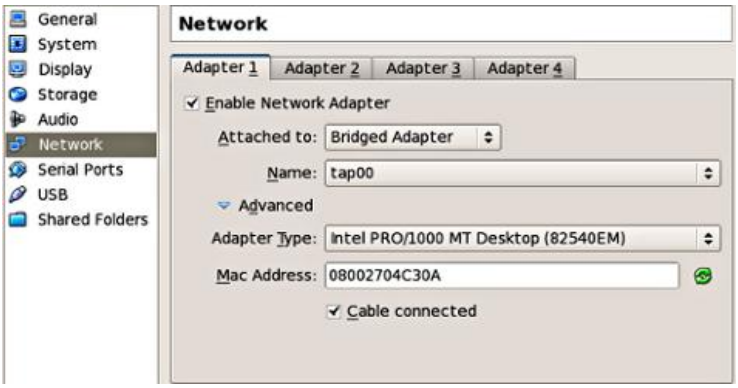


Figura 37 Configuración de las interfaces de red para las maquinas virtuales fuente: Autor

Para el adaptador de la maquina “peppermint-client0”, se utilizó también “Bridge Adapter” con puente a la interface tap22, con dirección MAC 080027E72A3A.
Para el adaptador de la maquina virtual “fedora_xorp1”, se utilizó el “Bridge Adapter” con puente a la interface tap00 y tap 01, correspondiéndole por orden la dirección MAC 0800278A5283 y MAC 0800272C2F24. Para la maquina virtual, peppermint_cliente1, se habilito la interface tap01 con dirección MAC 080027EE0A95 . Una vez se estableció el orden y número de la interface tap que debería tener cada maquina. Se procedió a inicializar la maquina virtual “fedora_xorp0”, ya dado el paso de autentificación del usuario.
Se deberá iniciar la configuración de las interfaces, para esto se cuentan dos formas de lograrlo, una con una herramienta que tiene fedora, la cual se llama con

el comando, “system-config-network”;es una aplicación que ofrece una interfaz grafica, que tiene como función el configurar dispositivos para la red. Ver Figura 38.

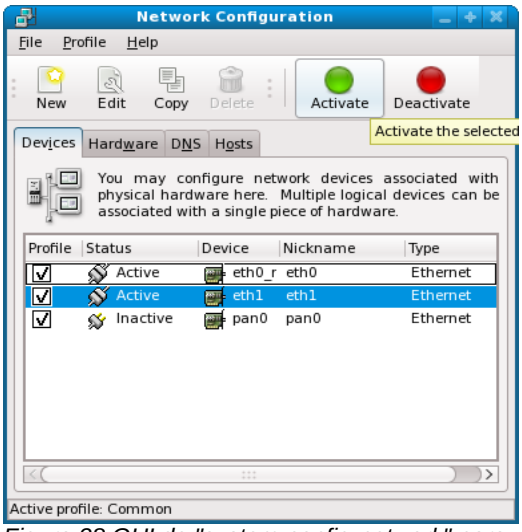


Figura 38 GUI de "system-config-network" para fedora 10 fuente: Autor

La segunda forma es por comandos en el terminal. Para ambos casos se deben tener cuidado en configurar la dirección IP correcta de la sud net correcta para cada dispositivo de red, el modo de averiguarlo es utilizando la dirección MAC de cada interface que se configuro en Oracle VirtualBox setting/network y compararla ya sea por vía comando en el terminal o por el GUI Network configuration. figura 39.

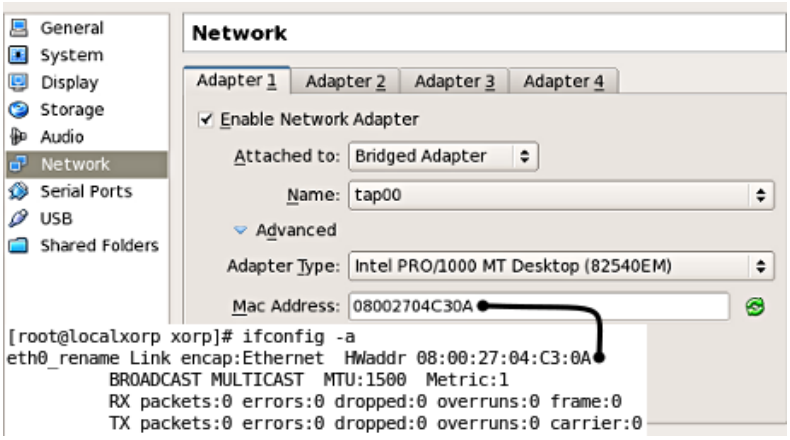


Figura 39. Comparación de las MAC Fuente: Autor

Para tap00 le corresponde la eth0_rename en la maquina virtual “fedora_xorp0”, de este modo se configura en eth0_rename y eth1 una dirección IP acorde a la sud net. ver figura 40.

```
[root@localxorp xorp]# ifconfig eth0_rename 20.30.10.1 netmask 255.255.255.0 up
[root@localxorp xorp]# ifconfig eth1 20.40.10.1 netmask 255.255.255.0 up
[root@localxorp xorp]# ip route show
20.30.10.0/24 dev eth0_rename  proto kernel  scope link  src 20.30.10.1
20.40.10.0/24 dev eth1  proto kernel  scope link  src 20.40.10.1
[root@localxorp xorp]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
20.30.10.0      *               255.255.255.0   U     0      0        0 eth0_rename
20.40.10.0      *               255.255.255.0   U     0      0        0 eth1
[root@localxorp xorp]#
```

Figura 40 Configuración de los dispositivos de red para "fedora_xorp0"
fuente: Autor

Durante las pruebas de encontrar una forma rápida y sencilla de configurar los dispositivos de red, se encontró un posible problema, que consiste, en el menú de Oracle VirtualBox en *setting/Network*, se puede cambiar la dirección MAC de los dispositivos, al realizar el cambio, en la Sistema Operativo fedora 10 (actual huésped de 1.5-XORP) esta información no cambia generando una advertencia. Para solucionarlo se debe editar un registro manualmente, modificando el valor "HWADDR=" /etc/sysconfig/networking/devices/ifcfg-{device}, el cual es la MAC del dispositivo.

3. Para el cliente del "fedora_xorp0". Configurar el dispositivo de red se puede hacer por vía comando en el terminal, o por un GUI determinado que trae Peppermint. ver figura 41

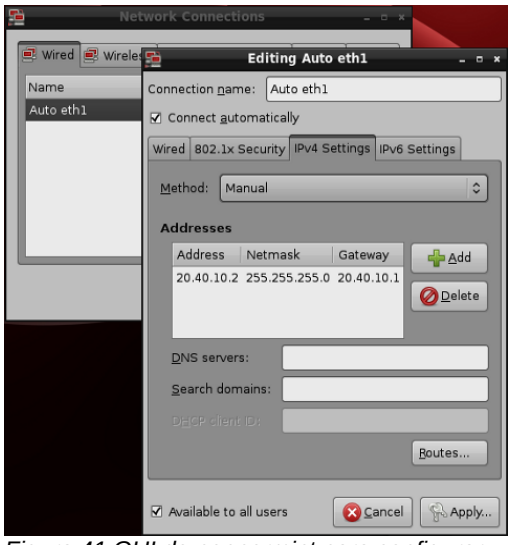


Figura 41 GUI de peppermint para configurar dispositivos de la red fuente :Autor

En definitiva para el dispositivo de red de la maquina virtual "peppermint-client0", se debe verse así en consola, ver figura 42.

```
xorp-cliente0@xorp-cliente0-desktop ~ $ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 08:00:27:e7:2a:3a
          inet addr:20.40.10.2  Bcast:20.40.10.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fee7:2a3a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)

xorp-cliente0@xorp-cliente0-desktop ~ $ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
20.40.10.0      *               255.255.255.0   U     1      0        0 eth1
default         20.40.10.1      0.0.0.0         UG    0      0        0 eth1
xorp-cliente0@xorp-cliente0-desktop ~ $
```

Figura 42 Resultado final en consola después de haber configurado los dispositivos de red fuente: Autor

Una vez se termino de configurar los dispositivos de red, se pasa a probar las “conexiones” de la red por medio de la utilidad “ping”; la parte configurada en la mini red que pertenece al enrutador de la maquina virtual “fedora_xorp0” debe lucir como la figura 43.

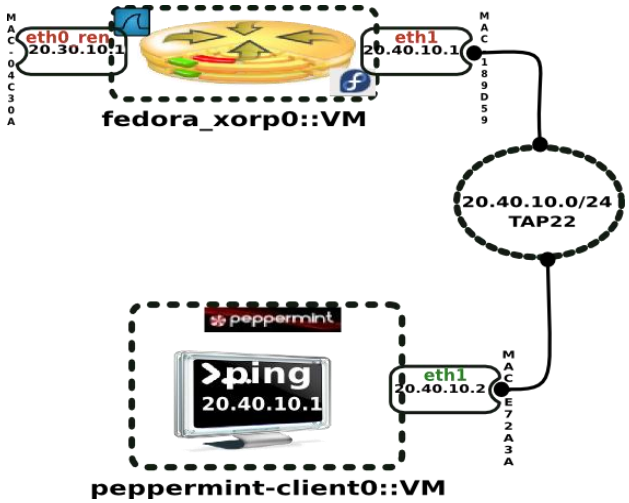


Figura 43 Los dispositivos de red configurados fuente: Autor

Desde la maquina virtual “peppermint-cliente0” hacia los dispositivos de red, eth0_rename y eth1 de la MV “fedora_xorp0”, presenta los siguientes resultados de los ping en la figura 44 (siguiente página), demuestra que la red esta lista a iniciar XORP, permitiendo seguir hacia la configuración del archivo .boot.

```
xorp-cliente0-desktop xorp-cliente0 # ping -c 2 20.40.10.1
PING 20.40.10.1 (20.40.10.1) 56(84) bytes of data.
64 bytes from 20.40.10.1: icmp_seq=1 ttl=64 time=4.97 ms
64 bytes from 20.40.10.1: icmp_seq=2 ttl=64 time=0.034 ms

--- 20.40.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.034/2.503/4.972/2.469 ms
xorp-cliente0-desktop xorp-cliente0 # ping -c 2 20.30.10.1
PING 20.30.10.1 (20.30.10.1) 56(84) bytes of data.
64 bytes from 20.30.10.1: icmp_seq=1 ttl=64 time=0.449 ms
64 bytes from 20.30.10.1: icmp_seq=2 ttl=64 time=0.871 ms

--- 20.30.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.449/0.660/0.871/0.211 ms
xorp-cliente0-desktop xorp-cliente0 #
```

Figura 44 Registro de los ping hacia "fedor_xorp0"
fuente Autor

El procedimiento de configuración de la otra parte de la mini red, que hace referencia a la “fedora_xorp1” y “ peppermint-cliente1” es similar a la que se acabo de presentar, con las precauciones en configurar los dispositivos de red acorde a su MAC.

Solo queda mostrar los registros de los ping en esta parte de la sud red, que corresponden a las maquinas virtuales “fedora_xorp1” y “peppermint-cliente1”. Ver figura 45.

```
xorp-cliente1-desktop xorp-cliente1 # route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
20.20.10.1      *               255.255.255.255 UH    0      0        0 eth0
20.20.10.2      *               255.255.255.254 U     1      0        0 eth0
default         20.20.10.1      0.0.0.0         UG    0      0        0 eth0
xorp-cliente1-desktop xorp-cliente1 # ping -c2 20.20.10.1
PING 20.20.10.1 (20.20.10.1) 56(84) bytes of data.
64 bytes from 20.20.10.1: icmp_seq=1 ttl=64 time=2.64 ms
64 bytes from 20.20.10.1: icmp_seq=2 ttl=64 time=1.26 ms

--- 20.20.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 1.260/1.954/2.648/0.694 ms
xorp-cliente1-desktop xorp-cliente1 # ping -c2 20.30.10.2
PING 20.30.10.2 (20.30.10.2) 56(84) bytes of data.
64 bytes from 20.30.10.2: icmp_seq=1 ttl=64 time=0.778 ms
64 bytes from 20.30.10.2: icmp_seq=2 ttl=64 time=0.433 ms

--- 20.30.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.433/0.605/0.778/0.174 ms
xorp-cliente1-desktop xorp-cliente1 #
```

Figura 45 Registro de los ping hacia "fedora_xorp1" fuente : Autor

La configuración de los dispositivos de red para las dos maquinas virtuales que sirven a la aplicación XORP es opcional pero recomendable, ya que, en el archivo de configuración de XORP (.boot), estos dispositivos se pueden configurar, pero no activar, de este modo se puede asegurar que los enlaces entre las sudnets están hechos; Precauciones que se tomaron en caso de algún error durante la práctica se descartaría la posibilidad de algún enlace roto.

3.2 PRUEBAS OPERACIONALES DE XORP

Ya con los enlaces hechos y corriendo las cuatro instancias de las maquinas virtuales, se procedió a configurar el archivo de ejecución para inicializar XORP. Las capturas de los mensajes en los diferentes protocolos fueron guardados en un archivo, presente en una SD disk, los archivos de configuración .boot para la ejecución de XORP son accesibles en los anexos E, F, G. También se cubrirán aspectos relacionados con las *policy statement*, VLAN y CLICK probados con otros protocolos de enrutamiento.

3.2.1 Reconocimiento de la línea de comandos de XORP ejecutando RIP

El modo de reconocimiento de la CLI *comand line interface* fue a través de la ejecución de RIPv2 (Router Information Protocol) en la mini red descrita anteriormente.

RIP[1] es a *Interior Gateway Protocol*, originalmente definido por el RFC 1058, técnicamente limitado pero introducido por primera vez por desarrolladores de la U.C Berkeley, en maquinas Unix para emplearlo en redes locales, permitiendo a administradores de servidores utilizarlo en sus redes.
RIP también es conocido como *distance-vector routing algorithm*, significa que, hay una “distancia” un costo y un “vector” una dirección para cada destino (el vector sólo muestra el nombre de *the neighboring router*, no todo el camino).
En vez de estar pasando de en cuanto en cuanto la vigencia de un enlace en una red, el enrutador le dice a sus vecinos como esta conformada la red entera, donde el “best link” es calculado, con el menor número de saltos, no necesariamente el mas rápido.
En la versión 2 de RIP[2], introduce principalmente autentificación de los mensajes y especificación implícita de la mascara de red, permitiendo soporte para la detección de subredes. Operacionalmente RIPv2 se diferencia de RIPv1 utilizando mensajes *multicast*, en ves de *broadcast* haciendo uso de la dirección 204.0.0.9, en el puerto 520, como protocolo de transporte utiliza UDP, además incorpora una serie de mejoras como son:

1. Especificación del siguiente salto (*Next Hop Specification*): Típicamente el enrutador que genera el mensaje de propagación de la tabla de enrutamiento es el best next hop. Sin embargo en ciertas inusuales circunstancias, el enrutador que propaga el mensaje podría indicarle que el

1 http://www.networkcomputing.com/unixworld/feature/002.html
2 Network Routing Algorithms, Protocols, and Architectures, 2007.p-150. autor: Deepankar Medhi, Karthikeyan Ramasamy.

siguiente salto es diferente al mismo, tal evento ocurre cuando los dominios de dos enrutadores están conectados a las misma red Ethernet. Como es el caso del ejercicio planteado.

2. *Route tag:* Este campo es para diferenciar enrutadores internos dentro de un dominio de enrutamiento RIP a enrutadores externos. Para enrutadores internos el campo es cero. Si un enrutador es obtenido desde un protocolo de enrutamiento externo el valor del campo es arbitrario o es el número del sistema autónomo de donde provino el enrutador.

3. *Subnet mask*: Este campo permite enrutar basado en la subred, definiendo para cada dirección IP una mascara de red.

3.2.2 Ejecución de XOPR con RIP probando la CLI

En el archivo del código fuente de XORP, hay una carpeta con ejemplos de configuración de los protocolos de enrutamiento que soporta XORP. La ubicación de dicha carpeta esta en to XORP/XORP{VERSION}/rtrmgr/config/. Además del manual de usuario en su versión 1.6 de la documentación oficial, una invaluable fuente de consulta para los temas relacionados de los protocolos de enrutamiento. Para ejecutar RIP, se necesita el archivo de configuración .boot, disponible como anexo E, el cual hace referencia al esquema de la figura 46

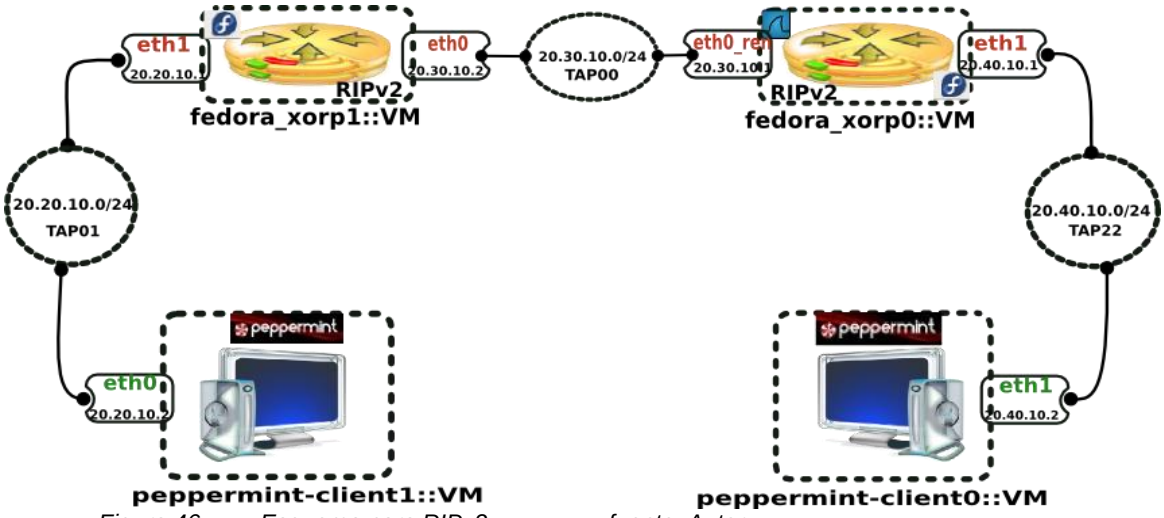


Figura 46 Esquema para RIPv2 fuente: Autor

Para ejecutar a XORP y dar inicio a las operaciones de enrutamiento, el operador debe ubicarse en la carpeta que tenga el binario “xorp_rtrmgr”, para las versiones de la compañía ya desaparecida xorp.inc, el binario se encuentra en el directorio de ubicación XORP/XORP-{VERSION}/rtrmr/. Ver figura 47 (siguiente pagina).

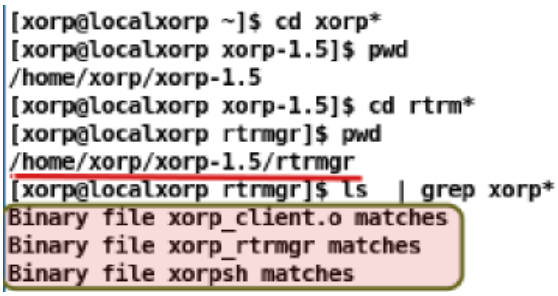


Figura 47 Ubicación del binario xorp_rtrmgr fuente: Autor

Para la versión 1.7 y 1.8 XORP-ct, el binario xorp_rtrmgr se ubica en /usr/local/xorp/sbin. Ver figura 48.

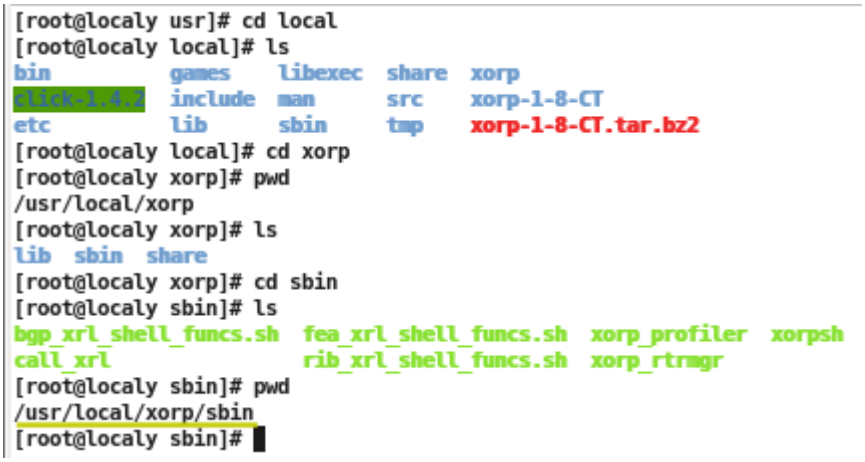


figura 48 Ubicación de xorp_rtrmgr en XORP-ct fuente: Autor

El usuario también puede recorrer las opciones de ejecución con el comando, ./xorp_rtrmgr –h generando una lista de opciones en ejecución. Ver figura 49 (Siguiente Página).

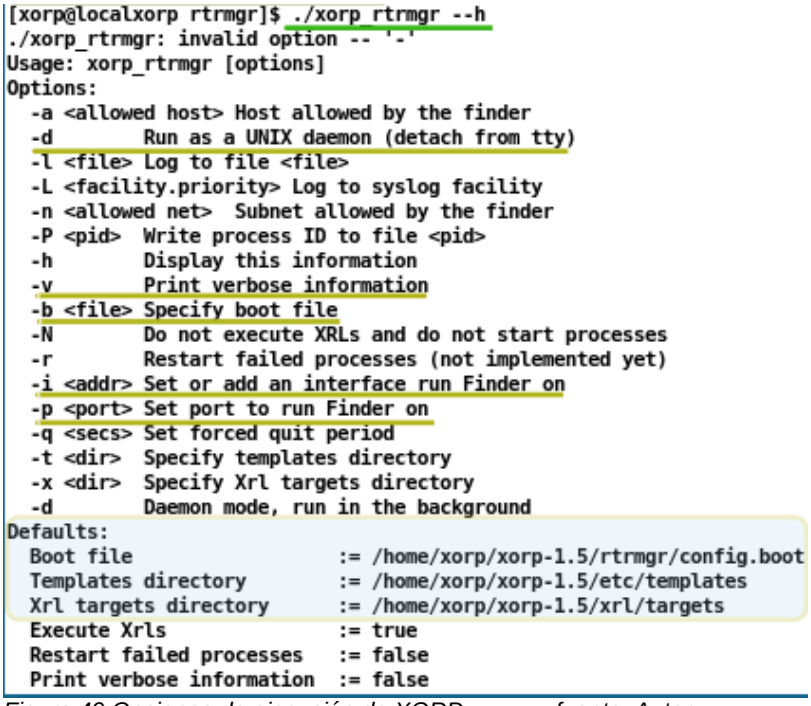


Figura 49 Opciones de ejecución de XORP fuente: Autor

En las opciones que destacan en amarillo para la figura 62, se pueden describir como:

-d ejecutar en modo *Daemon:* es una ejecución en segundo plano

-v imprimir información de las acciones: permite ver en consola como actúa el Módulo rtrmgr, observando como este Módulo llama los xrls en proceso tras proceso, recorriendo los pasos ya vistos en la pre-inicialización de XORP por medio de rtrmgr.

-b para especificarle al binario la ubicación del archivo de configuración .boot.

-i el finder puede residir en una maquina distinta a la que hospeda los *Daemons* de enrutamiento, esta opción permite señalarle donde esta ubicado el finder para hacer las solicitudes de los llamados,

-p apunta a cambiar el puerto por defecto 19999 de los llamados.

Las opciones por defecto que habilitan la ubicación de los archivos de operación de XORP están reseñados de azul claro, la extensión de estos archivos son .boot, .tp , .xif, los dos últimos archivo tipo, son diseñados por el conocedor de programación, como parte de los pasos en la creación de un nuevo proceso. Mas adelante se detallara su construcción en un nuevo capitulo, relacionado a crear procesos en XORP.

Examinando las opciones anteriores para inicializar a XORP, el comando de ejecución sería:

./xorp_rtrmgr –b {ubicación del archivo configuración}/config.boot.

Para esta práctica en particular se introdujo en consola lo siguiente, bajo el árbol

de carpetas /home/XORP/XORP-1.5/rtrmgr, el comando;

./xorp_rtrmgr -b /home/XORP/ripZ.boot

El archivo de configuración ripZ.boot esta disponible como anexo2 y corresponde a la ejecución exitosa de la figura 50.

```
[root@localxorp rtrmgr]# ./xorp_rtrmgr -b  /home/xorp/ripZ.boot
[ 2011/02/23 08:49:22  INFO xorp_rtrmgr:2693 RTRMGR +239 master_conf_tree.cc execute
 ] Changed modules: interfaces, firewall, fea, rib, policy, rip
[ 2011/02/23 08:49:22  INFO xorp_rtrmgr:2693 RTRMGR +96 module_manager.cc execute ]
Executing module: interfaces (fea/xorp_fea)
[ 2011/02/23 08:49:23 INFO xorp_fea MFEA ] MFEA enabled
[ 2011/02/23 08:49:23 INFO xorp_fea MFEA ] CLI enabled
[ 2011/02/23 08:49:23 INFO xorp_fea MFEA ] CLI started
[ 2011/02/23 08:49:23 INFO xorp_fea MFEA ] MFEA enabled
[ 2011/02/23 08:49:23 INFO xorp_fea MFEA ] CLI enabled
[ 2011/02/23 08:49:23 INFO xorp_fea MFEA ] CLI started
[ 2011/02/23 08:49:24  INFO xorp_rtrmgr:2693 RTRMGR +96 module_manager.cc execute ]
Executing module: firewall (fea/xorp_fea)
[ 2011/02/23 08:49:28  INFO xorp_rtrmgr:2693 RTRMGR +96 module_manager.cc execute ]
Executing module: fea (fea/xorp_fea)
[ 2011/02/23 08:49:34  INFO xorp_rtrmgr:2693 RTRMGR +96 module_manager.cc execute ]
Executing module: rib (rib/xorp_rib)
[ 2011/02/23 08:49:36  INFO xorp_rtrmgr:2693 RTRMGR +96 module_manager.cc execute ]
Executing module: policy (policy/xorp_policy)
[ 2011/02/23 08:49:38  INFO xorp_rtrmgr:2693 RTRMGR +96 module_manager.cc execute ]
Executing module: rip (rip/xorp_rip)
[ 2011/02/23 08:49:40  INFO xorp_rtrmgr:2693 RTRMGR +2228 task.cc run_task ] No more
 tasks to run
```

Figura 50 Configuración RIP ejecución exitosa fuente :Autor

En la figura 50 muestra los procesos que son llamados por el rtrmgr, descritos en el archivo de configuración .boot, procesos como interfaces, fea, rip, policy y rib todos necesarios para la ejecución correcta de ripZ.boot.

Para que el operador o usuario interactúe con la línea de comandos debe escribir en consola ./xorpsh, bajo el árbol de carpetas /home/XORP/XORP-1.5/rtrmgr luego si todo ha ido bien, XORP le confirmara que ha entrado a la consola de XORP con un mensaje de bienvenida:
“Welcome to XORP on {nombre de la maquina que hospeda a XORP}”.

Lo que queda por destacar es la posibilidad de ver la tabla de enrutamiento y comandos referentes al protocolo RIP.
En la figura 51 referenciado de azul, esta la tabla de enrutamiento de “fedora_xorp0” que indica que cualquier dirección IP con destino 20.20.10./24 y 20.30.10.0/24 tiene salida por el dispositivo de red eth0_rename hacia la dirección 20.30.10.2.

A pesar que el llamado del comando ./xorpsh se hizo desde root, este no tiene permisos para cambiar el archivo de configuración .boot, para ello debe utilizar el usuario que habilito desde el grupo XORP. Ver figura 51, referencia roja.

El xorpsh también presenta un mecanismo de corrección en la sintaxis, como se presento en la versión del CD autoejecutable, los comandos del shell referentes a

la configuración son commit, delete, edit, exit, help, load, quit, run, save, set, show, top, up. Cada uno con un label de información, accesible con el carácter clave “?”.

```
[root@localxorp rtrmgr]# ./xorpsh
Welcome to XORP on localxorp.xorp.x
root@localxorp.xorp.x> show route table ?
Possible completions:
  ipv4                Show IPv4 routes
  ipv6                Show IPv6 routes
root@localxorp.xorp.x> show route table ipv4 unicast rip
20.20.10.0/24   [rip(120)/1]
                > to 20.30.10.2 via eth0_rename/eth0_rename
20.30.10.0/24   [rip(120)/1]
                > to 20.30.10.2 via eth0_rename/eth0_rename
root@localxorp.xorp.x> ?
Possible completions:
  configure           Switch to configuration mode
  exit                Exit this command session
  help                Provide help with commands
  ping                Ping a hostname or IP address
  ping6               Ping an IPv6 hostname or IPv6 address
  quit                Quit this command session
  show                Display information about the system
  traceroute          Trace the IP route to a hostname or IP address
  traceroute6         Trace the IPv6 route to a hostname or IPv6 address
root@localxorp.xorp.x> configure
ERROR: You do not have permission for this operation.
root@localxorp.xorp.x> exit
[root@localxorp rtrmgr]# su xorp
[xorp@localxorp rtrmgr]$ ./xorpsh
Welcome to XORP on localxorp.xorp.x
xorp@localxorp.xorp.x> configure
Entering configuration mode.
There are no other users in configuration mode.
[edit]
xorp@localxorp.xorp.x#
```

Figura 51 Ejecución del xorpsh fuente : Autor

Después de ejecutar ripZ.boot la tabla de enrutamiento del sistema huésped de XORP cambia .Ver figura 52. Como observación.

```
[xorp@localxorp rtrmgr]$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
20.30.10.0      *               255.255.255.0   U     0      0        0 eth0_rename
20.20.10.0      20.30.10.2      255.255.255.0   UG    1      0        0 eth0_rename
20.40.10.0      *               255.255.255.0   U     0      0        0 eth1
[xorp@localxorp rtrmgr]$
```

Figura 52 Tabla de enrutamiento del sistema huésped "fedora-xorp0" fuente: Autor

A continuación se presentarán los comandos de RIP habilitados desde CLI de XORP, se tiene en referencia verde y amarilla ya fueron presentados anteriormente cuando se hizo el estudio del cd auto ejecutable 1.6 XORP.inc.

Las referencias azules y magenda, muestran respectivamente a cual dirección y dispositivo de red, debe esperar intercambio de mensajes tipo RIP, además de verificar el estado del enlace. Ver figura 53 (siguiente página).

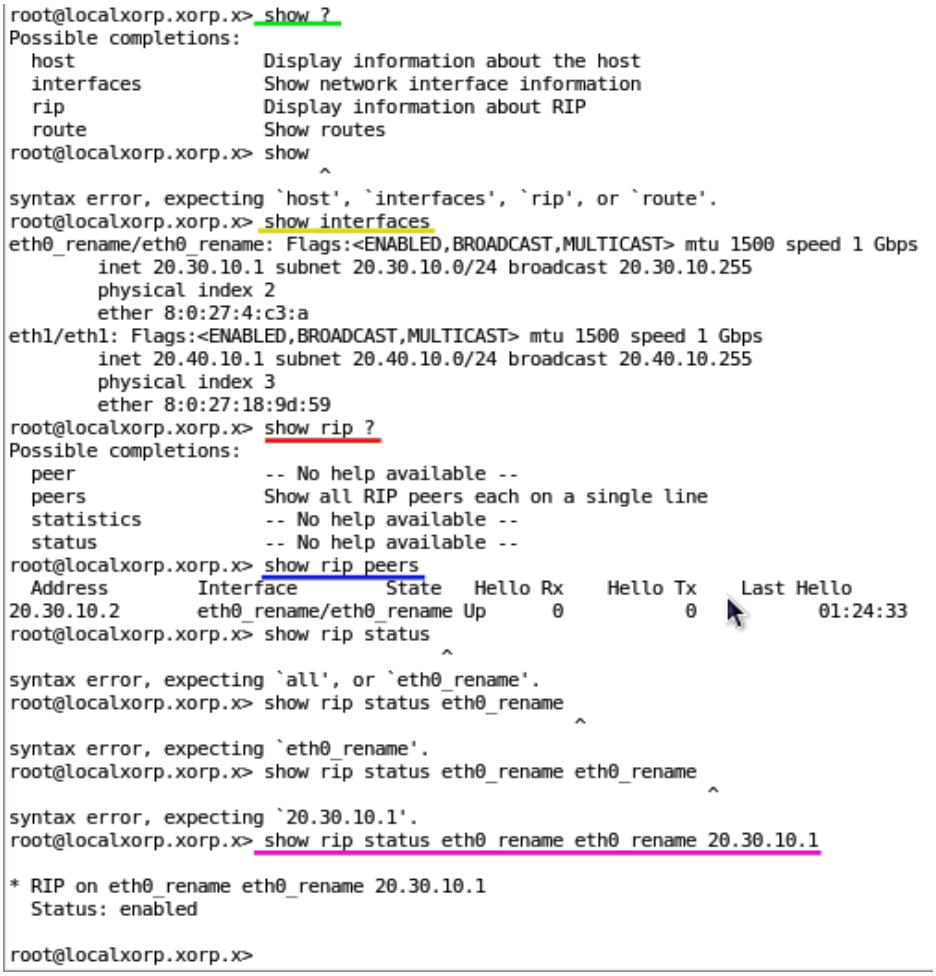


Figura 53. comandos de RIP en el xorpsh fuente :Autor

La referencia en verde (figura 53) indica las opciones de comandos para completar, la referencia amarilla muestra el resultado del comando "*show interfaces*" donde presenta las interfaces de red habilitadas desde XORP. La referencia roja muestra las opciones para requerir información sobre el protocolo RIP. La línea de comandos es habilitada por el Módulo xorpsh.

Para la figura 54, se muestran algunas estadísticas de los paquetes recibidos en el protocolo, además también muestran en *update* que es aprox. de 30 segundos como se puede ver en el tiempo referenciado en rojo. (Siguiente Página)

```
xorp@localxorp.xorp.x> show rip peer statistics eth0_rename eth0_rename 20.30.10.1

* RIP statistics for peer 20.30.10.2 on eth0_rename eth0_rename 20.30.10.1

  Last Active at 01:00:32
  Counter                                 Value
  ------------------------------- ---------------
  Total Packets Received                     27
  Request Packets Received                    0
  Update Packets Received                    27
  Bad Packets Received                        0
  Authentication Failures                     0
  Bad Routes Received                         0
  Routes Active                               1
xorp@localxorp.xorp.x> show rip peer statistics eth0_rename eth0_rename 20.30.10.1

* RIP statistics for peer 20.30.10.2 on eth0_rename eth0_rename 20.30.10.1

  Last Active at 01:01:06
  Counter                                 Value
  ------------------------------- ---------------
  Total Packets Received                     28
  Request Packets Received                    0
  Update Packets Received                    28
  Bad Packets Received                        0
  Authentication Failures                     0
  Bad Routes Received                         0
  Routes Active                               1
xorp@localxorp.xorp.x>
```

Figura 54 comando "rip peer statistics" Fuente : Autor

A continuación se presentarán algunas imágenes de las capturas logradas con el *wireshark*, capturas en la interface de red eth0_rename.

En la figura 55 en el mensaje RIP se identifica el puerto 520, la dirección ip 224.0.0.9 como mensaje tipo multidifusión de destino, utilizando a UDP hacia el puerto 520 y el valor de la métrica 16.

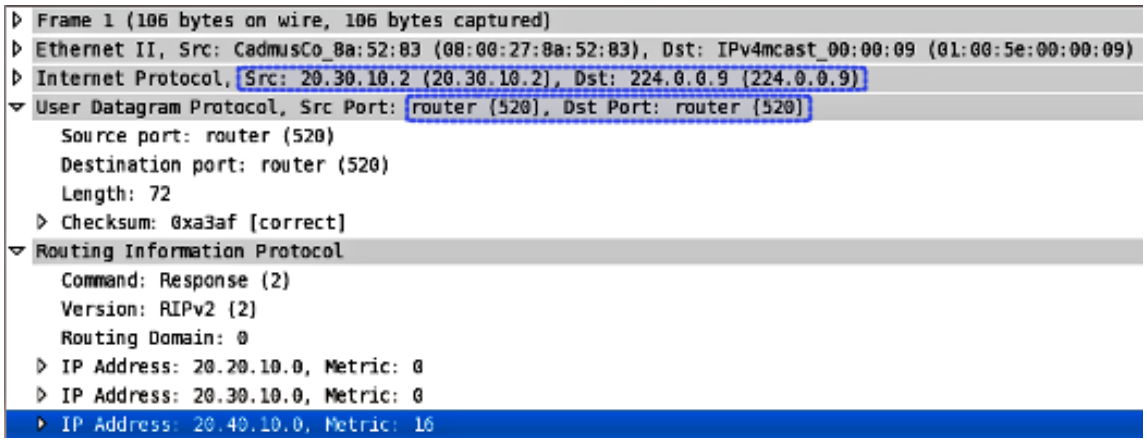


Figura 55 Mensaje RIP en el wireshark fuente: Autor

Para la Figura 56 (siguiente página) se tiene los valores de los campos que caracterizan a RIPv2, como son next-hop, route tag , Netmask y las redes que el enrutador propagó como son 20.20.10.0, 20.30.10.0 y 20.40.10.0.

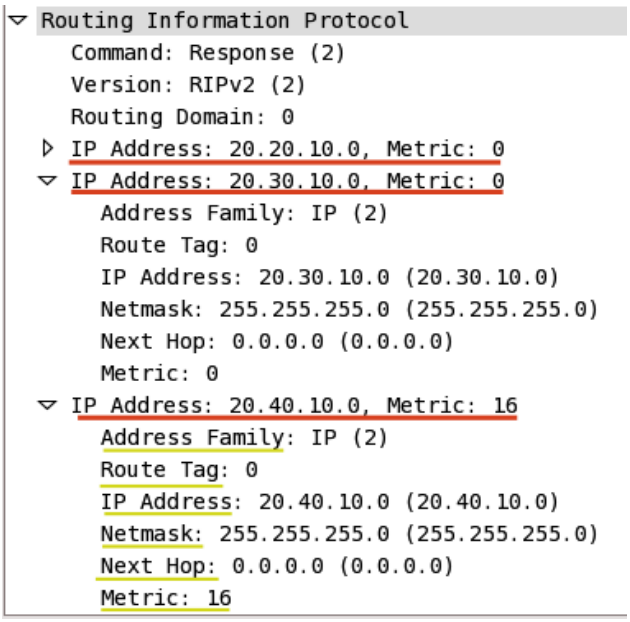


Figura 56 los campos de RIPv2 propagados desde eth0_rename fuente: Autor

En la figura 57 resalta el tiempo del update que es aproximadamente de 30 sg, que sale de la diferencia entre la captura número 9 y 11.

No.	Time	Source	Destination	Protocol	Info
5	28.628650	20.30.10.1	224.0.0.9	RIPv2	Response
6	30.215744	20.30.10.1	224.0.0.22	IGMP	V3 Membership Report / Join group 224.0.0.9
7	55.194879	20.30.10.1	224.0.0.9	RIPv2	Response
8	60.363663	20.30.10.2	224.0.0.9	RIPv2	Response
9	83.891337	20.30.10.1	224.0.0.9	RIPv2	Response
10	88.915397	20.30.10.2	224.0.0.9	RIPv2	Response
11	110.413841	20.30.10.1	224.0.0.9	RIPv2	Response
12	117.712960	20.30.10.2	224.0.0.9	RIPv2	Response
13	141.493162	20.30.10.1	224.0.0.9	RIPv2	Response
14	149.837077	20.30.10.2	224.0.0.9	RIPv2	Response

Figura 57 mensaje RIP, resaltando el tiempo UPDATE fuente: Autor

Para el protocolo de enrutamiento vectorial de distancia RIP, es bastante sencillo en su configuración, y esa misma sencillez ha sido una de las razones para que sea actualmente poco utilizado, ya que esta enfocado para redes pequeñas que no tengan posibilidad de crecer.
También se logró, por medio de la utilidad de red netcat habilitar una comunicación *sockets tcp* entre los dos clientes, los registros de esta comunicación están presentes en el archivo .pcap, accesible en una SD disk disponible que trae el presente trabajo.

XORP es bastante estable en operaciones de la línea de comandos, la usabilidad en modo texto es muy buena ya que cuenta con un mecanismo de ayuda, que no sólo indica donde el error esta presente, sino que propone las opciones para la corrección, además la sintaxis a modo de nodos para configurar el archivo .boot

es bastante intuitiva, también se encontró un mecanismo de autocompletar el comando, al digitar la primer letra del comando a escribir y darle espacio este completa el comando.
El único problema que generaría el xorpsh, sería el caso que, durante el proceso de pre - instalación, se ignore la parte de la creación del grupo XORP y definir al usuario XORP.

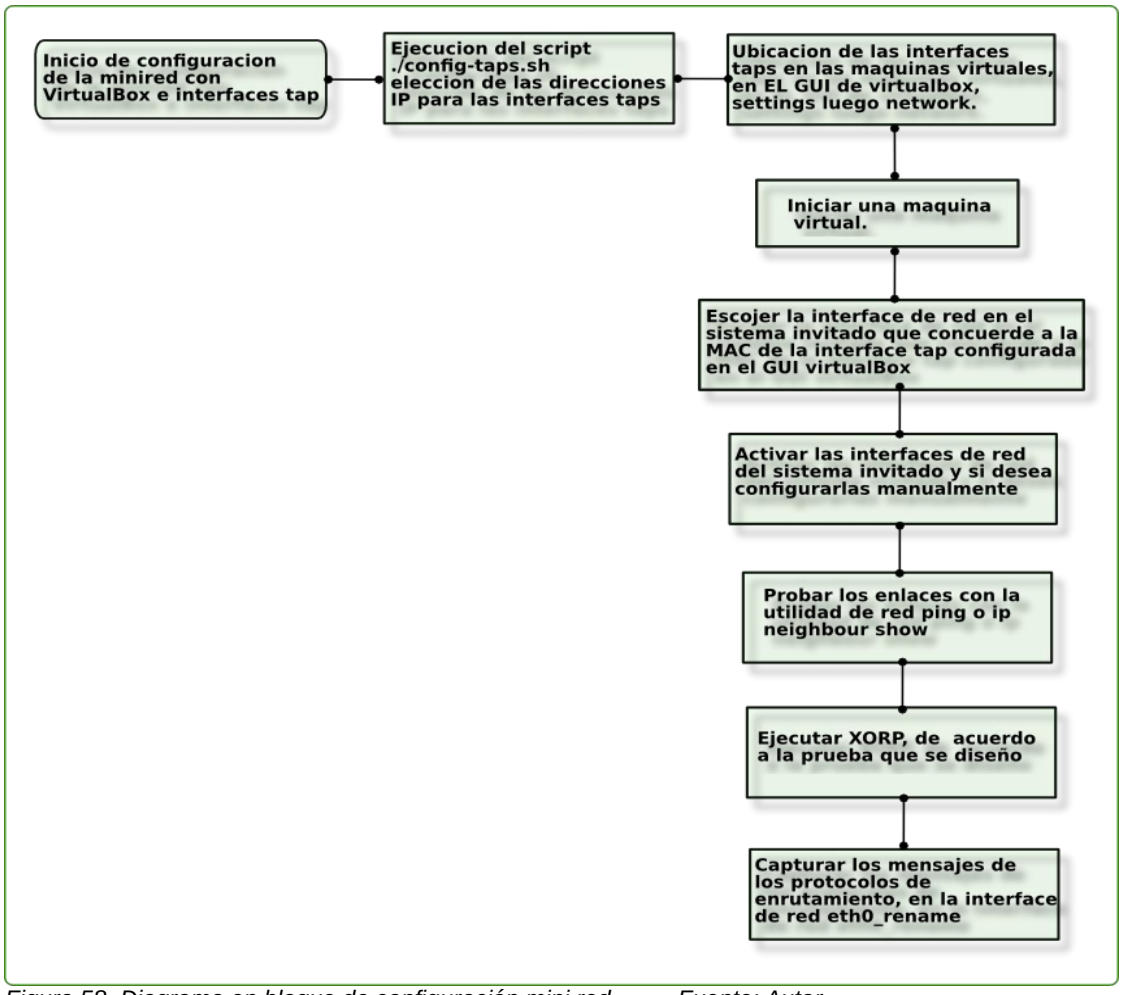


Figura 58 Diagrama en bloque de configuración mini red Fuente: Autor

En el diagrama de bloques de la figura 58 muestra el proceso para iniciar la mini red con el fin de probar el éxito de la ejecución de los protocolos de enrutamiento y capturar los mensajes de comunicación de los enrutadores, probando a XORP en un modo de alto nivel. El diagrama en bloque de la figura 70 no contempla la posibilidad de tratar errores que surjan en el proceso de la configuración de la mini red.

3.2.3 Prueba de XORP En policy-statement y redistribución de enrutadores en IBGP

Para tener una fácil administración, enrutadores de grandes redes son agrupados en Sistemas Autónomos(SA), que son controlados independientemente por organizaciones y compañías. Protocolos de enrutamiento interior como RIP y OPPF es usado para comunicar información de enrutamiento dentro de los Sistemas Autonomos[1] otro grupo de protocolos de enrutamiento se dedica a la comunicación entre Sistemas Autónomos llamado protocolos de enrutamiento exterior entre estos esta BGP.
Desde Junio de 1989 se han presentado mejoras para el rfc 1105 de BGP, hasta llegar a la versión BGP-4, donde la función primaria es intercambiar información de la red entre sistemas autónomos permitiendo a cada sistema autónomo enviar y recibir mensajes eficientemente hacia cada uno.

BGP [2] en vez de enviar información de la red, en términos de destino y distancia, publican información de las redes en base a direcciones y descripciones de caminos para buscar posibles destinos. Esto significa que BGP usa un algoritmo de *path-vector* en vez de *distance-vector*. Cada comunicación, provee una considerable información acerca de la secuencia de enrutadores para llegar a destino, esta información es almacenada en una base de información de enrutamiento.

En la parte operacional dispositivos BGP trabaja con TCP[3], creando una sesión de comunicación entre enrutadores en los bordes de sistemas autónomos, escuchando por el puerto 179. Esta comunicación necesita estar siempre vigente, ya que es usada por los dos lados para enviar información, el tipo de mensajes creados por el protocolo poseen un formato y tarea especifica.

BGP define cuatro tipos de mensajes, OPEN, UPDATE, KEEPALIVE y NOTIFICATION.
El mensaje OPEN, es el primer mensaje enviado para establecer la sesión de comunicación, después de que se produjo con éxito la conexión TCP, el mensaje OPEN contiene la versión de BGP y el AS.
El mensaje UPDATE, es el mensaje clave para las operaciones BGP, este mensaje porta información de los prefijos IP, si un dispositivo BGP tiene nueva información, este propaga el mensaje al *peering* del BGP vecino, por medio de un mensaje UPDATE.
Una vez que la sesión BGP esta hecha y corriendo, el mensaje KEEPALIVE es

---

1 Charles M. Kozierok.The TCP/IP Guie.version 3.0.2005.p. 765-769
2 Charles M. Kozierok.The TCP/IP Guie.version 3.0.2005.p. 778.
3 Network Routing Algorithms, Protocols, and Architectures, 2007.p-239. autor: Deepankar Medhi, Karthikeyan Ramasamy.

generado de vez en cuanto, el tiempo es establecido en que cada mensaje se genera depende por el parámetro *hold time.*
El mensaje de NOTIFICATION es para avisar que la sesión se cerró, esto ocurre cuando hay algún error y requiere esta acción.

3.2.3.1 Ejecución de la prueba para BGP

Para la prueba de ejecución de BGP en XORP, se basa en el esquema de la figura 59, donde los dos dispositivos BGP se encuentran en el AS 6500, con un enlace de punto a punto en la dirección 66.7.8.x. El procedimiento para configurar los enlaces y dispositivos de red, es el mismo que se presento en pre-inicialización de la mini red, la diferencia radica en el orden de los dispositivos de red virtuales taps han cambiado así como sus direcciones.

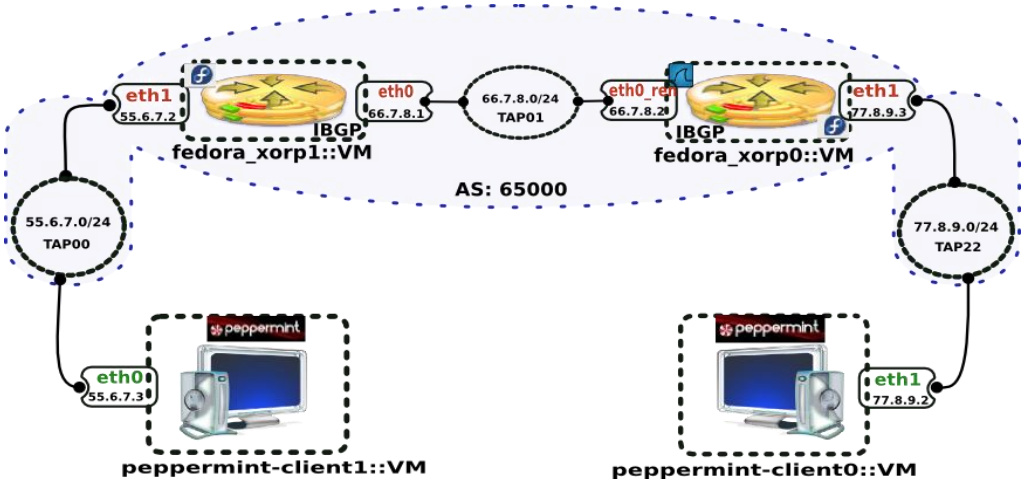


Figura 59 Diagrama para probar IBGP en XORP Fuente : Autor

Para comprobar en la práctica si están hechos las conexiones punto a punto, existe un comando, el "ip neighbour show". En la figura 60 se demuestra como los enlaces existen para "fedora_xorp1" entre los dispositivos de red de la maquina Virtual "peppermint-client1" eth0 con ip 55.6.7.3 y eth0_rename "fedora_xorp0" con IP 66.7.8.2.

```
[xorp@localxorp ~]$ su root
Password:
[root@localxorp xorp]# ifconfig eth0 66.7.8.1 netmask 255.255.255.0 up
[root@localxorp xorp]# ifconfig eth1 55.6.7.2 netmask 255.255.255.0 up
[root@localxorp xorp]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
66.7.8.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
55.6.7.0        0.0.0.0         255.255.255.0   U     0      0        0 eth1
[root@localxorp xorp]# ip neighbour show
66.7.8.2 dev eth0 lladdr 08:00:27:04:c3:0a STALE
[root@localxorp xorp]# ip neighbour show
55.6.7.3 dev eth1 lladdr 08:00:27:ee:0a:95 REACHABLE
66.7.8.2 dev eth0 lladdr 08:00:27:04:c3:0a STALE
[root@localxorp xorp]# ip route show
66.7.8.0/24 dev eth0  proto kernel  scope link  src 66.7.8.1
55.6.7.0/24 dev eth1  proto kernel  scope link  src 55.6.7.2
[root@localxorp xorp]#
```

Figura 60 Muestra enlaces point-to-point, con el comando ip neighbour show fuente: Autor

```
[xorp@localxorp ~]$ su root
Password:
[root@localxorp xorp]# ifconfig eth0_rename 66.7.8.2 netmask 255.255.255.0 up
[root@localxorp xorp]# ifconfig eth1 77.8.9.3 netmask 255.255.255.0 up
[root@localxorp xorp]# ip neighbour show
[root@localxorp xorp]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
66.7.8.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0_ren
ame
77.8.9.0        0.0.0.0         255.255.255.0   U     0      0        0 eth1
[root@localxorp xorp]# ip neighbour show
77.8.9.2 dev eth1 lladdr 08:00:27:e7:2a:3a REACHABLE
[root@localxorp xorp]# ip route show
66.7.8.0/24 dev eth0_rename  proto kernel  scope link  src 66.7.8.2
77.8.9.0/24 dev eth1  proto kernel  scope link  src 77.8.9.3
[root@localxorp xorp]# ip neighbour show
66.7.8.1 dev eth0_rename lladdr 08:00:27:2c:2f:24 STALE
[root@localxorp xorp]# ip neighbour show
77.8.9.2 dev eth1 lladdr 08:00:27:e7:2a:3a DELAY
66.7.8.1 dev eth0_rename lladdr 08:00:27:2c:2f:24 STALE
[root@localxorp xorp]# ip neighbour show
77.8.9.2 dev eth1 lladdr 08:00:27:e7:2a:3a REACHABLE
66.7.8.1 dev eth0_rename lladdr 08:00:27:2c:2f:24 STALE
[root@localxorp xorp]#
```

figura 61 Verificación de los enlaces creados en la interface tap 22 fuente: autor

La figura 61 muestra la utilización del comando “ip neighbour show” para la MV (maquina virtual) “fedora_xorp0”, mostrando las interfaces vecinas de la MV. Ya demostrado la viabilidad de la red para realizar la prueba de BGP se procedió a ejecutar el archivo bgpZ.boot. El éxito de la ejecución se ve en la figura 62, donde se habilita el proceso BGP en XORP.

```
[root@localxorp rtrmgr]# ./xorp_rtrmgr -b  /home/xorp/bgpZ.boot
[ 2011/03/06 12:58:33  INFO xorp_rtrmgr:2751 RTRMGR +239 master_conf_tree.cc execut
e ] Changed modules: interfaces, firewall, fea, rib, policy, static_routes, bgp
[ 2011/03/06 12:58:33  INFO xorp_rtrmgr:2751 RTRMGR +96 module_manager.cc execute ]
 Executing module: interfaces (fea/xorp_fea)
[ 2011/03/06 12:58:34 INFO xorp_fea MFEA ] MFEA enabled
[ 2011/03/06 12:58:34 INFO xorp_fea MFEA ] CLI enabled
[ 2011/03/06 12:58:34 INFO xorp_fea MFEA ] CLI started
[ 2011/03/06 12:58:34 INFO xorp_fea MFEA ] MFEA enabled
[ 2011/03/06 12:58:34 INFO xorp_fea MFEA ] CLI enabled
[ 2011/03/06 12:58:34 INFO xorp_fea MFEA ] CLI started
[ 2011/03/06 12:58:35  INFO xorp_rtrmgr:2751 RTRMGR +96 module_manager.cc execute ]
 Executing module: firewall (fea/xorp_fea)
[ 2011/03/06 12:58:39  INFO xorp_rtrmgr:2751 RTRMGR +96 module_manager.cc execute ]
 Executing module: fea (fea/xorp_fea)
[ 2011/03/06 12:58:45  INFO xorp_rtrmgr:2751 RTRMGR +96 module_manager.cc execute ]
 Executing module: rib (rib/xorp_rib)
[ 2011/03/06 12:58:47  INFO xorp_rtrmgr:2751 RTRMGR +96 module_manager.cc execute ]
 Executing module: policy (policy/xorp_policy)
[ 2011/03/06 12:58:50  INFO xorp_rtrmgr:2751 RTRMGR +96 module_manager.cc execute ]
 Executing module: static_routes (static_routes/xorp_static_routes)
[ 2011/03/06 12:58:52  INFO xorp_rtrmgr:2751 RTRMGR +96 module_manager.cc execute ]
 Executing module: bgp (bgp/xorp_bgp)
[ 2011/03/06 12:58:58  INFO xorp_rtrmgr:2751 RTRMGR +2228 task.cc run_task ] No mor
e tasks to run
```

Figura 62 Muestra ejecución exitosa de bpgZ.boot fuente: Autor

A continuación se presentarán los mensajes BGP capturados con el wireshark en la interface de red eth0_rename. Estos mensajes están disponibles con la extensión .cap en una memoria SD disk que trae el presente trabajo.

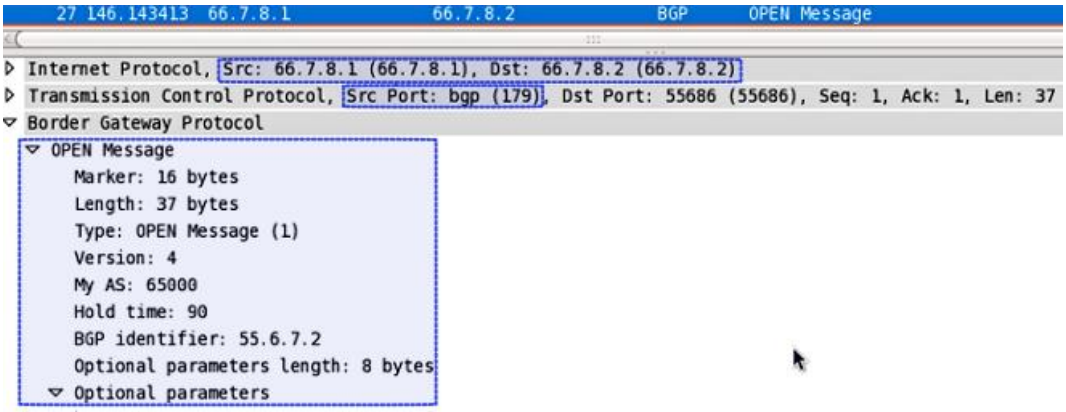


Figura 63 Mensaje OPEN en BGP *fuente: Autor*

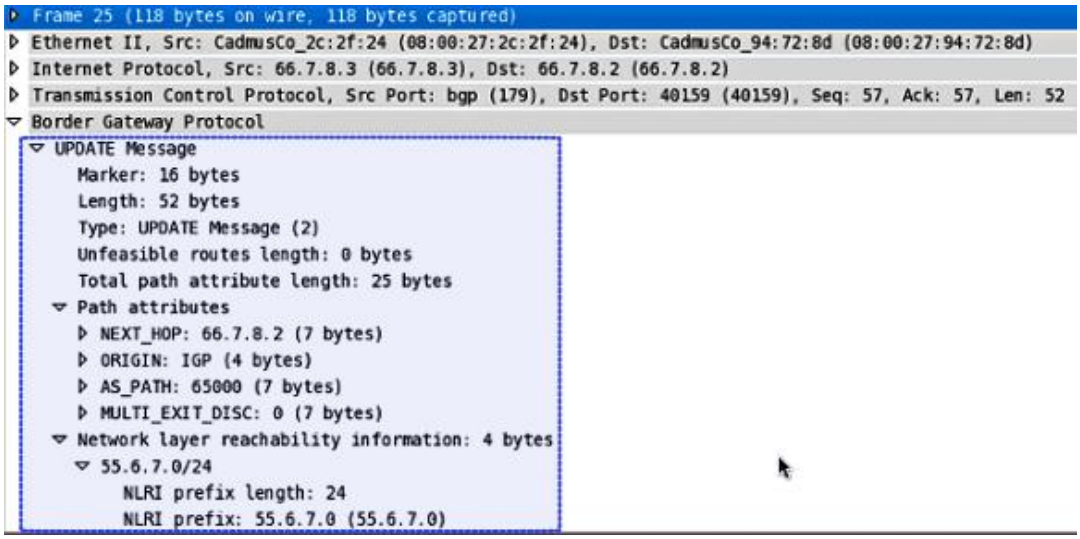


Figura 64 Mensaje UPDATE de BGP *fuente: Autor*

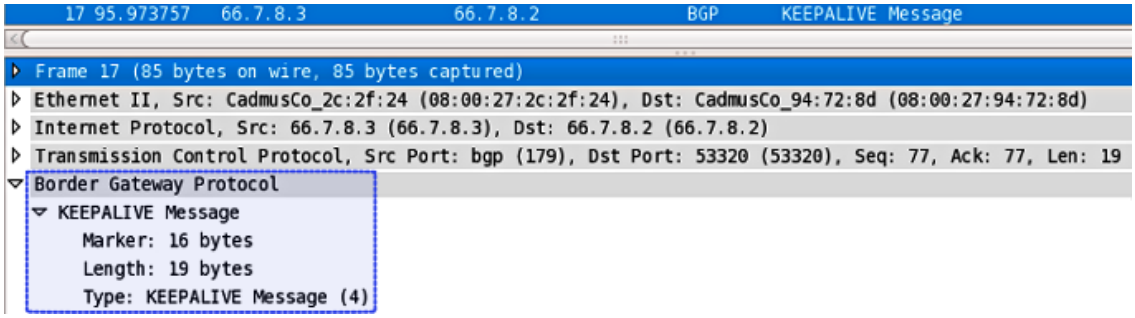


Figura 65 Mensaje KEEPALIVE de BGP fuente: Autor

Para generar el mensaje de NOTIFICATION, se procedió a cambiar el AS por un valor diferente a 65000, valor que debería esperar el enrutador “fedora_xorp0” en el mensaje OPEN.

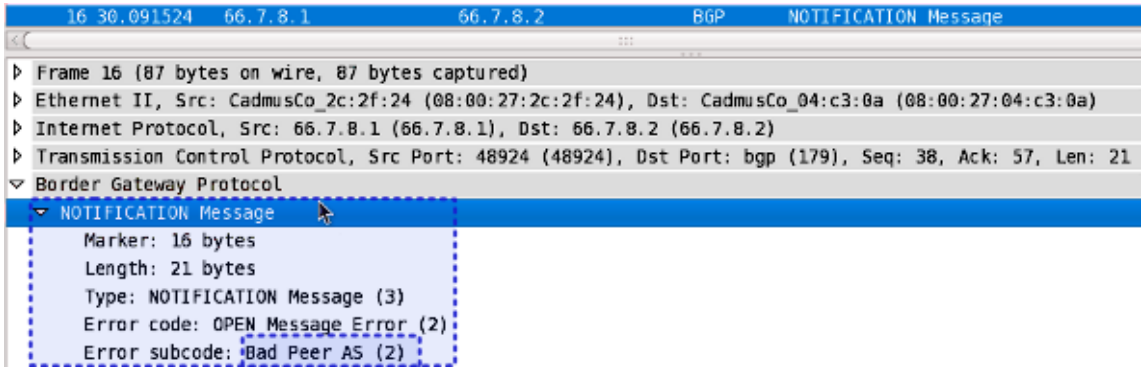


Figura 66 Mensaje NOTIFICATION Fuente :Autor

También se utilizó la utilidad de red “traceroute” para el cliente1 para observar los saltos del paquete hacia el destino 77.8.9.2 . Ver figura 67.

```
xorp-cliente1-desktop xorp-cliente1 # traceroute 77.8.9.2
traceroute to 77.8.9.2 (77.8.9.2), 30 hops max, 60 byte packets
 1  55.6.7.2 (55.6.7.2)  0.442 ms  0.879 ms  0.763 ms
 2  66.7.8.2 (66.7.8.2)  3.873 ms  3.767 ms  3.663 ms
 3  77.8.9.2 (77.8.9.2)  3.548 ms  3.293 ms  2.082 ms
```

Figura 67 Traceroute para cliente1 fuente:autor

Ya probado XORP en su funcionamiento con BGP, al menos para esta pequeña mini red, se detallara las partes del archivo de configuración bgpZ.boot, accesible en el anexo F.

3.2.3.2 Descripción del archivo bgpZ.boot para ejecutar XORP en la Mini red.

En el anexo F del presente trabajo se encuentra el archivo de configuración para XORP que se utilizó para esta prueba, sólo se presento el .boot de una sola maquina “xorp_fedora1”, sin embargo todos los archivos que se utilizaron para las

pruebas son accesibles en una memoria sd disk que trae el presente trabajo.

La primer parte del anexo corresponde a la definición de las interfaces de red,

```
interfaces {
    interface eth0 {
                vif eth0 {
                    address 66.7.8.1 {
                          prefix-length: 24
                     }
                }
    interface eth1 {
                vif eth1 {
                     address 55.6.7.2 {
                          prefix-length: 24
                       }
                 }
          }
 }
```

Se definen las direcciones IP de los dispositivos de red de los enrutadores para “fedora_xorp1”. Exponiendo también el prefijo de distancia.

```
 fea {
     unicast-forwarding4 {
         disable: false
    }
}
```

La fea se configura para que reciba tráfico tipo IPv4 y *unicast.*

```
policy {
    policy-statement to-bgp {
  term include {
       from  {
         protocol: "static"
              }
```

Las declaración de Políticas, se refieren a que enrutadores aceptar y cuales propagar según la coincidencia de una regla. Además de proveer el mecanismo de distribuir enrutadores aprendiendo de otros protocolos y propagarlos hacia otros protocolos, este concepto se llama *re- distribución de enrutadores.* La forma de configurar la declaración de políticas se inicia con la definición del nodo *policy*, luego sigue el sínodo *policy-statement* que define el nombre de la política a establecer, luego esta *terma,* en este bloque se ejecutan tareas como recibir, descartar, aceptar, clasificar y también modificaciones que se le pueden hacer a

los paquetes para luego propagarlos.[1] Dentro de este bloque *term* se tienen las palabras claves *from*, *to* y *then.*
Para *from* realiza operaciones de inclusión o exclusión de enrutadores, para esto utiliza dentro del bloque variables, que sirven para hacer una regla. La coincidencia con esta regla permite luego realizar acciones con los enrutadores.
En la figura 68 se muestra una tabla de las variables, operadores, tipo del argumento y descripción, herramientas para armar una regla para los enrutadores a propagar o recibir.

Variable	Operator	Argument type	Semantics
protocol	:	txt	Matches the protocol via which the route was learnt. Only valid for export policies. Used in route redistribution.
network4	: (or ==) longer (or <) orlonger (or <=) shorter (or >) orshorter (or >=) not (or !=)	ipv4net	Matches the prefix of an IPv4 route. Matches the route with a longer netmask. Matches longer or exact route. Matches the route with a shorter netmask. Matches shorter or exact route. Does not match route.
network6	: longer orlonger shorter orshorter not	ipv6net	Same as IPv4, but for IPv6 prefixes.
network4-list	:	txt	Matches if the named IPv4 set contains the route.
network6-list	:	txt	Matches if the named IPv6 set contains the route.
prefix-length4	:	u32range	Matches if the IPv4 route has a prefix length within the specified range.
prefix-length6	:	u32range	Matches if the IPv6 route has a prefix length within the specified range.
nexthop4	:	ipv4range	Matches if the IPv4 next-hop of the route lies within the specified range.
nexthop6	:	ipv6range	Matches if the IPv6 next-hop of the route lies within the specified range.
tag	:	u32range	Matches the route tag. Routes can be arbitrarily tagged (labeled) via policies.
policy	:	txt	Executes a policy as a subroutine. If the policy rejects the route, false is returned and no match occurs. Otherwise, true is returned and the match is successful.

Figura 68 Tabla de las variables del bloque from fuente: user-manual.dvi

Para la prueba que presenta este trabajo se utilizó *protocolo con los parámetros "connected"* y "static" además de la variable *network4* y los operadores !==, ==.
En el archivo *template* de las políticas del enrutador, permite extraer información

```
protocol {
    %help: short "Protocol from which route was learned";
    %allow: $(@) "bgp" %help: "BGP routes";
    %allow: $(@) "connected" %help: "Directly connected sub-network routes";
    %allow: $(@) "olsr4" %help: "OLSRv1 IPv4 routes";
    %allow: $(@) "ospf4" %help: "OSPF IPv4 routes";
    %allow: $(@) "ospf6" %help: "OSPF IPv6 routes";
    %allow: $(@) "rip" %help: "RIP routes";
    %allow: $(@) "ripng" %help: "RIPng routes";
    %allow: $(@) "static" %help: "Static routes";
```

figura 69 El nodo protocol del archivo plantilla policy.tp fuente:autor

acerca, de la implementación del proceso *policy,* ver figura 69.

En uno de los nodos llamado *protocolo*, permite ver de cuales protocolos es permitido aprender rutas, para el caso de *connected* aprende de las redes que están conectadas al enrutador, para s*tatic* los enrutadores son cargados por el protocolo de enrutamiento estático.

```
protocols {
   static {
     route 77.8.9.0/24 {
        next-hop: 66.7.8.2
                  }
               }
      bgp {
         export: "to-bgp"

        bgp-id: 55.6.7.2
        local-as: 65000

        peer 66.7.8.2  {
           local-ip: 66.7.8.1
           as: 65000
          next-hop: 66.7.8.1
        }
     }
}
```

Luego en el nodo *protocols se* configuraron dos protocolos "static" y "bgp", el protocolo estático de enrutamiento presenta dos variables la primera *route*, que describe que dirección que se debe enrutar, la segunda *next-hop* la salida determinada de los paquetes que poseen el *route* especificado.

Para bgp se presenta en el nodo *export, que sirve para hacer* el llamado a la configuración de la *policy-statement,* también este espacio puede utilizarse *import*, aunque no se utilizó, este tiene un efecto diferente en el enrutador dependiendo el tipo de algoritmo de enrutamiento ya sea[1] *distance-vector* o *link-state.* El *bgp-id* identifica al enrutador que generalmente puede ser una de las direcciones ip en las interfaces de red provistas por el enrutador, en la figura 75 en el mensaje OPEN hay un campo para este valor. *local-as* identifica a Sistema Autónomo al que pertenece el enrutador.

---

1 Andrea Bittau, Mark Handley.University College London.Decoupling Policy from Protocols:Implementation Issues in Extensible IP Router Software.p-3

La siguiente parte esta enfocado a como se va a realizar la comunicación entre otro dispositivo bgp, allí el nodo *peer* presenta el valor de la dirección ip a la cual se van a intercambiar los mensaje bgp, seguido en este bloque incorpora el *local-ip* que establece la dirección IP local con la cual se va a realizar la conexión TCP. Luego viene el AS, al cual se va a recibir los mensajes, sino coincide este parámetro, generara un error, enviado un mensaje NOTIFICATION ver figura 78. Para *next-hop* establece desde el punto del enrutador que se configura, cual debería ser el siguiente salto para el enrutador del cual recibe mensajes.

3.2.3.3 Prueba re-distribución de enrutadores en BGP.

Para hacer uso de redistribución de enrutadores sobre un protocolo de enrutamiento, hay que implementar declaración de políticas. Para esta prueba se utilizaron las opciones "static" y "connected" como los elementos de enrutamiento de los cuales un dispositivo bgp deberá aprender de las rutas ofrecidas por "static" y "connected".
Para la prueba se trabajo en tres partes, la primera es la implementación de bgp utilizando la opción de *protocol:connected*, la segunda la implementación de *protocolo: static* en bgp y la ultima elección es la de enrutadores *connected*, pero además utilizando una regla, que será definida en el bloque *term* en el archivo de configuración.

Para la utilización de *connected* como protocolo, para que bgp aprenda de sus enrutadores se presenta la siguiente figura, describiendo las rutas de las cuales van a escuchar y aceptar.

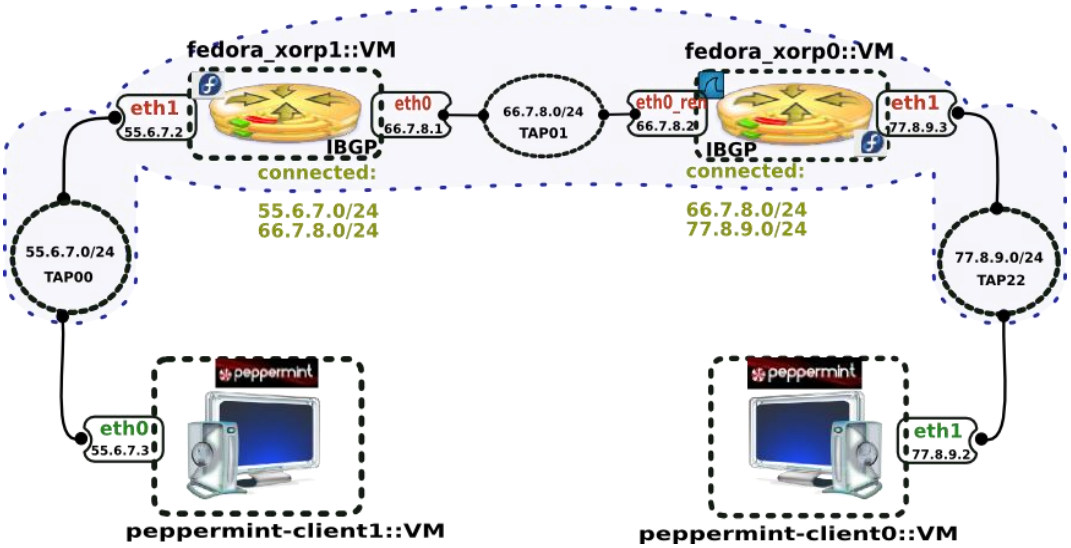
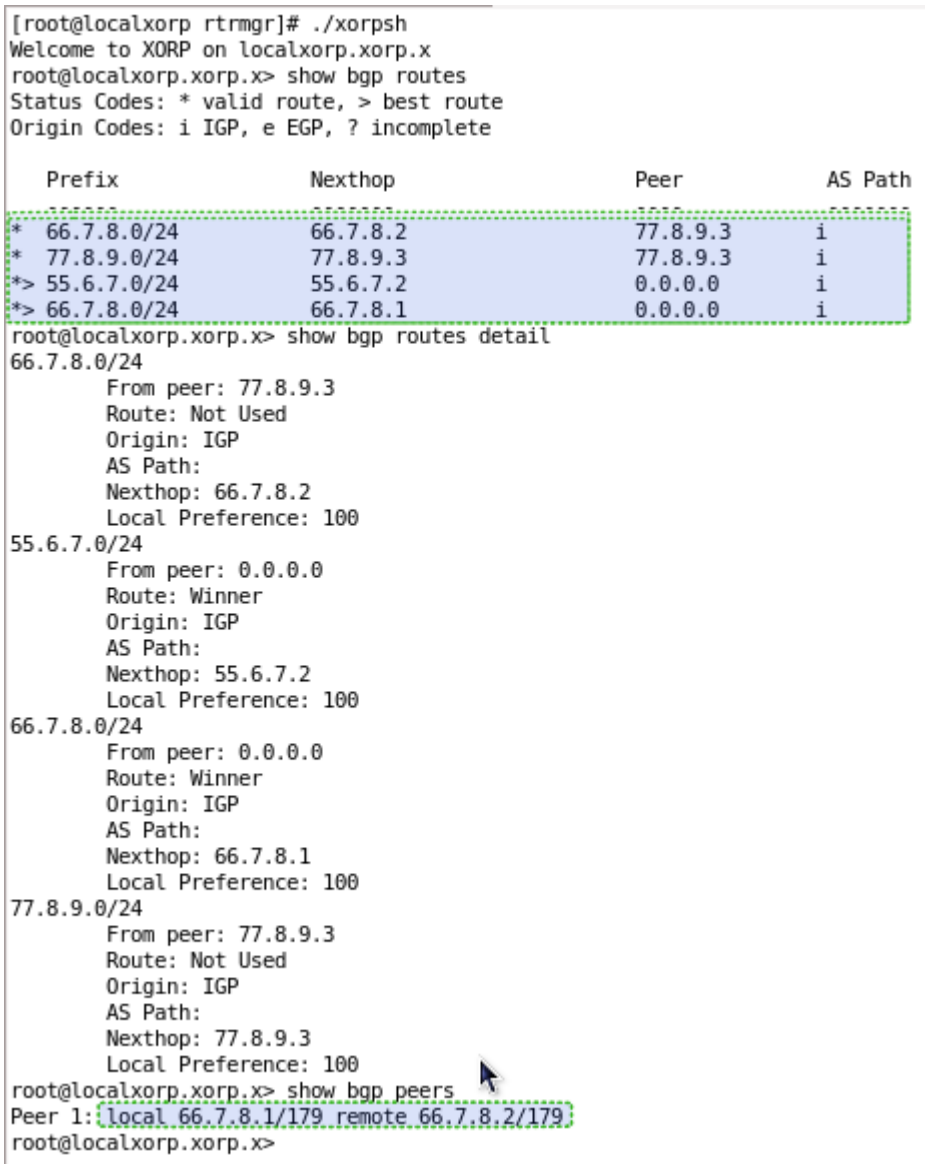


figura 70 Protocol:connected para IBGP fuente: Autor

En este punto de la configuración presentada en el siguiente párrafo:

```
policy {
   policy-statement to-bgp {
  term include {
      from  {
        protocol: "connected"
       bgp {
           export: "to-bgp"
```

Expone a BGP aprender de los enrutadores propuestos por *connected*. Para “fedora_xorp1” los enrutadores que propone connected son 55.6.7.0/24 , 66.7.8.0/24 que son las redes a las cuales pertenece este mismo enrutador. Para “fedora_xorp0” son 66.7.8.0/24 y 77.8.9.0/24.



```
[root@localxorp rtrmgr]# ./xorpsh
Welcome to XORP on localxorp.xorp.x
root@localxorp.xorp.x> show bgp routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

   Prefix               Nexthop                 Peer            AS Path
   ------               -------                 ----            -------
*  66.7.8.0/24          66.7.8.2                77.8.9.3        i
*  77.8.9.0/24          77.8.9.3                77.8.9.3        i
*> 55.6.7.0/24          55.6.7.2                0.0.0.0         i
*> 66.7.8.0/24          66.7.8.1                0.0.0.0         i
root@localxorp.xorp.x> show bgp routes detail
66.7.8.0/24
        From peer: 77.8.9.3
        Route: Not Used
        Origin: IGP
        AS Path:
        Nexthop: 66.7.8.2
        Local Preference: 100
55.6.7.0/24
        From peer: 0.0.0.0
        Route: Winner
        Origin: IGP
        AS Path:
        Nexthop: 55.6.7.2
        Local Preference: 100
66.7.8.0/24
        From peer: 0.0.0.0
        Route: Winner
        Origin: IGP
        AS Path:
        Nexthop: 66.7.8.1
        Local Preference: 100
77.8.9.0/24
        From peer: 77.8.9.3
        Route: Not Used
        Origin: IGP
        AS Path:
        Nexthop: 77.8.9.3
        Local Preference: 100
root@localxorp.xorp.x> show bgp peers
Peer 1: local 66.7.8.1/179 remote 66.7.8.2/179
root@localxorp.xorp.x>
```

figura 71 Lista de enrutadores y saltos de "fedora_xorp1" para protocolo:connected
fuente: Autor

A continuación se presentarán la lista de enrutadores de BGP en XORP, utilizando *protocol:connected*. Para “fedora_xorp1” se tiene la figura 72.

```
Welcome to XORP on localxorp.xorp.x
root@localxorp.xorp.x> show bgp routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

   Prefix               Nexthop                 Peer            AS Path
   ------               -------                 ----            -------
*  55.6.7.0/24          55.6.7.2                55.6.7.2        i
*  66.7.8.0/24          66.7.8.1                55.6.7.2        i
*> 66.7.8.0/24          66.7.8.2                0.0.0.0         i
*> 77.8.9.0/24          77.8.9.3                0.0.0.0         i
root@localxorp.xorp.x> show route table ipv4 unicast ibgp
root@localxorp.xorp.x> show route table ipv4 unicast static
Request for routes to be redistributed from static failed.
The protocol is probably not active.

root@localxorp.xorp.x> show route table ipv4 unicast connected
66.7.8.0/24     [connected(0)/0]
                > via eth0_rename/eth0_rename
77.8.9.0/24     [connected(0)/0]
                > via eth1/eth1
root@localxorp.xorp.x> show bgp routes detail
66.7.8.0/24
        From peer: 0.0.0.0
        Route: Winner
        Origin: IGP
        AS Path:
        Nexthop: 66.7.8.2
        Local Preference: 100
77.8.9.0/24
        From peer: 0.0.0.0
        Route: Winner
        Origin: IGP
        AS Path:
        Nexthop: 77.8.9.3
        Local Preference: 100
55.6.7.0/24
        From peer: 55.6.7.2
        Route: Not Used
        Origin: IGP
        AS Path:
        Nexthop: 55.6.7.2
        Local Preference: 100
66.7.8.0/24
        From peer: 55.6.7.2
        Route: Not Used
        Origin: IGP
        AS Path:
        Nexthop: 66.7.8.1
        Local Preference: 100
root@localxorp.xorp.x>
```

figura 72 Lista de enrutadores y saltos de "fedora_xorp0" para protocolo:connected
fuente: Autor

Para la lista de enrutadores de bgp en “fedora_xorp1”, que esta referenciado en azul, también están presentes los enviados o propagado por bgp por la maquina virtual “fedora_xorp0”.
Con el comando “show bgp peers” se establece que la procedencia de esta tabla

es de “fedora_xorp1” ya que el *peer-local* esta en la interface de red eth0, “fedora_xorp1”.

La figura 72 incorpora otra información útil referente a que enrutadores le son inyectados a BGP, con el comando, “show route table {ipv4 ipv6} {unicast multicast} {protocolo}” se puede establecer la tabla de enrutamiento final. También se puede ver que el protocolo *static* no esta presente en la configuración y la lista de enrutadores de BGP de “fedora_xorp0” es bastante parecida a “fedora_xorp1”, ya que los enrutadores BGP propagaron al *protocolo:connected.*

Para la segunda parte se trabajo con el “protocolo: static”. El archivo de configuración queda de la siguiente forma en el “fedora_xorp1” :

```
policy {
   policy-statement to-bgp {
 term include {
     from  {
      protocol: "static"
                .....
     static {
      route 77.8.9.0/24 {
       next-hop: 66.7.8.2
                 }
             }
      bgp {
          export: "to-bgp"
```

En el código de configuración de arriba se declara una política en BGP llamada *to-bgp,* y la ejecución de protocolos de enrutamiento como static y bgp. En la política de bgp se le dijo a bgp que la ruta estática 77.8.9.0/24 fuera tenida en cuenta para propagaciòn en la tabla de enrutamiento bgp. Esto se pudo constatar con los resultados presentados en la figuras 74 y 75. (Siguiente pagina)

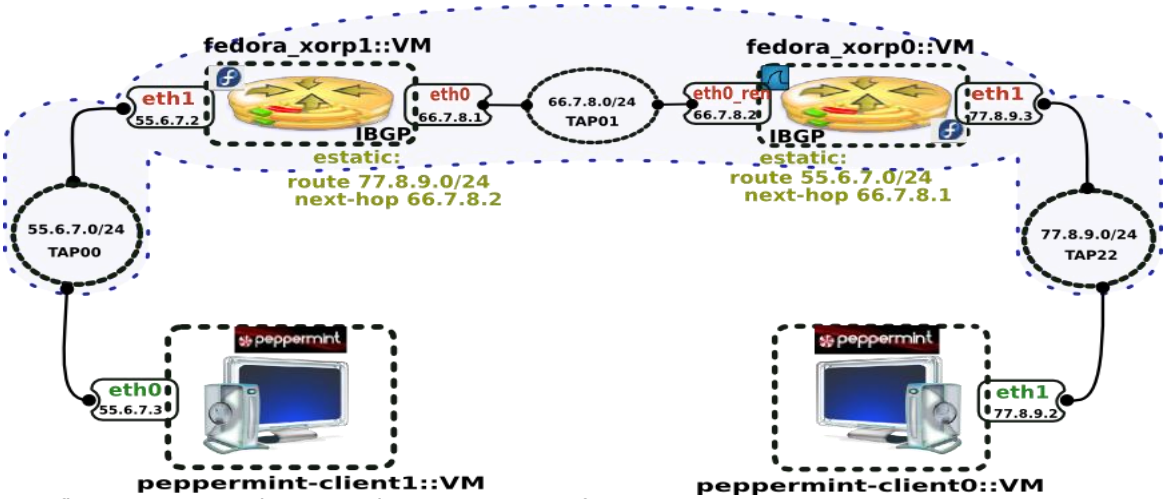


figura 73 Protocol:connected para IBGP *fuente: Autor*

En el dispositivo enrutador bgp llamado “fedora_xorp1”, se introduce una nueva ruta, por medio del protocolo *static*, este protocolo también esta disponible en “fedora_xorp0”, como se visualiza en la figura 73.

Esta configuración expone al dispositivo enrutador BGP aprender del enrutador propuesto por el protocolo de enrutamiento estático que fue fijado manualmente en el archivo de configuración.

A continuación se presentarán screenshot del terminal que muestran los resultados de propagar el enrutador estático por medio de BGP en la mini red detallada de la figura 73.
Para la MV “fedora_xorp1”, se tiene (siguiente Página):

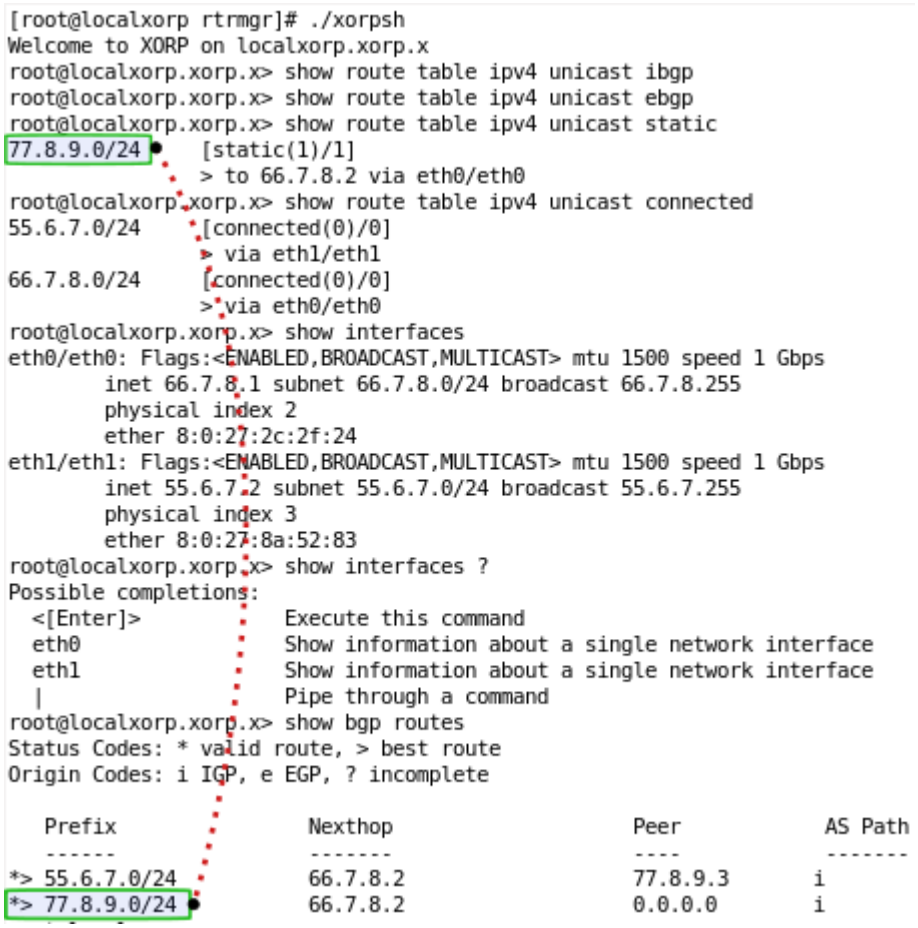


Figura 74 lista de los enrutadores de IBGP con el protocol:"static" fuente: Autor

En la figura 74, se ve como para la tabla de enrutamiento estático, toma el enrutador 77.8.9.0/24 y lo coloca en la tabla de enrutadores de BGP, también se observa, el enrutador 55.6.7.0/24 proveniente del dispositivo BGP de “fedora _xorp0”, propagado por este.

Para la figura anterior vale comentar el comando *show interfaces*, que interviene, para recalcar la ubicación de terminal del cual se obtuvo esa información, el terminal es perteneciente a la MV “fedora_xorp1” al cual le pertenecen las interfaces de red eth0 66.7.8.1 y eth1 55.6.7.2

El *protocolo: “connected”* aunque vigente no esta disponible en la configuración para ser utilizado.

A continuación se mostrara en la figura 75 (siguiente página), la parte correspondiente a la MV “fedora_xorp0”.

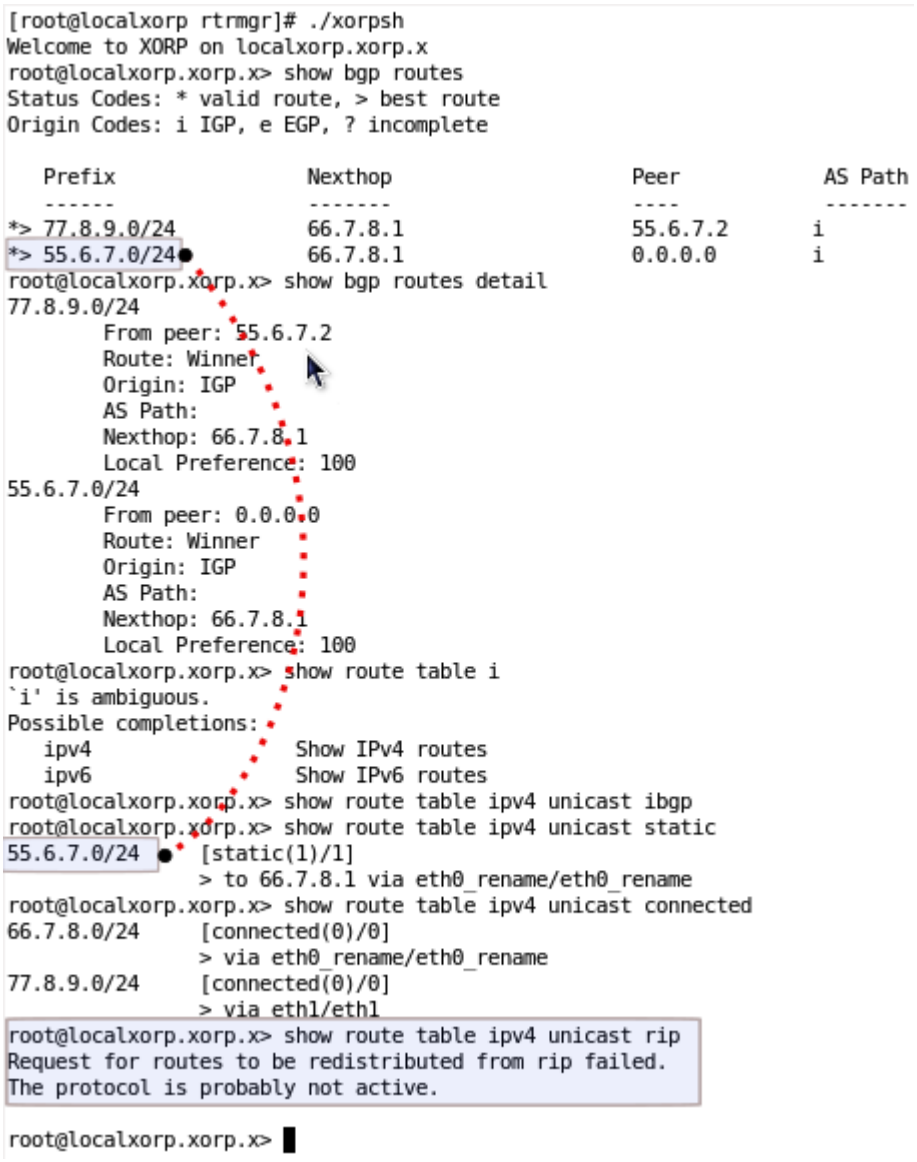


Figura 75 lista de los enrutadores de IBGP con el protocol:"static" fuente: Autor

En la figura 75, muestra como el enrutador configurado para el protocolo de enrutamiento estático es distribuido en BGP y propagado también se ve en la tabla de BGP al enrutador 77.8.9.0/24, propagado desde el dispositivo BGP “fedora_xorp0”.

Para la tercera parte, en la utilización en la declaración de políticas en BGP, se incluye una regla en la redistribución del enrutador. En la prueba sólo se utilizó la parte de la mini red de la MV “”fedora_xorp1” y con el *protocol:connected*, el objetivo es que cambiando la regla sólo se tenga encuentra uno o ninguno de los dos enrutadores propuestos por *connected (55.6.7.0/24, 66.7.8.0/24)*, para ser

incluido en el BGP.
A continuación se presentara la figura concerniente a la parte tres de las pruebas de XORP en BGP con la declaración de políticas.
Para redistribuir un enrutador en particular se utilizó a *connected* y se hizo uso de los demás bloques pertenecientes a term como son *to* y *then.* Ver figura 76

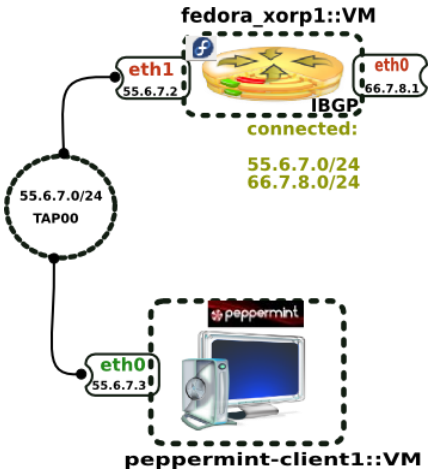


Figura 76. Diagrama parte tres redistribuir un enrutador fuente :autor

Se tomaron screenshot de los resultados al ir cambiando la regla en el archivo de configuración bgpZ.boot y observando los resultados de cuales enrutadores propago. El objetivo es coincidir los enrutadores con la redes 55.6.7.0/24 y 66.7.8.0/24. Estos son los resultados:

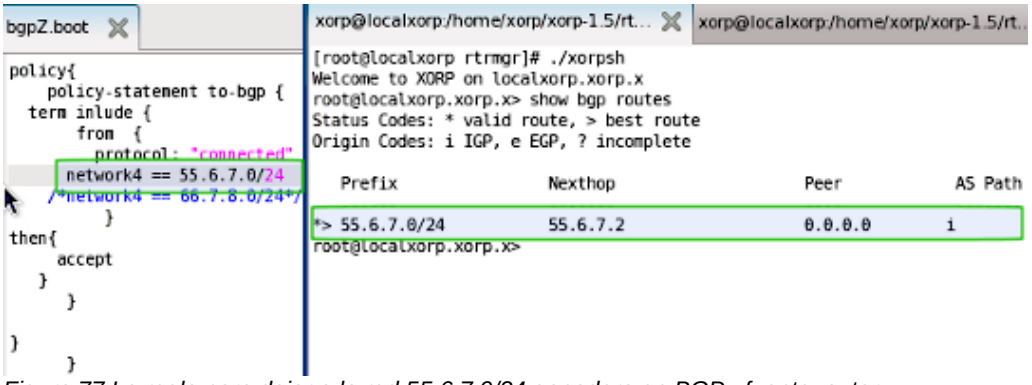


Figura 77 La regla para dejar a la red 55.6.7.0/24 ganadora en BGP fuente: autor

En la figura 77 la regla dice (en la parte izquierda de la figura) que del protocolo connected, de la red 55.6.7.0/24 lo acepte, los demás que no están cobijados por esta regla serán descartados. Como se vio en el resultado de la ejecución del código de la parte derecha se tiene que el enrutador bgp en su tabla de envío tuvo en cuenta a la ruta 55.6.7.0/ 24.

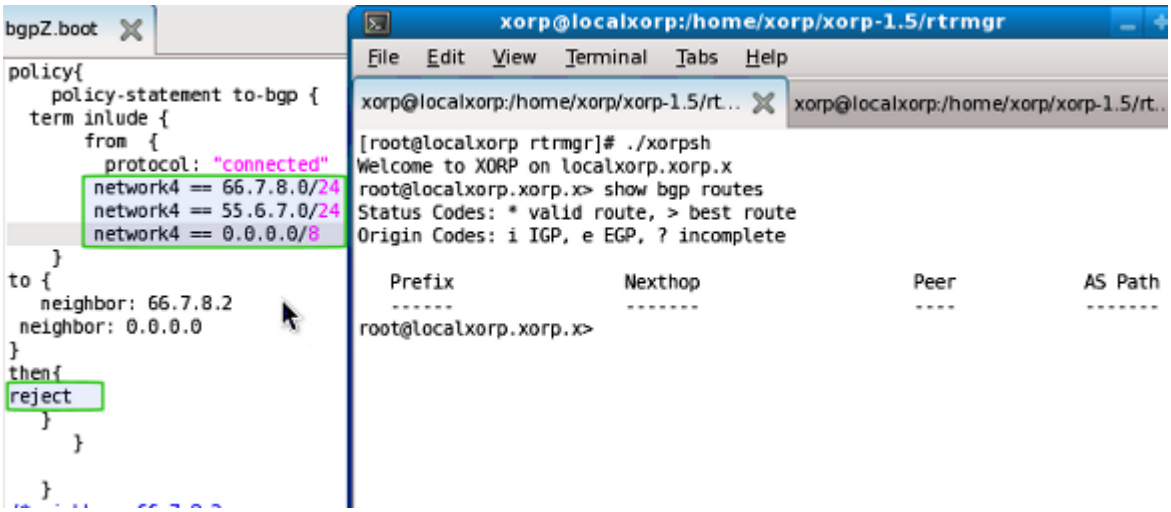


Figura 79 la regla fijada descarta las dos redes propuestas por connected fuente: Autor

Para la figura 79 se utiliza la acción *reject* , descartando las redes propuestas por *connected.*

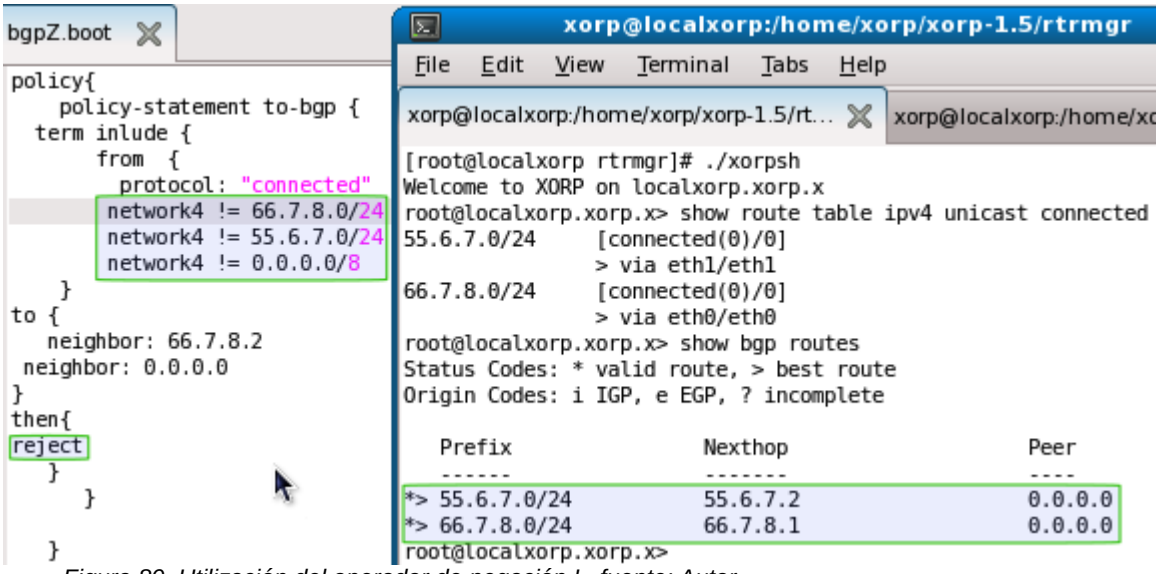


Figura 80 Utilización del operador de negación ! fuente: Autor

Para la figura 80, se utiliza el operador negador. La regla dice que a excepción de las redes 66.7.8.0/24, 77.8.9.0/24 y 0.0.0.0/0, serán descartadas.

Estos operadores están disponibles en la tabla de coincidencias generales[1], del

1 XORP.Inc.2009.User-manual.dvi. p-124-127

manual de usuario.
Además de las acciones *accept, reject, next-policy,* y de las reglas generales de coincidencia, también hay reglas particulares y acciones particulares para cada protocolo. Para BGP por ejemplo, el tipo de acciones después de que la regla coincida serían, modificación de prefijo de distancia, el campo de *community*, la *med*, para la regla de coincidencia puede utilizar los atributos de *as-path, neighbor, med* entre otros.

3.2.4 Prueba de OSPF en XORP y creación de VLANs

El *Internet Engineering Task Force* (IETF) reconoció que RIP por el mismo no podía hacerse cargo de las necesidades de todos los sistemas autónomos en internet,[1] formando un equipo en 1988, para desarrollar un nuevo protocolo de enrutamiento basado en el algoritmo *link-state*, también llamado *shortest path first.*
El nuevo protocolo fue llamado *Open Shortest Path First, o OSPF,* El cual describe en su nombre dos características, una que fue desarrollado como un estándar libre por el proceso de rfc, no se necesita licenciamiento para utilizarlo, la otra característica viene del tipo de algoritmo que utiliza, diseñado para que el enrutador determine dinámicamente el camino mas corto entre dos redes.

En tanto a la operación de OSPF, se basa en el concepto fundamental de *link-state database* (LSDB), cada enlace a una red o a otro enrutador es representado por una entrada en una base de datos, cada estructura de datos tiene asociado un costo (*metric*). La métrica puede estar hecha para incluir diferentes aspectos en desempeño del enrutador no solamente por el número de saltos como el caso de RIP.

Información acerca del Sistema Autónomo se mueve alrededor de todo el SA, en forma de *link-state advertisements* (LSAs), mensajes dejados de cada enrutador, comentan el estado de todo el Sistema Autónomo. En un tiempo, estos mensajes convergen y son pasados a todo el SA, permitiendo que los enrutadores posean el mismo tipo de datos. Cuando ocurren cambios dentro del SA, enrutadores involucrados en ese cambio envían mensajes tipo UPDATE, asegurándose que el SA este actualizado.

Algunas características de OSPF, están en autentificación, soporte para grandes redes, también permite enrutadores ser clasificados en jerarquías, permitiendo mejor organización y desempeño al reducir el trafico de mensajes de propagación *link-state.*
También OSPF permite trabajar en cinco tipos de redes,[2] redes punto a punto, redes *Broadcast,* redes *Non–broadcast multiaccess,* redes *Point-to-multipoint*

---

1 .Charles M. Kozierok.The TCP/IP Guie.P-734.
2 Network Routing Algorithms, Protocols, and Architectures, 2007.p-169. autor: Deepankar Medhi, Karthikeyan Ramasamy.

networks y *Virtual links.*

La clasificación de enrutadores según tareas y ubicación en aéreas especificas de operación, involucra reseñar y configurar un enrutador de características singulares, esta complejidad de OSPF, podría disponer de dos estilos de configuración, una para redes pequeñas y poco escalable y otra para redes grandes.
Para redes pequeñas mantener los mensajes *link-state* es fundamental, pero cuando la red presenta una estabilidad mayor estos mensajes pudieran afectar el desempeño de la red, por eso es recomendable fracturar la red en AREAS, cada área contiene una serie de continuos enrutadores y redes, estas AREAS son numeradas y manejadas independientemente por los enrutadores pertenecientes a estas aéreas, es como si cada área fuera un mini sistema Autónomo, aun así, estas aéreas están interconectadas, entonces la información de enrutamiento puede ser compartida entre aéreas y atreves de todo el Sistema Autónomo.

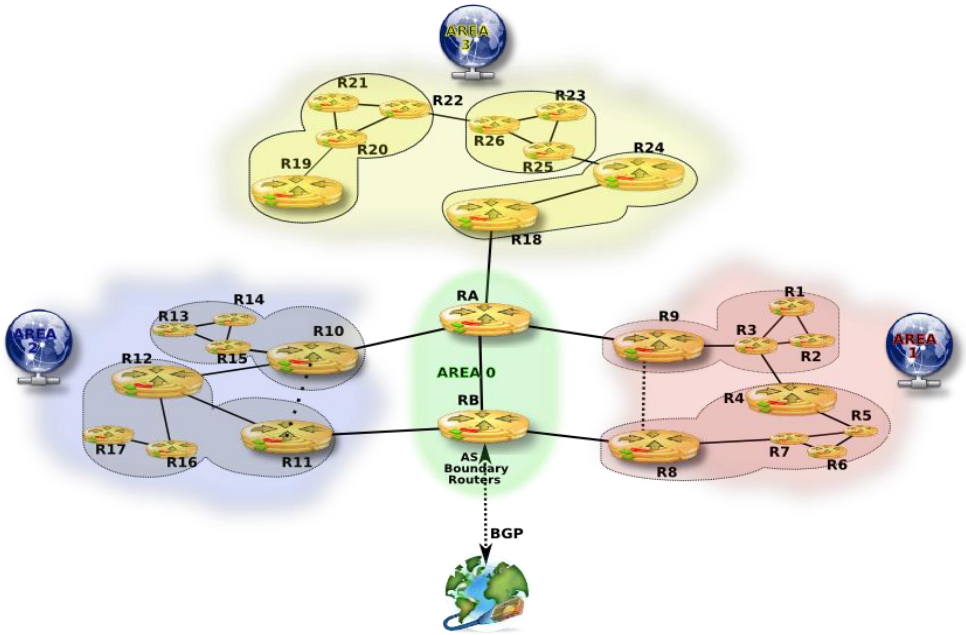


Figura 81 Fragmentación de OSPf en areas *fuente:autor*

Los papeles que desempeñan enrutadores en cada área se clasifican de forma jerárquica, se tienen los siguientes:
enrutadores Internos o enrutadores de bajo nivel: Estos enrutadores están conectados con otros enrutadores o otras redes, dentro de la misma área, estos enrutadores mantienen la LSDB, de sólo la AREA a que pertenecen[1], para la figura 81 enrutadores de este tipo en el área 3 son, R19, R21, R20, R22 también para la

1 .Charles M. Kozierok.The TCP/IP Guie.2005.P-739.

siguiente red están, R26, R23, y R25.

Enrutadores *Backbone:* Estos enrutadores están ubicados en la AREA 0.0.0.0. Son enrutadores de este tipo RA y RB.
enrutadores de AREA en los bordes: Estos enrutadores están ubicados en los bordes entre los enrutadores *Backbone* y los enrutadores internos. Cada enrutador de borde debe tener al menos un dispositivo de red en un Backbone y otro al AREA al cual conecta. Enrutadores en el AREA 2 serían R10 y R11.
Enrutadores *AS Boundary:* [2]Estos enrutadores están localizados en el AREA 0 con conectividad hacia otros AS; estos deben manejar más de un protocolo de enrutamiento como BGP, las funciones están en intercambiar información con otros AS, también deben tener interfaces de red que conectan a enrutadores *Backbone.* El enrutador RB es un ejemplo de esto.

Debido a la diversidad de conceptos que toma OSPF, para realizar enrutamiento en redes de gran escalabilidad, también sus mensajes representan cierto grado de complejidad al depender de donde y cuando se originan; por ello los *Link State Advertisement* presentan al menos cinco tipos,[1] *Router LSA (type code = 1), Network LSA (type code = 2), Network Summary LSA (typecode = 3), AS Border Router (ASBR) Summary LSA (type code = 4), and AS External LSA(typecode=5).*
un *Router LSA* es el más básico *link-state advertisement,* que es generado por cada interface de red, tal mensaje es generado en redes punto a punto, este enrutador es almacenado en un LSD y son usados con módulos de enrutamiento computacional.
Un *Network LSAs* es aplicable a redes de multi acceso, el alcance de estos mensajes esta sujeto a sólo el área donde se origino el mensaje.
Un *Área-Border Routers* genera un *Network Summary LSAs* que son usados para propagar destinos para otras AREAS; es decir que el *Network Summary LSAs* permite propagar IP-prefijos entre aéreas.

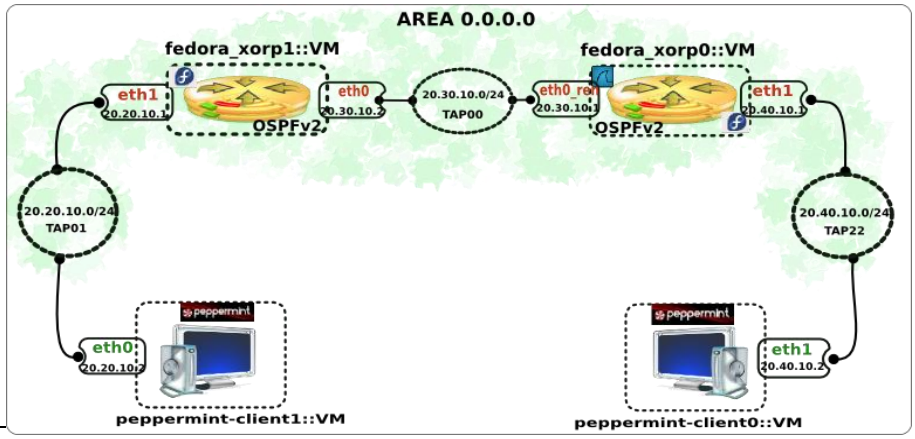


Figura 82 Mini red para ejecutar OSPF en XORP fuente: Autor

2 N[illegible]pankar Medhi, Karthikeyan Ramasamy.

1 Network Routing Algorithms, Protocols, and Architectures, 2007.p-171. autor: Deepankar Medhi, Karthikeyan Ramasamy.

OSPF permite clasificar los mensajes en medios de propagación y alcance de área, pero el formato de cada mensaje esta supeditado al campo destinado al tipo de mensaje en OSPF, el nombre de este campo es M*essage type,* a continuación se presentarán los mensajes capturados con el wireshark en la interface de red eth0_rename, que corresponde al diagrama de la figura 82, el archivo de configuración ospf.boot, disponible en el anexo G. En la siguiente sección se podrán observar los mensajes entre dos entidades OSPF de AREA 0.
Los mensajes OSPF tienen unos campos comunes por mensaje perteneciente a OSPF, estos campos son conocidos como *common Headers*, estos son:[1]

Versión: Este campo especifica la versión de OSPF, la actual versión es 2.

Type: Este campo especifica el paquete a que tipo de mensaje pertenece. OSPF tiene cinco tipos de paquetes: *hello* (1), *database description* (2), *link state request* (3), *link state Update*(4), y *LSA* (5).

Packet Length: Este campo indica el tamaño del paquete OSPF.

Router ID: Este campo indica del ID del enrutador originador del mensaje.

Área ID: Este ID es el área de donde el mensaje OSPF se origino. El valor de 0.0.0.0 es reservado para el *backbone área*.

Checksum: Este es el IP *checksum* de todo el paquete OSPF.

AuType y *Authentication Field*: Este campo especifica el tipo de autentificación.

Los campos anteriormente referenciados están presentes en el figura 83, en el campo *OSPF Header.*

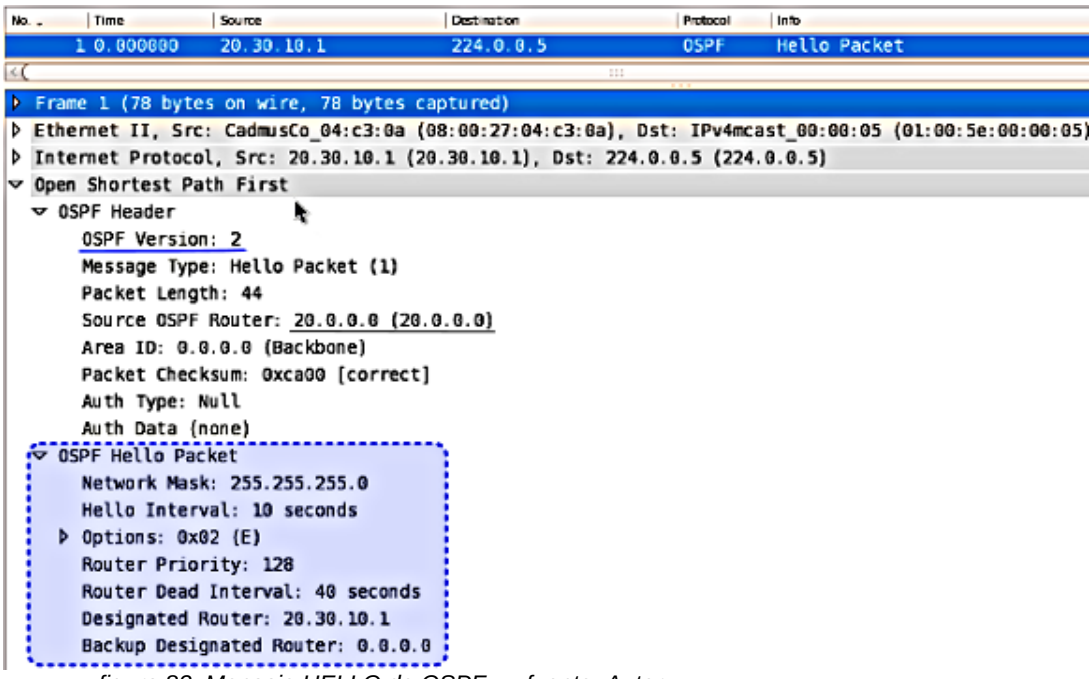


figura 83 Mensaje HELLO de OSPF fuente: Autor

Para la figura 83 el mensaje HELLO, referenciado en azul, involucra los campos[1] siguientes:

Network Mask: Esta es la mascara de dirección del enrutador en la interface de red de la cual se envió el mensaje.

Hello Interval: En este campo es designado como el tiempo de diferencia en segundos entre mensajes tipo *Hello.* Para redes punto a punto es tiempo por defecto es de 10 seg.

Options: Las opciones del campo permiten compatibilidad con enrutadores vecinos para ser verificados.

Priority:[2] Indica la prioridad el enrutador cuando es elegido para que funcione como un enrutador de respaldo.

Router Dead Interval: Este es el tiempo que espera un enrutador OSPF, para recibir un mensaje HELLO del dispositivo OSPF vecino.

Designated Router (DR) (Backup Designated Router (BDR)): Es la dirección de un enrutador designado para funciones especiales en la misma red.

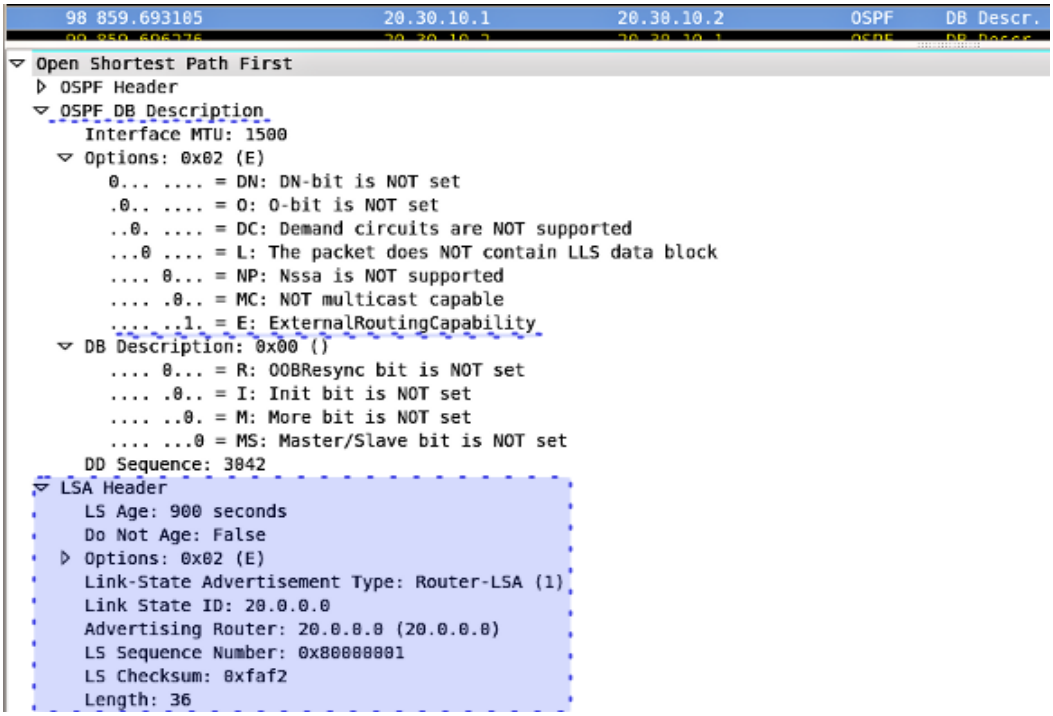


Figura 84 Mensaje DB DESCRIPTION *fuente:autor*

El mensaje DB DESCRIPTION además del campo llamado *OSPF header* tiene los siguientes campos:[1]

---

1 Network Routing Algorithms, Protocols, and Architectures, 2007.p-178. autor: Deepankar Medhi, Karthikeyan Ramasamy.

2 Charles M. Kozierok.The TCP/IP Guie.2005.P-749.

1 Network Routing Algorithms, Protocols, and Architectures, 2007.p-179. autor: Deepankar Medhi, Karthikeyan Ramasamy.

Interface Maximum Transmission Unit (MTU): Este campo indica el mayor tamaño en la unidad de transmisión de un paquete puede tener, sin tener que fragmentarlo.

Options: Este campo contiene una serie de campos en bits, describiendo las capacidades que el enrutador soporta. Donde el bit mas importante es “E”, que describe si el enrutador en donde habita el área, puede soportar el procesar un External LSAs.

I/M/MS bits: Estos bits de banderas son utilizadas para indicar información en el intercambio de mensajes tipo DB DESCRIPTION; *I-bit (initial-bit)* indica que este es el primer mensaje de una secuencia próxima de mensajes tipo *db description. M-bit (more-bit)* es usado para indicar que este no es el último paquete para la sesión de *db description,* para el último paquete esta bandera es colocada en cero. *MS-bit (master-slavebit)* es usado para indicar, cual de los enrutadores origino el mensaje, clasificándolos como *master* o *slave.*

DD Sequence number: Este campo es usado para tener la secuencia de número de paquetes enviados a la *DB message.*

LSA Header: Contiene información de las cabeceras de *link-state advertisement* las cuales llevan información acerca de la LSDB

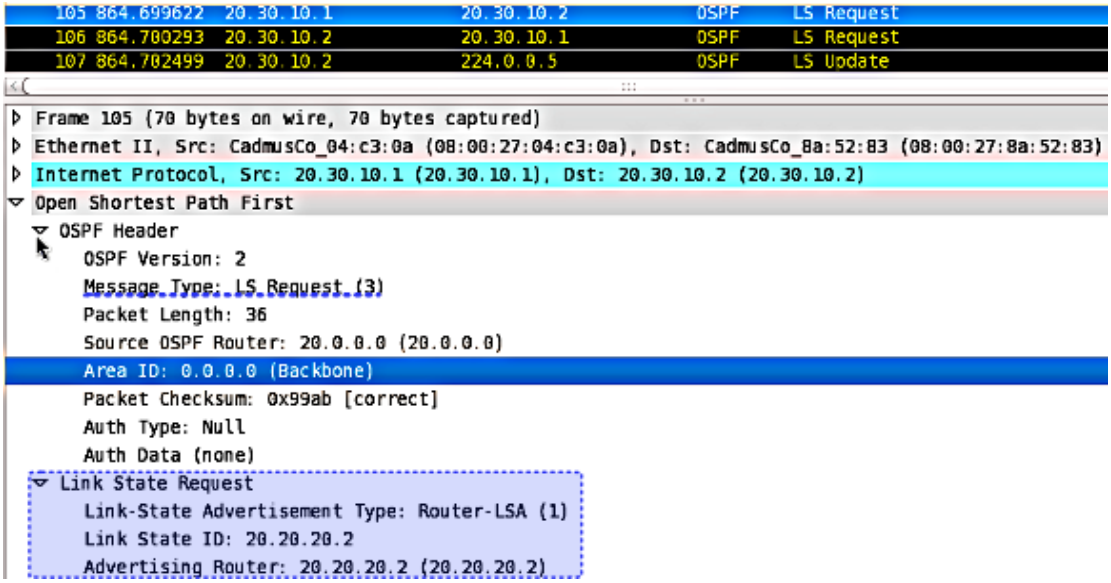


Figura 85 Mensaje LS Request en OSPF *fuente: Autor*

El mensaje de la figura 85, es el tipo número 3 en la cabecera de OSPF, que corresponde a *LS Request*, tiene como objetivo extraer información de un dispositivo OSPF, [1]ejerciendo una operación *pull,* esto ocurre cuando uno de los enrutadores se muestra interesado, cuando ya ha intercambiado información de tipo DBLS, este enrutador interesado pudiera establecer comunicación para

1 Network Routing Algorithms, Protocols, and Architectures, 2007. autor: Deepankar Medhi, Karthikeyan Ramasamy. p-180.

información de tipo Link State. Para este caso el tipo de enlace es Router-LSA

Link State Type: Este campo identifica el tipo de mensaje *link state.*
Link State ID: Es usualmente un valor tipo IP address, ya sea del enrutador o de la red a la cual se enlazo el enrutador interesado.
Advertising Router: Es el ID de el enrutador que creo la LSA de quien el UPDATE esta siendo propagado.
Finalizado el mensaje tipo número 3 de la captura, seguido se presentara el mensaje tipo 4 el *LS UPDATE.*

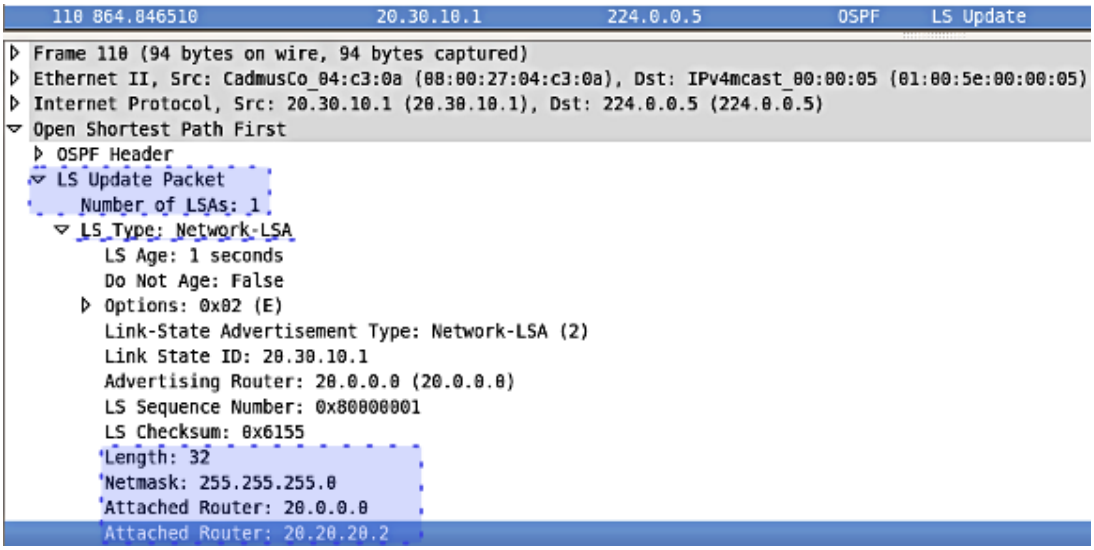


figura 86 Mensajes LS Update con LS type:Network LSA Fuente : Autor

Las figuras 86 y 87 tienen en común que pertenecen al mensaje tipo 4 llamado

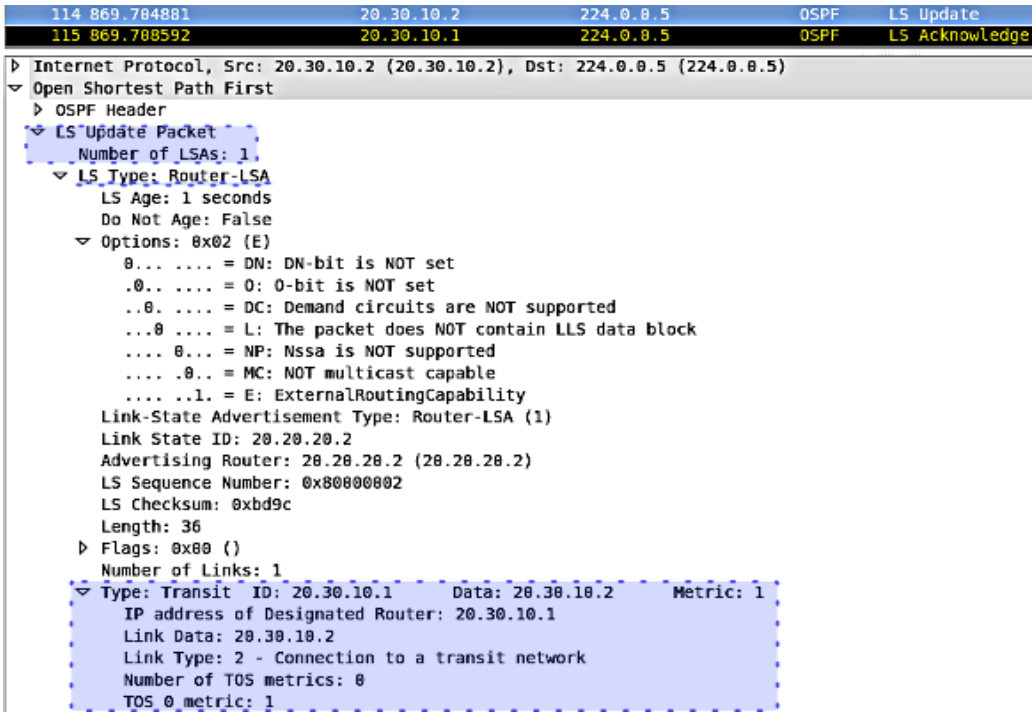


figura 87 Mensaje LS Update con LS Type: Router-LSA fuente: Autor

LS UPDATE, pero se diferencian del tipo de mensaje de propagación el LS Type.

Number of LSAs: Contiene el número de mensajes *LS Type* que lleva el UPDATE.

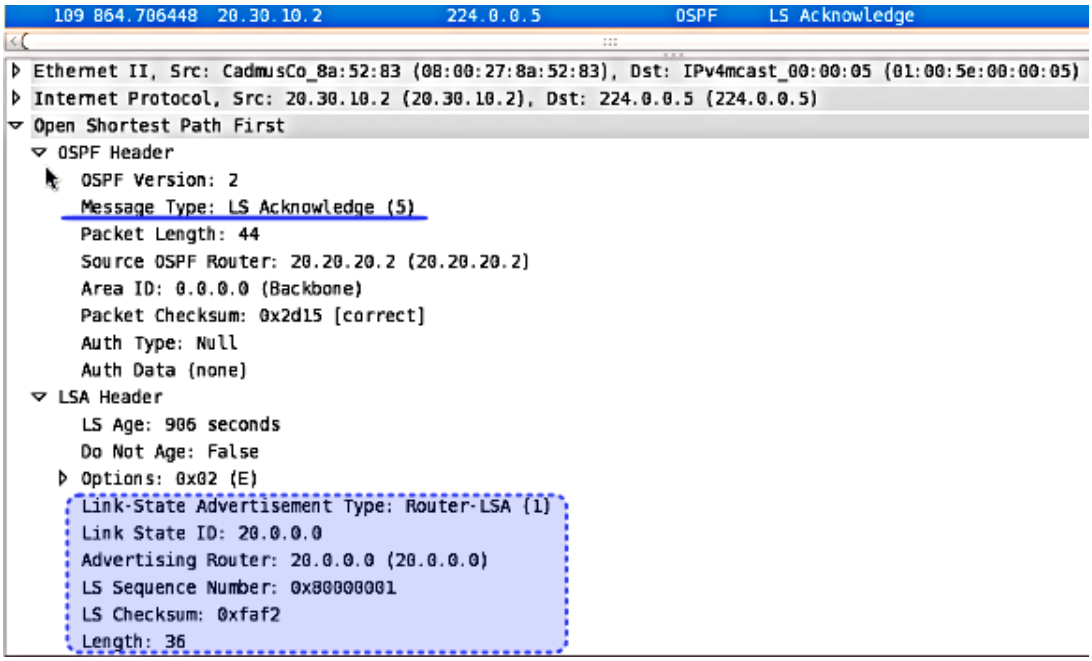


Figura 88 Mensaje LS ACKNOWLEDGE fuente: Autor

Este tipo de *Link State Advertisement,* contiene cabeceras LSA, para generar un mensaje tipo *Acknowledg.* Ver figura 88.

A continuación se presentarán screenshot de la línea de comando de XORP para cada una de las maquinas virtuales, haciendo énfasis en comandos para el protocolo de enrutamiento OSPF.

La siguiente imagen presenta las opciones para OSPF en la línea de comando para requerir información de la base de datos.

```
[root@localxorp rtrmgr]# ./xorpsh
Welcome to XORP on localxorp.xorp.x
root@localxorp.xorp.x> show ospf4 database ?
Possible completions:
  <[Enter]>           Execute this command
  area                -- No help available --
  asbrsummary         Show Summary-LSA (AS boundary router) database
  brief               Display brief output (default)
  detail              Display detailed output
  external            Show External-LSA database
  netsummary          Show Summary-LSA (network) database
  network             Show Network-LSA database
  nssa                Show NSSA-LSA database
  router              Show Router-LSA database
  summary             Display summary output
  |                   Pipe through a command
root@localxorp.xorp.x> show ospf4 database
```

Figura 90 Información del comando "show ospf database" fuente: Autor

En la siguiente figura para “fedora_xorp1” se utiliza por primera vez comandos concernientes a adquirir información del protocolo OSPF configurado.

```
[root@localxorp rtrmgr]# ./xorpsh
Welcome to XORP on localxorp.xorp.x
root@localxorp.xorp.x> show ospf4 database
   OSPF link state database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router  *20.20.20.2       20.20.20.2      0x80000002  1460  0x2  0xc791  36
Router   20.0.0.0         20.0.0.0        0x80000002  1456  0x2  0xdad1  36
ASExt-2  20.30.10.0       20.0.0.0        0x80000001  1470  0x2  0x281d  36
Network *20.30.10.2       20.20.20.2      0x80000001  1460  0x2  0x7516  32
ASExt-2  20.40.10.0       20.0.0.0        0x80000001  1470  0x2  0xd89f  36
root@localxorp.xorp.x> show ospf4 database external
   OSPF link state database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
ASExt-2  20.30.10.0       20.0.0.0        0x80000001  1482  0x2  0x281d  36
ASExt-2  20.40.10.0       20.0.0.0        0x80000001  1482  0x2  0xd89f  36
root@localxorp.xorp.x> show ospf4 database network
   OSPF link state database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Network *20.30.10.2       20.20.20.2      0x80000001  1500  0x2  0x7516  32
root@localxorp.xorp.x>
```

figura 91 Comandos de OSPF en xorpsh en "fedora_xorp1" fuente: Autor

En la figura 91 muestra los comandos referenciados en azul “show ospf4 database”, “show ospf4 database external” en verde y “show ospf4 database network” en rojo, para la MV “fedora_xorp1”, muestran respectivamente en azul, la base de datos de las redes del enrutador vecino y de la red a la cual esta conectado a este, especificando información el ID del enrutador del cual recibe y propaga mensajes, el siguiente en verde, la base de datos aportada por el enlace externo, seguido en rojo muestra la red a la cual esta conectado en el enrutador y además el ID del enrutador propagador de mensajes de la red.

```
[root@localxorp rtrmgr]# ./xorpsh
Welcome to XORP on localxorp.xorp.x
root@localxorp.xorp.x> show ospf4 database
   OSPF link state database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router  *20.0.0.0         20.0.0.0        0x80000002   816  0x2  0xdad1  36
ASExt-2 *20.30.10.0       20.0.0.0        0x80000001   826  0x2  0x281d  36
ASExt-2 *20.40.10.0       20.0.0.0        0x80000001   826  0x2  0xd89f  36
Router   20.20.20.2       20.20.20.2      0x80000002   812  0x2  0xc791  36
Network  20.30.10.2       20.20.20.2      0x80000001   817  0x2  0x7516  32
root@localxorp.xorp.x> show ospf4 database external
   OSPF link state database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
ASExt-2 *20.30.10.0       20.0.0.0        0x80000001   909  0x2  0x281d  36
ASExt-2 *20.40.10.0       20.0.0.0        0x80000001   909  0x2  0xd89f  36
root@localxorp.xorp.x> show ospf4 database net
`net' is ambiguous.
Possible completions:
   netsummary           Show Summary-LSA (network) database
   network              Show Network-LSA database
root@localxorp.xorp.x> show ospf4 database network
   OSPF link state database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Network  20.30.10.2       20.20.20.2      0x80000001   958  0x2  0x7516  32
root@localxorp.xorp.x>
```

Figura 92 Comandos de OSPF en xorpsh en "fedora_xorp0" fuente: Autor

Para la figura 92 muestra la respuesta de los comandos “show ospf4 database external” referenciado en rojo y “show ospf4 database network” referenciado en verde.

```
root@localxorp.xorp.x> show ospf4 database detail
   OSPF link state database, Area 0.0.0.0
Router-LSA:
LS age 1237 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x1
Link State ID 20.0.0.0 Advertising Router 20.0.0.0 LS sequence numb
er 0x80000002 LS checksum 0xdad1 length 36
        bit Nt false
        bit V false
        bit E true
        bit B false
        Type 2 Transit network IP address of Designated router 20.3
0.10.2 Routers interface address 20.30.10.1 Metric 1
As-External-LSA:
LS age 1246 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x5
Link State ID 20.30.10.0 Advertising Router 20.0.0.0 LS sequence nu
mber 0x80000001 LS checksum 0x281d length 36
        Network Mask 0xffffff00
        bit E true
        Metric 0 0
        Forwarding address 20.30.10.1
        External Route Tag 0
As-External-LSA:
LS age 1246 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x5
Link State ID 20.40.10.0 Advertising Router 20.0.0.0 LS sequence nu
mber 0x80000001 LS checksum 0xd89f length 36
        Network Mask 0xffffff00
        bit E true
        Metric 0 0
        Forwarding address 0.0.0.0
        External Route Tag 0
Router-LSA:
LS age 1233 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x1
Link State ID 20.20.20.2 Advertising Router 20.20.20.2 LS sequence
number 0x80000002 LS checksum 0xc791 length 36
        bit Nt false
        bit V false
        bit E false
        bit B false
        Type 2 Transit network IP address of Designated router 20.3
0.10.2 Routers interface address 20.30.10.2 Metric 1
Network-LSA:
LS age 1238 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x2
Link State ID 20.30.10.2 Advertising Router 20.20.20.2 LS sequence
number 0x80000001 LS checksum 0x7516 length 32
        Network Mask 0xffffff00
        Attached Router 20.20.20.2
        Attached Router 20.0.0.0
root@localxorp.xorp.x>
```

figura 93 Comando show ospf database detail para "fedora_xorp1"
fuente: Autor

Para la figura 93 muestra el resultado del comando “show ospf4 database detail” que involucra a “fedora_xorp1”, en los detalles especifican el tipo del mensaje, el tipo de propagación y en el ultimo ítem se puede ver la lista de los LSD de los

enrutadores pertenecientes al área 0.0.0.0.

```
[root@localxorp rtrmgr]# ./xorpsh
Welcome to XORP on localxorp.xorp.x
root@localxorp.xorp.x> show ospf4 database detail
   OSPF link state database, Area 0.0.0.0
Router-LSA:
LS age 1691 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x1
Link State ID 20.20.20.2 Advertising Router 20.20.20.2 LS sequence
number 0x80000002 LS checksum 0xc791 length 36
        bit Nt false
        bit V false
        bit E false
        bit B false
        Type 2 Transit network IP address of Designated router 20.3
0.10.2 Routers interface address 20.30.10.2 Metric 1
Router-LSA:
LS age 1687 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x1
Link State ID 20.0.0.0 Advertising Router 20.0.0.0 LS sequence numb
er 0x80000002 LS checksum 0xdad1 length 36
        bit Nt false
        bit V false
        bit E true
        bit B false
        Type 2 Transit network IP address of Designated router 20.3
0.10.2 Routers interface address 20.30.10.1 Metric 1
As-External-LSA:
LS age 1701 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x5
Link State ID 20.30.10.0 Advertising Router 20.0.0.0 LS sequence nu
mber 0x80000001 LS checksum 0x281d length 36
        Network Mask 0xffffff00
        bit E true
        Metric 0 0
        Forwarding address 20.30.10.1
        External Route Tag 0
Network-LSA:
LS age 1691 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x2
Link State ID 20.30.10.2 Advertising Router 20.20.20.2 LS sequence
number 0x80000001 LS checksum 0x7516 length 32
        Network Mask 0xffffff00
        Attached Router 20.20.20.2
        Attached Router 20.0.0.0
As-External-LSA:
LS age 1701 Options  0x2 DC: 0 EA: 0 N/P: 0 MC: 0 E: 1 LS type 0x5
Link State ID 20.40.10.0 Advertising Router 20.0.0.0 LS sequence nu
mber 0x80000001 LS checksum 0xd89f length 36
        Network Mask 0xffffff00
        bit E true
        Metric 0 0
        Forwarding address 0.0.0.0
        External Route Tag 0
root@localxorp.xorp.x>
```

figura 94 Comando show ospf database detail para "fedora_xorp0"
fuente: Autor

Para la figura 94 como en la anterior figura expone los mensajes de propagación incorporando en los diferentes campos la información pertinente a cada tipo de mensaje OSPF, básicamente es la misma información que poseen los campos de

los mensajes tipo 5, 2, 1.
En la utilización de VLAN se presentara la configuración básica para crear una interface de red virtual que depende de otra interface de red, para el caso de la prueba dependerá de eth0_rename. Por definición en XORP las interfaces de red están divididas jerárquicamente en dos tipos, las reales (físicas), que son llamadas *interface* y las interfaces virtuales llamadas *vif.* La parte que define las direcciones IP corresponde a la parte de las *vif.* Si alguna interface de red física con un NIC puede soportar el manejo de Virtual LAN, estas son definidas en un segundo sud bloque debajo de la definición de *interface*. ver figura 95

ospfv2Y_rip.boot

```
1  /* $XORP$ */
2
3  interfaces {
4      interface eth0_rename {
5          vif eth0_rename {
6              address 20.30.10.1 {
7                  prefix-length: 24
8              }
9
10         }
11     vif vlan1 {
12         vlan {
13             vlan-id:01
14               }
15
16              address 20.30.10.4 {
17                      prefix-length: 24
18                      }
19               }
20         }
21
22     interface eth1 {
23         vif eth1 {
24             address 20.40.10.1 {
25                 prefix-length: 24
26             }
27         }
28     }
29     }
```

Figura 95 Configuración de una vlan en XORP fuente: Autor

En la siguiente figura se puede ver el resultado del comando “show interfaces”

```
[root@localxorp rtrmgr]# ./xorpsh
Welcome to XORP on localxorp.xorp.x
root@localxorp.xorp.x> show ?
Possible completions:
  host                Display information about the host
  interfaces          Show network interface information
  ospf4               Display information about OSPFv2
  route               Show routes
root@localxorp.xorp.x> show interfaces
eth0_rename/eth0_rename: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500 speed 1 Gbps
        inet 20.30.10.1 subnet 20.30.10.0/24 broadcast 20.30.10.255
        physical index 2
        ether 8:0:27:4:c3:a
eth0_rename/vlan1: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500 speed 1 Gbps
        inet 20.30.10.4 subnet 20.30.10.0/24 broadcast 20.30.10.255
        physical index 5
        ether 8:0:27:4:c3:a
        vlan: 1 parent interface: eth0_rename
eth1/eth1: Flags:<ENABLED,BROADCAST,MULTICAST> mtu 1500 speed 1 Gbps
        inet 20.40.10.1 subnet 20.40.10.0/24 broadcast 20.40.10.255
        physical index 3
        ether 8:0:27:18:9d:59
root@localxorp.xorp.x>
```

Figura 96 Comando "show interfaces" fuente: Autor

comprobando la existencia de la vlan disponible en el sistema anfitrión que ha sido creada desde el archivo .boot

3.2.5 Prueba de CLICK en un Fedora 12

CLICK no hace parte del código base en XORP-ct, en el tiempo del desarrollo de este trabajo, la opción de activar CLICK esta disponible en el código fuente de la versión 1.7-xorp.ct pero para versiones mas recientes no esta disponible.

Para ejecutar CLICK se debe recurrir a una opción de compilación ("enable_click=yes"), pero infortunadamente a pesar de la opción, después de compilado e instalado aun así presenta errores en la ejecución del Módulo; el desarrollador encargado del soporte no tiene intensiones en trabajar sobre CLICK. Las versiones que soportan CLICK son anteriores a 1.6 de XORP de la compañía ya desaparecida XORP.inc.

Lastimosamente al probar CLICK en una maquina Virtual "fedora-XORPx", presento errores sobre la comunicación tipo *socket* para ejecutar el binario ./click. La primer impresión es que el tiempo de espera para que se ejecute CLICK se vence, debido a que la comunicación de *sockets* en una maquina virtual aun en *localhost* es muy lenta, aprox. 1.5ms mas lenta que en un SO (Sistema Operativo) normal.
La opción para utilizar CLICK en la práctica con algún protocolo de enrutamiento es ejecutarlo desde afuera de XORP. El inconveniente que podría presentar la anterior opción en un ambiente de producción, vendría en el momento en que CLICK dejara de funcionar, XORP no daría cuenta de ello y las funciones de CLICK en el manejo de paquetes dejarían de existir, aunque con un script se puede superar el impase descrito anteriormente.

CLICK lo define Edi Kroner, en su Tesis Doctoral como un modelo de definición de elementos individuales (IPclasifiquer, Queue), en una unidad de procesar paquetes, con acciones pull o push (empujar o sacar paquetes), conectándolos de manera que se pasen paquetes uno a otro.

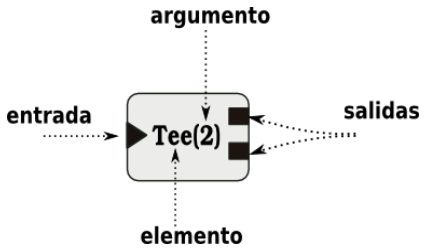


figura 97 Modelo de un elemento
fuente: Autor

Un modelo de un elemento básicamente consta una o varias entradas, como se puede ver en la figura 97, el elemento que tratara al paquete, un argumento requerido para que el elemento haga su tarea y las salidas.
La comunicación entre elementos se hacen mediante acciones *Push* y Pull, la comunicación es posible entre elementos sólo de tipo push o sólo de elementos pull, y como punto de convergencia están elementos "indiferentes" al tipo de acción como el elemento *queue*.

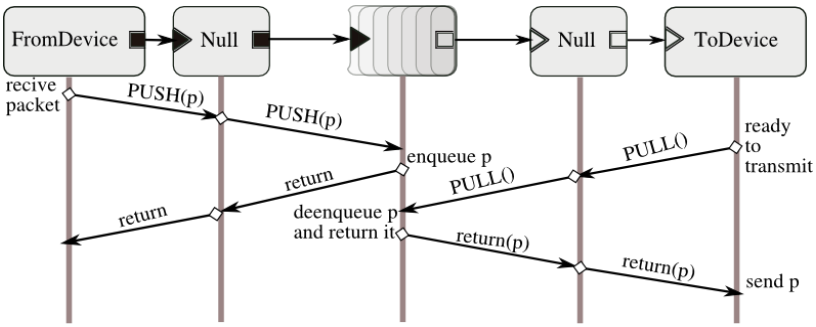


figura 98 Movimiento "Push" y "pull" entre elementos fuente: Autor

En la figura 98 sacada de la tesis del autor de CLICK, muestra las operaciones *push* y *pull,* donde la *queue* toma el papel de ser un acceso bidireccional entre las dos acciones, pero no ejecutando ninguna de las acciones de pasar paquetes.

El lenguaje de programación para CLICK es llamado click, utilizado para configurar el archivo de extensión .click; es escrito en un lenguaje tipo grafico con dos constructores:
Como lo define Eddie Kohler en su tesis,[1] el primero es Declaración; cuales elementos son creados, el segundo es Conexiones; como los elementos son conectados .Y poseen las siguientes formas de sintaxis:

name::class(config-string); Declaration.
nameI[portI] -> [portII]name2; Conexión.

Los punto y comas son opcionales, la sentencia de la conexión *"nameI[portI] -> [portII]name2"* crea una conexión desde la salida *namel*, del puerto *portI* hacia name2 entrada hacia el puerto portII. Los elementos deben ser declarados antes de ser usados, dentro de las conexiones.

Otra forma de establecer conexiones bastante amigable sería:[1]
//Declaraciones--

1 The Click Modular Router,Eddie Kohler, Submitted to the Department of Electrical Engineering and Computer Sciencein partial fulfillment of the requirements for the degree of Doctor of Philosophy at the
Massachusetts Institute of Technology February 2001.pags-28-30.

```
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;
//Conexiones--
  src -> ctr;
  ctr -> sink; o   src -> ctr -> sink;
```

Otras aproximaciones aceptables en la declaración y conexiones del lenguaje de programación de click serían:

```
name :: class;              // Puede omitir los argumentos.
name1 -> name2;       // omitir el número del puerto se toma como [0]
name1 [port1] -> [port2a] name2 [port2b] -> [port3] name3;//
name1 -> name2 :: class(config-string) -> name3; // puede declarar
                                                  //elementos dentro de las conexiones
```

También se puede crear clases de los elementos, sin escribir ningún código en c++, se denomina "*Compound element classes",* el siguiente código es un ejemplo sacado de la página oficial de CLICK que hace parte de la documentación:

```
elementclass ErrorChecker {
input -> cip :: CheckIPHeader
-> i1 :: IPClassifier(tcp, udp, icmp, -)
-> ctcp :: CheckTCPHeader
-> ttl :: IPFilter(drop ttl 0, allow all)
-> cl :: CheckLength(1500)
-> output;
i1[1] -> cudp :: CheckUDPHeader -> ttl
i1[2] -> cicmp :: CheckICMPHeader -> ttl
i1[3] -> ttl
}
```

Al definir el elemento con la palabra clave *elementclass {nombre de la clases particular}*, puede añadir mas elementos dentro de la nueva clase además se encuentran las palabras *input* y *output,* para establecer el punto de entrada y salida de los paquetes.

En la tesis descrita por el creador de este software clasificó los elementos según el tipo de tarea que hacen en el puerto de entrada y puerto de salida, entre estos se tiene:

- Fuente de paquetes
- Packet sinks (Paquetes para reciclar)
- modificadores de paquetes
- Elementos de enrutamiento

- Elementos de almacenamiento
- Elementos de información
- *Scheduling* elements

También hay presente extensiones de CLICK en sus ultimas versiones con propósitos especiales, tales como elementos para funciones *wireless accesspoint,* wifi.

Aun así referenciar a CLICK en una pequeña práctica, ejecutando CLICK, con el fin de tomar el campo TTL (TimeToLive) y disminuirlo dos valores como un punto de partida para entender lo básico de este *software* experimental.
Para la pequeña práctica de modificación del campo TTL, se creo el siguiente código:

FromHost(tap00, 20.30.10.1/24)->DecIPTTL->Print(ok)-> DecIPTTL ->IPPrint(ttl,TTL true)->PushNull->ToHost(tap00);

El código se resume en la utilización de seis elementos, todos con la debida documentación para su utilización en la página web http://read.cs.ucla.edu/click /elements.
Para decrecer el TTL se utilizó el elemento DecIPTTL dos veces y para ver el resultado en consola se utilizó IPPrint con la opción ttl.

Para la mini-práctica se utilizó el sistema operativo fedora 12, CLICK-1.8, una interface “tap00” y una maquina virtual para establecer la comunicación tipo *socket tcp* con la utilidad de red *netcat*. Ver figura 99

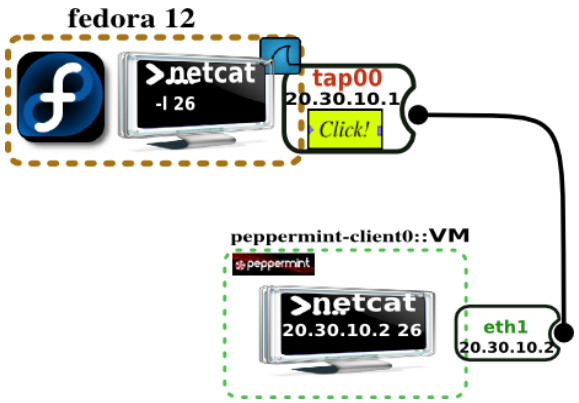


Figura 99 Diagrama práctica CLICK fuente Autor

Para comprobar la reducción del valor del campo TTL, se ejecuto la utilidad de red *netcat* sin CLICK. La utilización del *wireshark* para observar la salida de paquetes de la dirección 20.30.10.1 y ver el campo TTL que tiene un valor de 64, se puede observar en la figura 100 (Siguiente página).

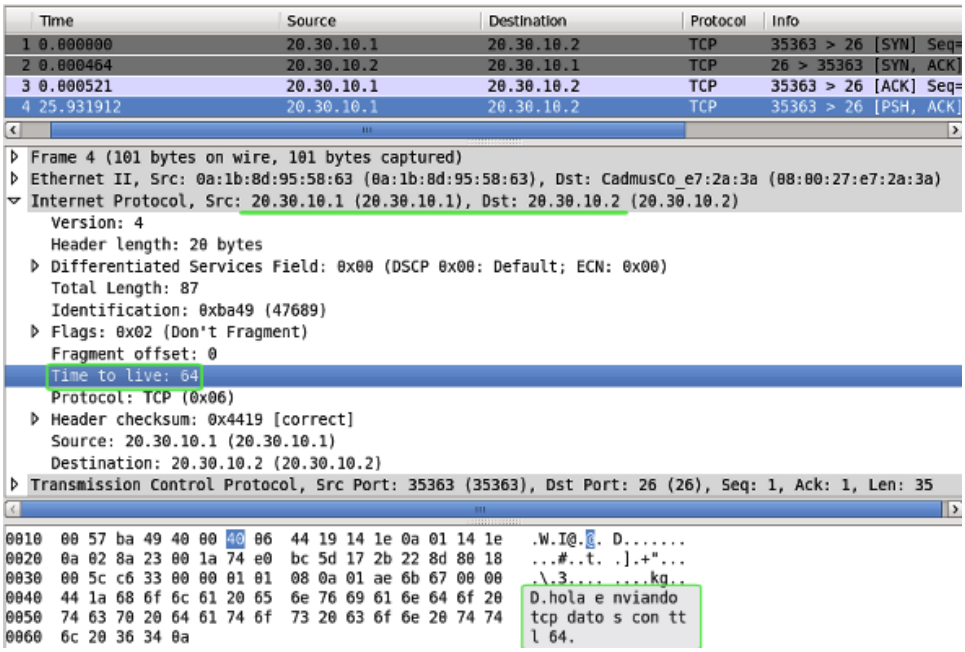


figura 100 TTL valor 64 *fuente : Autor*

A continuación en la figura 101 se puede observar como se ejecuta CLICK y la respuesta del elemento IPPrint en consola confirmando la reducción del campo TTL de 64 a 62, con la dirección de destino 20.30.10.2 y fuente 20.30.10.1.

```
[root@localy click-1.8.0]# cd user
bash: cd: user: No such file or directory
[root@localy click-1.8.0]# ls
aclocal.m4          click.spec          config.status       elementmap.xml  LICENSE         README
arpc                conf                config.sub          elements        linuxmodule     share
AUTHORS             config-bsdmodule.h.in configure         etc             m4              stamp-h
bin                 config.guess        configure.in        FAQ             Makefile        test
bsdmodule           config.h.in         config-userlevel.h.in include       Makefile.in     tools
click-buildtool     config-linuxmodule.h.in COPYING         INSTALL         missing         userlevel
click-buildtool.in  config.log          depcomp             installch       mkinstalldirs
click-compile       config.mk           DISTFILES           installch.in    NEWS
click-compile.in    config.mk.in        doc                 install-sh      po
click-mkelemmap     config-ns.h.in      drivers             lib             pathvars.h.in
[root@localy click-1.8.0]# cd user*
[root@localy userlevel]# ./click /home/Diegu/easy.click
ok:  112 | 080027e7 2a3a0a1b 8d955863 08004500 0062ba4a 40003f06
ok:   42 | 080027e7 2a3a0a1b 8d955863 08060001 08000604 0002091b
^C[root@localy userlevel]# ./click /home/Diegu/easy.click
ok:  152 | 080027e7 2a3a0a1b 8d955863 08004500 008aba4b 40003f06
ttl: 1301865742.492989: ttl 62 20.30.10.1.35363 > 20.30.10.2.26: P 1960885422:1960885508(86,152,138) ack 38870286
1 win 92
ok:   42 | 080027e7 2a3a0a1b 8d955863 08060001 08000604 0002091b
ttl: 1301865747.495992: ttl 8 88.99.20.30 > 10.1.8.0: ip-proto-27 (frag 1540:2048@16)
```

Figura 101 Ejecución de click en consola *fuente :Autor*

En la figura 102 se puede observar la confirmación de la reducción del campo TTL en dos unidades en la imagen de la captura por el *wireshark (Siguiente página).*

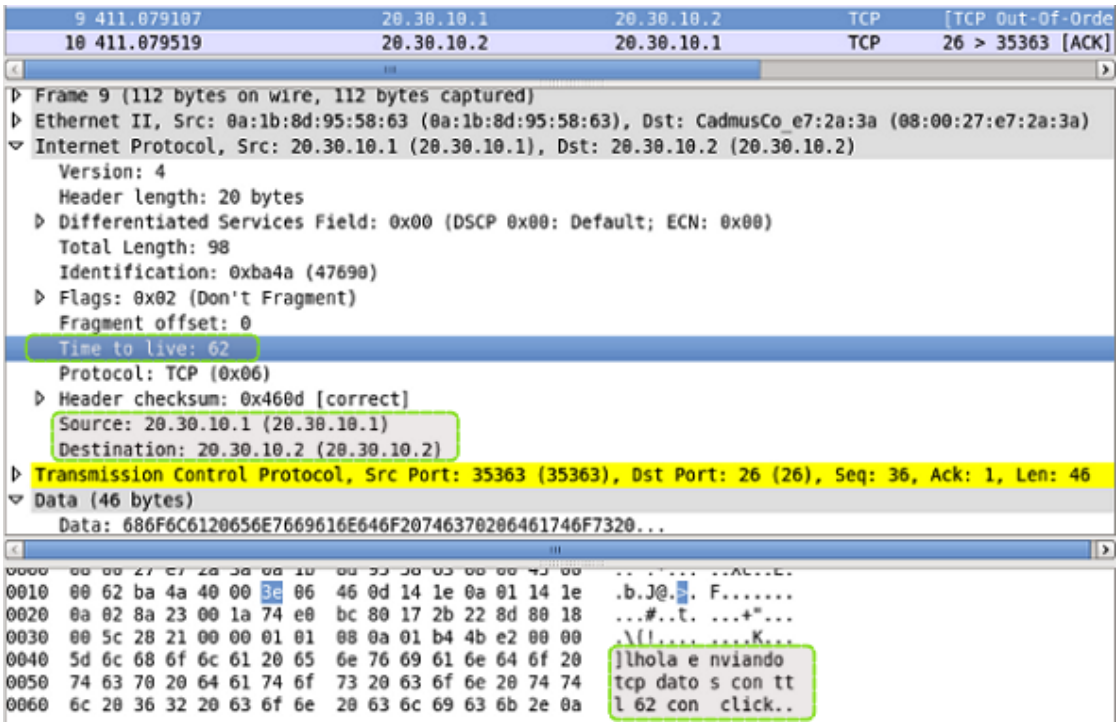


Figura 102 TTL valor 62 *fuente:Autor*

Una pequeña auditoria en el enlace punto a punto para la correcta comunicación tcp/socket se presenta en la figura 103, un *ping* y un *traceroute* para asegurarse que no habrá ningún decremento de campo TTL durante el recorrido de 20.30.10.2 a 20.30.10.1.

```
xorp-cliente0-desktop xorp-cliente0 # ping -c2 20.30.10.1
PING 20.30.10.1 (20.30.10.1) 56(84) bytes of data.
64 bytes from 20.30.10.1: icmp_seq=1 ttl=64 time=5.41 ms
64 bytes from 20.30.10.1: icmp_seq=2 ttl=64 time=5.34 ms

--- 20.30.10.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 5.349/5.381/5.414/0.080 ms
xorp-cliente0-desktop xorp-cliente0 # ip neighbor show
20.30.10.1 dev eth1 lladdr 06:90:cf:fb:b7:b4 REACHABLE
xorp-cliente0-desktop xorp-cliente0 # traceroute 20.30.10.1
traceroute to 20.30.10.1 (20.30.10.1), 30 hops max, 60 byte packets
 1  20.30.10.1 (20.30.10.1)  2.481 ms !X  2.492 ms !X  2.568 ms !X
xorp-cliente0-desktop xorp-cliente0 #
```

Figura 103 Traceroute cliente *fuente :Autor*

4. PRINCIPIOS BASICOS PARA LA CREACION DE UN NUEVO PROCESO EN XORP

En este capitulo se ahondara en documentar que necesita un usuario de XORP, que desempeñara el rol de desarrollador, para utilizar XORP en el campo de la Investigación.
Se empieza describiendo el perfil de desarrollador, que podrá hacerse cargo en la creación de un proceso. Se expondrá como crear los archivos de operación de XORP como son los .xif , .tgt, .tp tomando como ejemplo al Módulo OSPF; y como luego estos archivos hacen parte del proceso, que fue documentado paso a paso, en la creación de un Módulo.

Los Módulo tienen en común una forma particular de activación, propuestos por los creadores de XORP; se describió en forma de narración la serie de eventos que ocurren en el Módulo cuando el *router manager* hace solicitudes a los módulos.
También un estudio a la librería que se encarga del envió y recepción de xrls, llamada XrlRouter y del por que y para que del código auxiliar generado.
Para finalizar se expondrá una literatura relacionada en proyectos académicos que han utilizado XORP.

4.1 QUE NECESITA UN DESARROLLADOR SABER SOBRE XORP.

Para la creación de un proceso se deben tener claros conceptos como xrls, xrlTarget, xrl Interfaces, operaciones del finder, para esto se debe tener encuentra la documentación oficial de XORP, donde encontrara esta información en la documentación oficial en los títulos siguientes:

- *XORP Design Overview*
- *XORP LibXORP Library Overview*
- *XORP Inter-Process Communication Library Overview*
- *XRL Interfaces: Specifications and Tools*

También tener solidas bases y experiencia en c++ en especial múltiple herencia utilizando Interfaces para C++ y STL (standard Template Library).
Recomendable también estar familiarizado con el proceso de compilación de software, en especial con “scons”, aunque no es requisito obligatorio. Y tener buenos conocimientos en el sistema operativo Linux tanto para administración como para redes. sería recomendable también conocimientos en la creación de script ya sea en Bash, Python.

Es recomendable que el usuario tome su tiempo para estudiar las lecturas especificadas o en su defecto disponer del capitulo número 2 del presente trabajo. XORP prácticamente esta escrito en C++ y utiliza extremadamente la STL, y un

concepto de multiplexacion de interfaces de c++.
XORP adopta un modelo[1] *event-driven* y un simple hilo de ejecución asíncrono. Los eventos son generados por *Timers* y archivos descriptores como es el archivo *template* .tp.

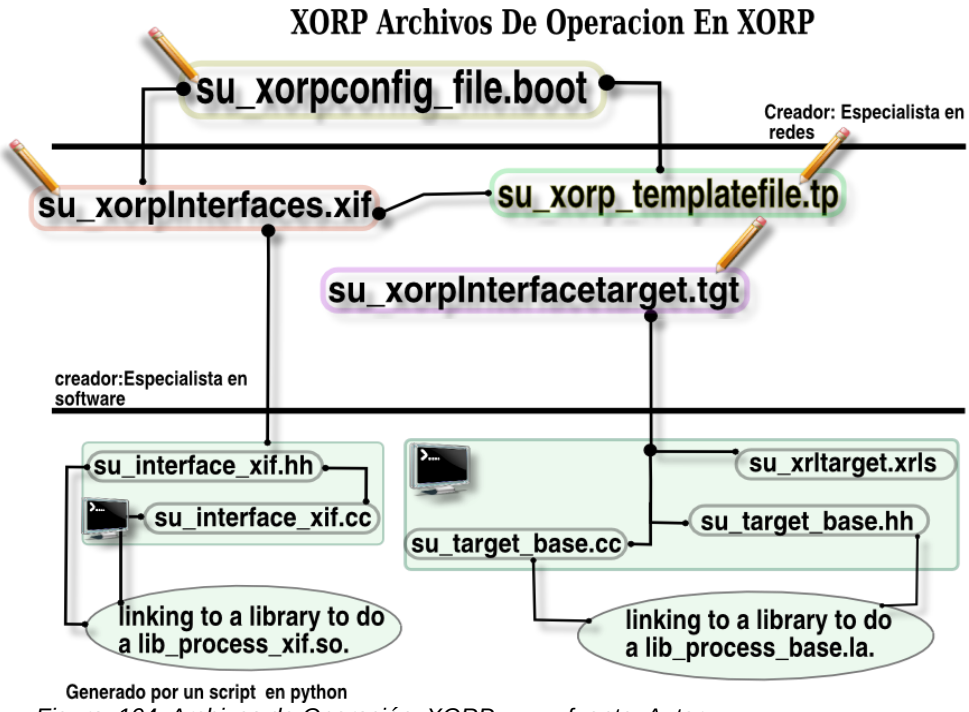


Figura: 104 Archivos de Operación XORP fuente: Autor

En la figura 104 se puede observar la serie de archivos funcionales necesarios para que XORP ejecute un proceso, sin mostrar aun las librerías de definición de nodos y el “main” del Módulo. En esta figura se clasifican los archivos según el tipo de función que desempeñan en XORP, organizados del nivel más alto de abstracción del proceso al más bajo. En la parte superior se encuentra un archivo con extensión “.boot”, el cual permite configurar al enrutador.

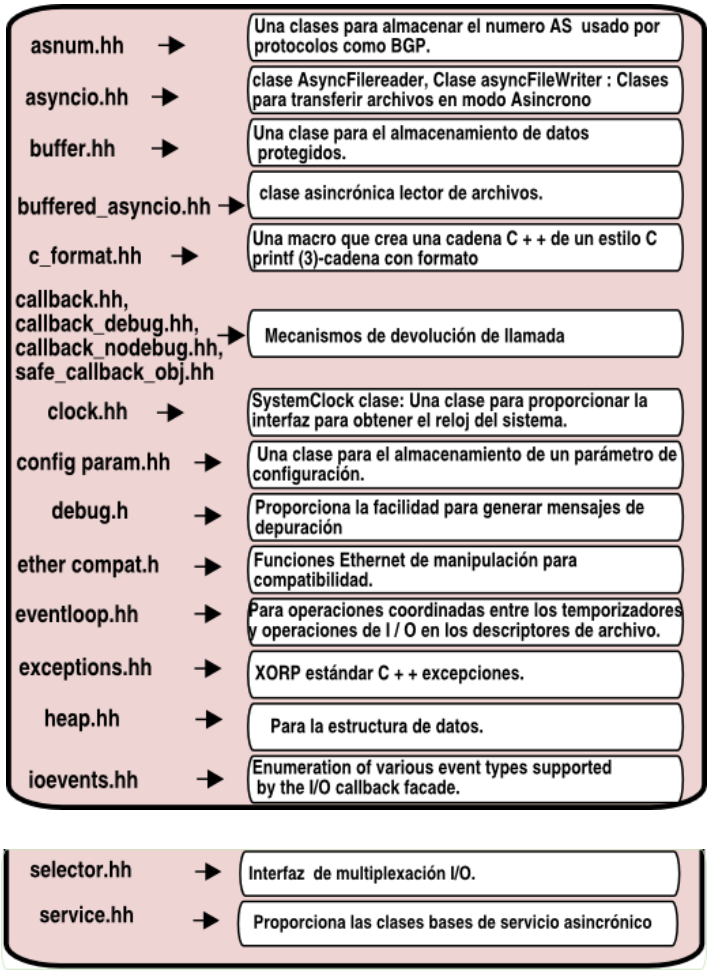
En la siguiente fila, en la zona intermedia se encuentran archivos que son creados por el programador, cada uno de estos archivos cumple un papel importante en la definición de las funciones que realizara el proceso y como se comunicaran con otros procesos de XORP. Estos archivos son creados en cualquier editor de texto convencional, mas adelante se procederá a dar mayor información acerca del papel que cumple cada archivo en las funciones y definición del nuevo proceso.

---

1 Designing Extensible IP Router Software. Mark Handley. Eddie Kohler.Atanu Ghosh.Orion Hodson. Pavlin Radoslavov. University College, London .p-4. UCLA.International Computer Science Institute.

Seguido, en la zona inferior de la figura, se presentan los archivos que son generados por software por medio de un script en *Python,* Para generar el código, utiliza las declaraciones y definiciones de los archivos de la fila intermedia, El código generado interviene directamente en la programación del nuevo proceso y contienen un conjunto de funciones Virtuales e interfaces de c++ dentro de una clases abstracta que lidian con la parte del transporte (El Finder) y verificación de un xrl.
Mas adelante se dará un ejemplo de como se generan estos archivos y de como estos interactúan con la programación final del Módulo.

XORP provee unas librerías que deberían acelerar el desarrollo en la creación de un nuevo Módulo están presentes en la ubicación /usr/local/xorp-1-8-CT/xorp/libxorp. La figura 105 se tiene una descripción de las librerías :



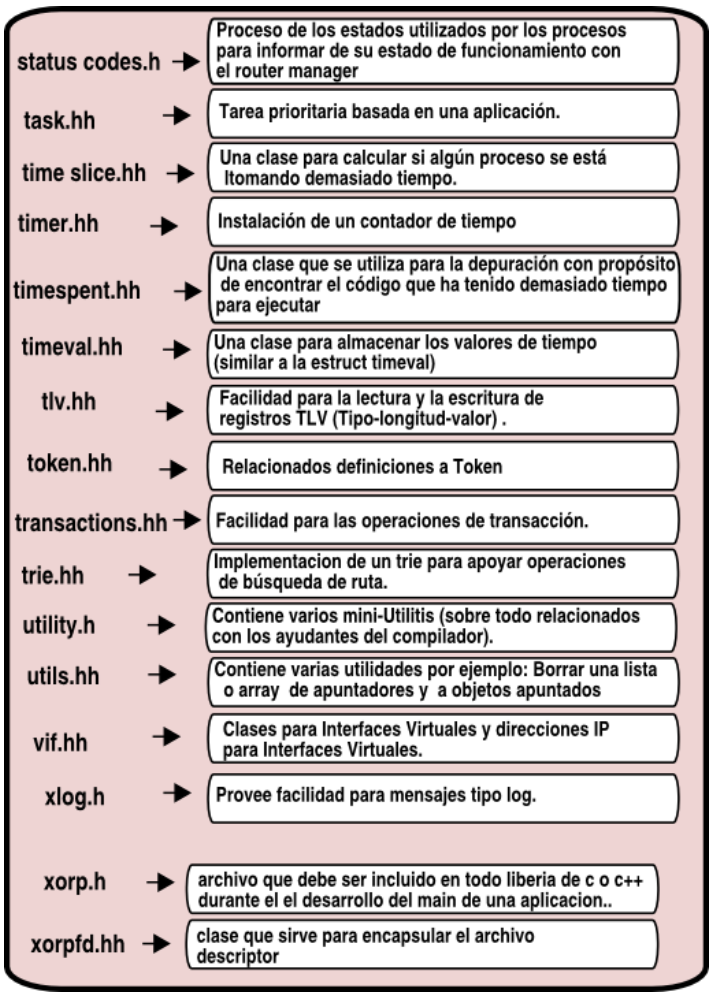


Figura 105 Librerías XORP fuente :Autor

Una de las características que deben ser tenidas en cuenta en el proceso de desarrollo y que XORP hace uso, cuando se ejecutan algunas pruebas después de la instalación exitosa, es la de procesar Scripts. En la carpeta /usr/local/xorp-1-8-CT/xorp/libxipc/tests, podrá encontrar las pruebas que XORP es sometido, utilizando scripts para hacer llamados a librerías muy especificas para comprobar la comunicación con el Finder.

También en la ubicación /usr/local/xorp-1-8-CT/xorp/libxorp/tests, encontrara algunos ejemplos de como utilizar las librerías que XORP provee para el proceso de desarrollo, señaladas en la Figura 105.

4.2 EXPLICACION DETALLADA DE LOS ARCHIVOS xrl-interface(.xif), Template (.tp), xrl-target (.tgt).

El archivo tipo .boot ya fue ampliamente utilizado y explicado en la ejecución de XORP, en modo de ejecución a un alto nivel, estudiado en el capitulo 3 donde se

lograron ejecutar protocolos como RIP, BGP, OSPF. En el capitulo 4 se estudiara los archivos .xif, .tp, .tgt, que son parte indiscutible de que protocolos de enrutamiento anteriormente mencionados den los resultados esperados.
Para dar definición y funciones de los archivos descriptores se utilizara como ejemplo el protocolo de enrutamiento OSPFv2.
OSPFv2 tiene ubicado los archivos descriptores de XORP-ct en:
- ./XORP-1-8-CT/XORP/xrl/interfaces/ospfv2.xif
- ./XORP-1-8-CT/XORP/xrl/targets/ospfv2.tgt
- ./XORP/share/XORP/templates/ospfv2.tp

El archivo "*template" o* plantilla es utilizado activamente por el proceso "xorp_rtrmgr" en la inicialización del *softroute,* ya que el rtrmgr lo emplea para crear un árbol de comandos dispuestos de manera ordenada que apuntan a los valores configurados en el archivo .boot obteniendo dichos valores e inyectarlos en el llamado de un XRL. A continuación fragmentos del archivo ospfv2.tp, describiendo las partes que lo conforman figura 106:

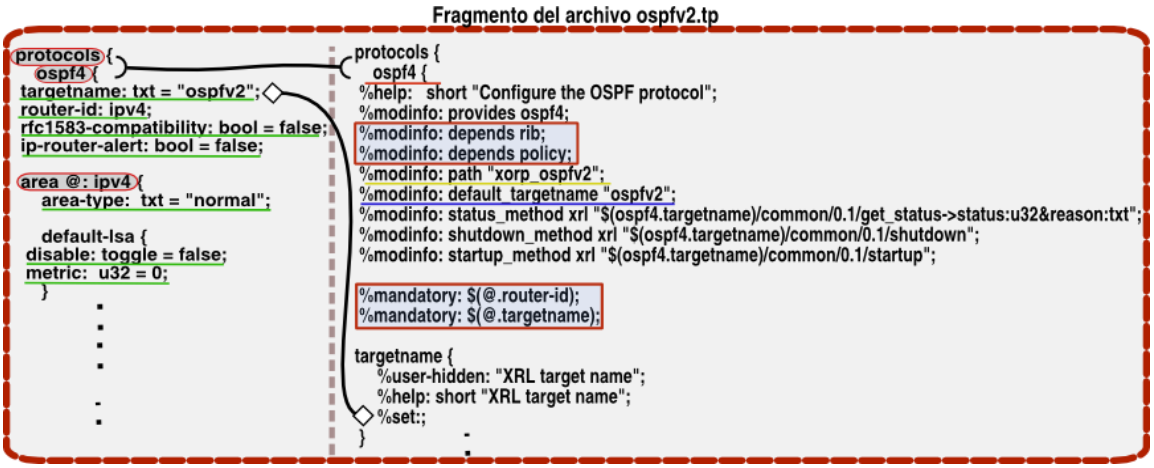


Figura 106 Fragmentos del archivo ospfv2.tp Fuente: Autor

El archivo *template* se construye en dos partes, la primera (en la parte izquierda de la figura 106) la declaración y ubicación jerárquica de los nodos (protocols, ospf, área, etc). Donde cada bloque de nodo puede contener definiciones de variables (referenciados en verde) las cuales son declaradas con los siguientes posibles tipos: text, bool, ipv4, ipv6, u32, com32, toggle, i32.[1]El símbolo del arroba "@" especifica que es obligatorio que el nodo "área" este definido en el archivo .boot y que el tipo de dato suministrado sea "ipv4". El nodo "área" presenta además un subnodo llamado "default-lsa" aunque no necesita valores para definir si toma los valores predefinidos como "false" y "0".
Para la figura 107 se muestran ordenados los comandos y acciones que hacen

1 XORP Router Manager Process (rtrmgr).Version 1.6.XORP, Inc. 2009.p-4

parte del archivo *template* (la parte derecha de la figura 106), con una corta

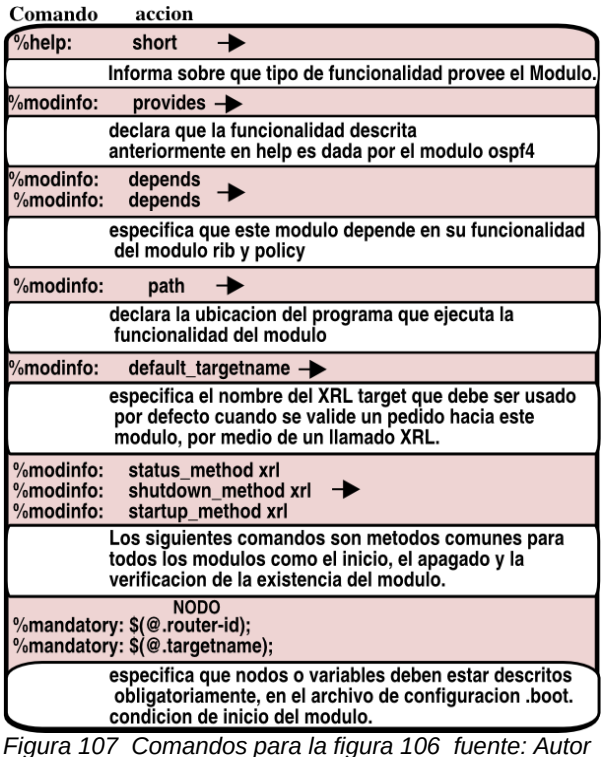


Figura 107 Comandos para la figura 106 fuente: Autor

explicación de las funciones que tiene cada comando. Específicamente para el nodo “protocolo ospf4”.

Para el siguiente nodo *“protocols ospf targetname”* declara la existencia de la variable “targetname” que define el *XRL target name* para el ejemplo ospfv2.

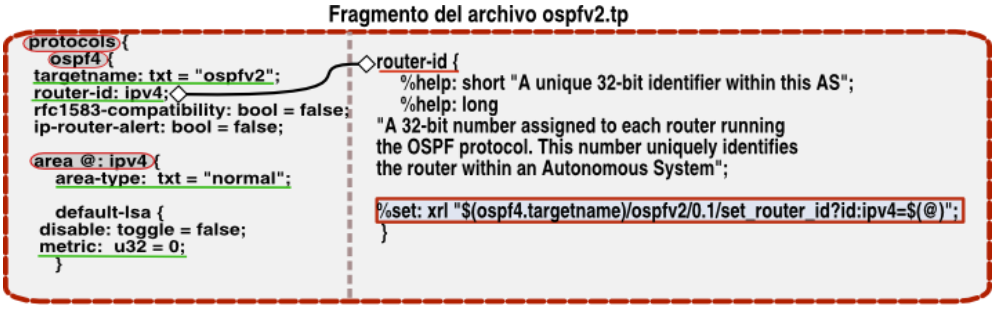


Figura 108 Nodos "route-id" fuente: Autor

Para la figura 108 continuando en la descripción del archivo ospfv2.tp, se tiene el nodo “protocols ospf router-id” donde se puede observar en la parte derecha el comando *%set:*

%set: xrl "$(ospf4.targetname)/ospfv2/0.1/set_router_id?id:ipv4=$(@)";

El comando seguido de la palabra clave o acción “xrl” declara que este xrl es

llamado por el rtrmgr y coloca el valor del router-id en el árbol de configuración.
Para esta declaración en particular se presentan dos símbolos $ y el @.
El símbolo $ trabaja como un apuntador de ubicación de nodos y variables, refiriéndose al $(ospf4.targetname) a la ubicación del nodo “protocolo ospf targetname” donde reside la definición de la variable *targetname* con el valor de “ospfv2”, el símbolo @ indica el valor definido en este subnodo, apuntando dicho valor con el símbolo $. En definitiva el rtrmgr va a realizar el llamado con el siguiente xrl: ***xrl "ospfv2/ospfv2/0.1/set_router_id?id:ipv4=1.2.3.4";***

Fragmento del archivo ospfv2.tp

```
protocols {
    ospf4 {
targetname: txt = "ospfv2";
router-id: ipv4;
rfc1583-compatibility: bool = false;
ip-router-alert: bool = false;

area @: ipv4 {
    area-type:  txt = "normal";

    default-lsa {
disable: toggle = false;
metric:  u32 = 0;
    }
            .
            .
```

```
rfc1583-compatibility {
    %help: short "Criteria for handling AS external routes";

    %set: xrl "$(ospf4.targetname)/ospfv2/0.1/set_rfc1583_compatibility?compatibility:bool=$(@)";
    %delete: xrl "$(ospf4.targetname)/ospfv2/0.1/set_rfc1583_compatibility?compatibility:bool=$(DEFAULT)";
}

ip-router-alert {
    %help: short "Send the IP router alert option in packets";
    %help: long
"If this option is true the IP router alert option will be set in all
transmitted packets";

    %set: xrl "$(ospf4.targetname)/ospfv2/0.1/set_ip_router_alert?ip_router_alert:bool=$(@)";
    %delete: xrl "$(ospf4.targetname)/ospfv2/0.1/set_ip_router_alert?ip_router_alert:bool=$(DEFAULT)";
}
```

Figura 109 Nodos "rfc1583-compabillity" y "ip-route-alert" fuente :Autor

Para la figura 109 se tienen la definición de los subnodos *rfc1583-compatibility* y *ip-router-alert,* lo que hay que destacar de la figura anterior es el comando %delete con la acción xrl donde se encuentra la palabra clave DEFAULT, si se llegan a declarar estas dediciones en el archivo .boot el rtmgr borrara los valores pre definidos y utilizara el %set para colocar los nuevos.

Fragmento del archivo ospfv2.tp

```
protocols {
    ospf4 {
targetname: txt = "ospfv2";
router-id: ipv4;
rfc1583-compatibility: bool = false;
ip-router-alert: bool = false;

area @: ipv4 {
    area-type:  txt = "normal";

    default-lsa {
disable: toggle = false;
metric:  u32 = 0;
    }
            .
            .
```

```
area @: ipv4 {
    %help: short "The OSPF area to which the attached network belongs";

    area-type {
%help: short "Type of area";
%allow: $(@) "normal" %help: "OSPF normal area";
%allow: $(@) "stub" %help: "OSPF stubby area";
%allow: $(@) "nssa" %help: "OSPF not-so-stubby area";

    %set: xrl "$(ospf4.targetname)/ospfv2/0.1/change_area_router_type?area:ipv4=$(area.@)&type:txt=$(@)";
    %delete: xrl "$(ospf4.targetname)/ospfv2/0.1/change_area_router_type?area:ipv4=$(area.@)&type:txt=$(DEFAULT)";
}

%create: xrl "$(ospf4.targetname)/ospfv2/0.1/create_area_router?area:ipv4=$(@)&type:txt=$(@.area-type)";
%delete: xrl "$(ospf4.targetname)/ospfv2/0.1/destroy_area_router?area:ipv4=$(@)";
```

Figura 110 Nodos "area" y "area-type" fuente: Autor

Para la figura 110 se destacan el comando *%allow*[1] y *%create.* El primer comando se aplica cuando de alguna forma se requiere restringir los valores que pueda tener la variable, para el caso de ospfv2 esta restringido a los valores de tipo texto

1 XORP Router Manager Process (rtrmgr).Version 1.6.XORP, Inc.http://www.xorp.January 7, 2009.p-13

“normal”, “stub” y “nssa”. El segundo comando %create es utilizado para crear en el árbol de configuración el espacio de la variable que corresponde a area@:ipv4 luego permitiendo la modificación de dicho espacio con comando %set.

El archivo *Template* expone las posibles funcionalidades y diagramación en un orden jerárquico de un proceso, reflejándose en la configuración del archivo de extensión .boot, haciendo énfasis en la declaración de tipos de variables y la activación de comandos que tienen como medio de propagación las formas no resueltas XRLs.

Estos xrl pedidos por el rtrmgr pueden en algunos casos generar otros llamados hacia otros módulos después de ser resueltos por el finder, entonces un Módulo debe tener configurado las interfaces de llamado y de que tipos de llamados puede soportar hacia otros módulos es allí donde los archivos de extensión .xif y .tgt toman este papel.

El archivo .xif expone las definiciones de los métodos públicos (en programación), que el Módulo desencadena cuando interactúa con el usuario, por ejemplo para el archivo ospfv2.xif, tiene el método “create_area_router” este método es disparado por medio del *xrl "ospfv2/ospfv2/0.1/create_area_router?id:ipv4=1.2.3.4";* definido en el ospfv2.tp, entonces para que el desarrollador o usuario que esta experimentado con XORP, pueda declarar y luego utilizar este método, primero tiene crear el archivo .xif, cuyo formato es descrito en los siguiente párrafos.

Inicia con la palabra clave “interface” seguido por el nombre y versión de la interface en particular, para el caso de ospfv2 sería así:

interface ospfv2/0.1 { ....}

después vendría la lista de los Metodos que soporta dicho Módulo entre los corchetes, para cada dedición de un método hay una línea, donde se define el nombre del método y los parámetros que tiene el método. El punto de partida de los parámetros inicia con el símbolo “?”. El número de los parámetros pueden ser mas de uno, la separación se realiza con el símbolo &; para saltar de línea se utiliza “\” en caso que se tengan varios parámetros.

```
create_area_router ? area:ipv4 \
                        & type:txt
```

También es permitido realizar comentarios al estilo del lenguaje c. En caso que se requiera algún dato la utilización de definición del tipo de retorno de algún método se declara de la siguiente forma “-> {retorno}”

```
/**
    * Get the list of neighbours.
    * Return a list of u32 type values. Each value is an internal
    * identifier that can be used with the get_neighbour_info XRL.
*/
```

get_neighbour_list -> areas:list<u32>;

En la figura 111 se representa las partes funcionales que debe tener un Módulo sin contar con el archivo *template.* En la figura tiene fragmentos del archivo para

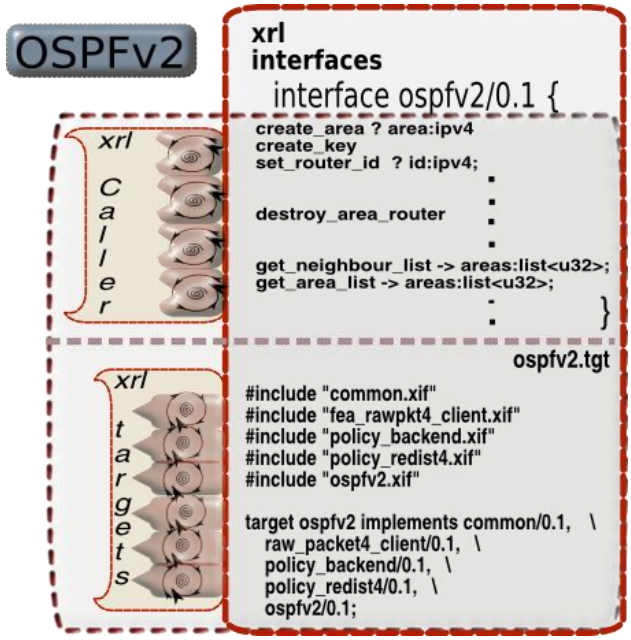


Figura 111 Interface "ospfv2" fuente : Autor

ospfv2.xif detallando los métodos públicos creados y accesibles al usuario.

también existen métodos ajenos o que ya se encuentran definidos en otros módulos necesarios para la construcción y definición de funcionalidades de un Módulo en particular, estas definiciones se encuentran en el archivo tipo .tgt (la parte inferior de la figura) donde se declara que tipo de métodos externos que puede acceder un Módulo a través de otros archivos .xif . En la figura anterior el Módulo ospfv2 necesita tener acceso a fea_rawpkt4_client.xif , policy_redist4.xif, common.xif, policy_backend.cif, ospfv2.xif.

En la construcción del archivo .tgt se declaran los archivos tipo .xif de módulos foráneos que se necesitan incluir, ya que estos poseen métodos para la definición de un Módulo en particular. A continuación una descripción de como construir el archivo tipo .tgt:
La palabra clave # include usado en c++, se utiliza en la declaración del Módulo, de la siguiente forma:
include {archivo.xif}. Ver figura 111
luego se llaman los archivos con la siguiente sintaxis,
target {nombre del target_name definido} implements { (nombre de la interface / versión)\

(nombre de la interface/versión)\
(nombre de la interface/versión)

4.3 PASOS PARA CREAR UN PROCESO EN XORP.

En cualquier editor de texto se pueden crear los archivos de ejecución de XORP. En este punto ya se debe tener claro que es lo que el Módulo va a implementar y que necesitara de otros módulos. Los pasos son los siguientes:

i. Definición del archivo {mi Módulo}.xif en */usr/local/xorp-1-8-CT/xorp/xrl/interfaces:* En este directorio es donde se ubican todas las definiciones de los módulos que integran a XORP.

ii. Modificar el archivo de la fuente de compilación "Sconscript", para incluir en la lista a el {mi Módulo}.xif, el script esta ubicado en */usr/local/xorp-1-8-CT/xorp/xrl/interfaces.*

iii. Después de ejecutar el script y recompilar, se genera el código auxiliar generado las siguientes librerías {mi Módulo_xif}.hh y {mi Módulo_xif}.cc. Después por medio de otro llamado a un script este enlaza las librerías anteriores y compila en *libxif_{mi Módulo}_interface.so.*

iv. Definir el archivo {miMódulo_target}.tgt en */usr/local/xorp-1-8-CT/xorp/xrl/ targets,* luego modifique el "Sconscript", específicamente la lista *tgts = [*
'bgp.tgt',
'cli.tgt',
' coord.tgt',...]
añadiendo el nombre {miMódulo_target}.tgt.

v. Después de ejecutar el script y recompilar, se generara un código para el *target,* este código aun no esta completo para los pedidos de un *xrl target.* Incluye a {miMódulo_target}.xrls, {miMódulo_target}_base.hh y {miMódulo_target}_base.cc, los dos últimos archivos son compilados y enlazados a la librería *libxst_{mi Módulo}_target.so.*

vi. Seguido viene la creación del Módulo e implementación del código de llamado.

El literal 3 y 5 hacen referencia al proceso de generación de código auxiliar, por medio de un *script* escrito en Python, con el fin que el usuario que esta experimentando con XORP, no tenga que crear el código para lidiar con la lógica

del transporte y tratamiento de un xrl para el envió y recepción, ya que esta tarea es común para todos los módulos.
Para el literal 2 y 4 tiene relación en la inclusión del archivo de extensión .xif y .tgt, para el proceso de compilación y generación de código.

4.4 CONCEPTOS Y SUCESOS INVOLUCRADOS EN LA ACTIVACIÓN DE UN MÓDULO

El concepto de modularización utilizado por los desarrolladores de XORP se ve reflejado en la forma como XORP compila separadamente cada Módulo pero permitiendo tener conexiones con otros módulos. Suponiendo que cada Módulo sabe de la existencia de otros Módulos, para el caso de XORP se necesitan el soporte de ubicación que ofrece el Finder y los métodos de direccionamiento XRL. El código auxiliar generado por los script de Python permite al Módulo poder lidiar con el direccionamiento que el Finder ofrece.
Entonces un Módulo en XOPR constaría de las Interfaces (.xif,.tgt), la implementación ({nombredelMódulo}_node.hh) y la comunicación entre procesos realizada por el Finder. Cada una de estas características esta rodeada de herramientas (clases), que permiten lograr el objetivo del Módulo.

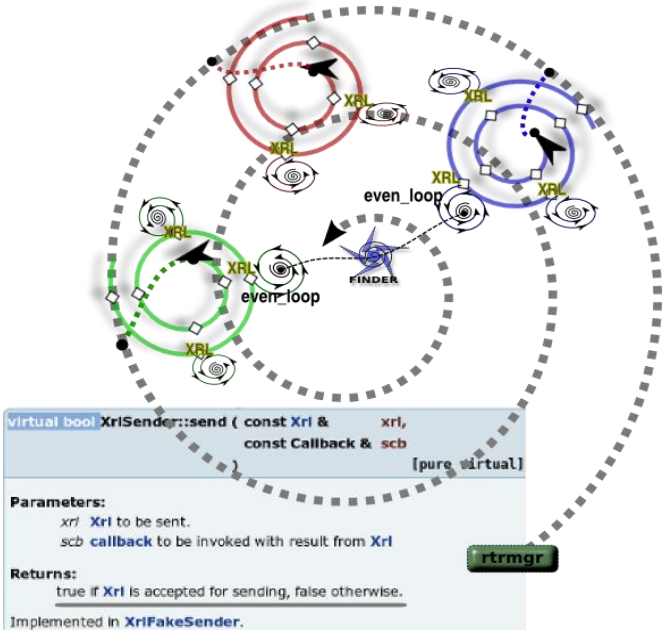


Figura 112 Bucle de ejecución de XORP. fuente: Autor

Otras generalidades de XORP referente a la programación que ya se menciono, es el hecho que XORP es ejecutado en un sólo hilo de ejecución y conducido por eventos.
Para la figura 112, una vez que el rtrmgr haya dispuesto el orden de ejecución de

los módulos y los módulos estén dispuestos a recibir pedidos para sus interfaces, el rtrmgr va ejecutando Módulo tras Módulo, haciendo solicitudes hacia las interfaces. Durante este proceso de llamadas, puede suceder que un Módulo deba enviar o requerir información hacia otro Módulo, generando una cola de eventos por las solicitudes de Módulos interesados en algún método en particular; los eventos son generados por llegadas o salidas de xrl, Timers, eventos para operaciones *sockets* I/O (ver figura 32, nivel naranja).
Cada método solicitado es acompañado por una función *callback* (Disponible en la librería de XORP). Entonces un método es solicitado por un Módulo X solicitud enviada con un formato de llamada xrl, generando un Evento al arribo de la solicitud al Módulo Y. En la solicitud del Módulo X en el código, lleva los parámetros necesarios para la ejecución de Método del Módulo Y, además lleva atado una función a la cual se remite la respuesta, llamada función *callback().*
En la programación orientada a eventos es necesario tener una función a la cual remitir una respuesta al disparo de un evento. Como por ejemplo cuando se le da click a un botón en una página web. El evento click dispara un función “callback” para implementar alguna acción.

Entonces el Módulo deberá decidir en una cola de eventos cual accionar primero; Para dar orden y compas (tiempo) de activación a los eventos. Los desarrolladores de XORP crearon unas librerías que se encarga de procesar la serie de eventos priorizando y disponiendo el tiempo en que se debe cargar al bucle principal de ejecución, es llamada *event_loop (Disponible en la librería de XORP)*. La librería se encuentra en la ubicación */usr/local/xorp-1-8-CT/xorp/libxorp;* en el archivo esta presente una descripción de la función que cumple esta librería cuando es llamada , aqui la descripción del equipo de desarrollo de XORP.

> *Co-ordinates interactions between a TimerList and a SelectorList for Xorp processes. All XorpTimer and select operations should be co-ordinated through this interface.*
> *Invoke all pending callbacks relating to XorpTimer and file descriptor activity.*

En resumen:

Un Módulo cliente hace una llamada asíncrona tipo XRL, para ello debe preparar por cada llamada un objeto “callback”, luego en un tiempo dado, este pasa por una función event_loop. Este event_loop, recoge la respuesta, que esta presente en un formato muy particular, para luego *unmarshalls* y poder invocar al objeto “callback” con el retorno de la respuesta. En caso de algún error, será notificado al Módulo y descrito por la librería estándar de errores de XORP llamada xlog. La anterior serie de eventos descritos es una síntesis del contenido en el archivo de la documentación oficial de xorp llamado An Introduction to Writing a XORP Process, explicando el movimiento entre las clases y funciones a declarar. Le

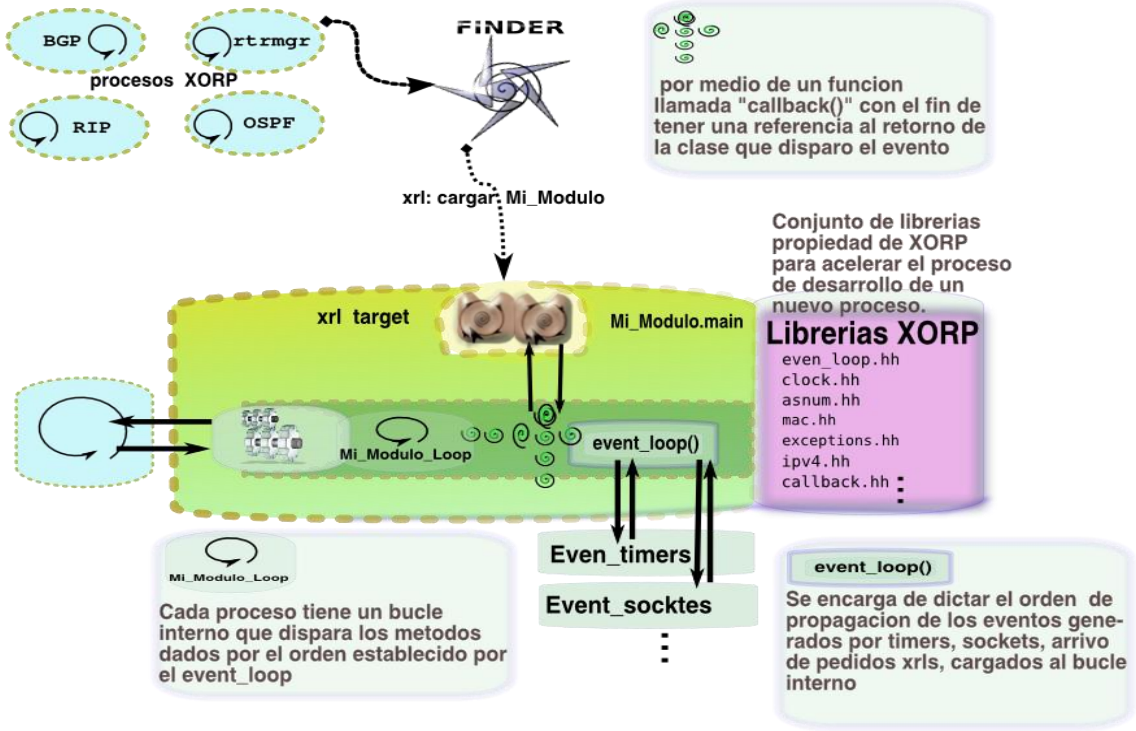


Figura 113 Descripción grafica en la activación de un Módulo fuente: Autor

puede dar al lector principios básicos de como desarrollar procesos en XORP.

En la figura 113 se describe gráficamente como es el mecanismo de inicialización del Módulo, definida en el archivo .main, utilizando librerías como *eventloop, xlog, xorp, callback, exceptions*;

Se podría decir que en el archivo .main de c++ se programa la parte de control de errores y eventos además la activación de un Módulo y la ejecución del bucle interno del Módulo para que interactué con la librería event_loop, esto se puede ver en el archivo de c++ xorp_static_routes.cc ubicado en usr/local/xorp-1-8-CT/xorp/static_routes.

En la descripción hecha del archivo plantilla se detallaron las operaciones por medio de comandos, la descripción y ubicación de los métodos xrl, también se recalco que dichas operaciones hacen parte de lo que se llama operaciones de nodo.

Entonces una vez XORP halla resuelto la parte de transporte y resolución de un xrl, es en el archivo _node.hh, donde se recibe la respuesta y se implementa. Por su puesto haciendo uso de las herramientas de programación de c++ ofrece, como Multiherencia.

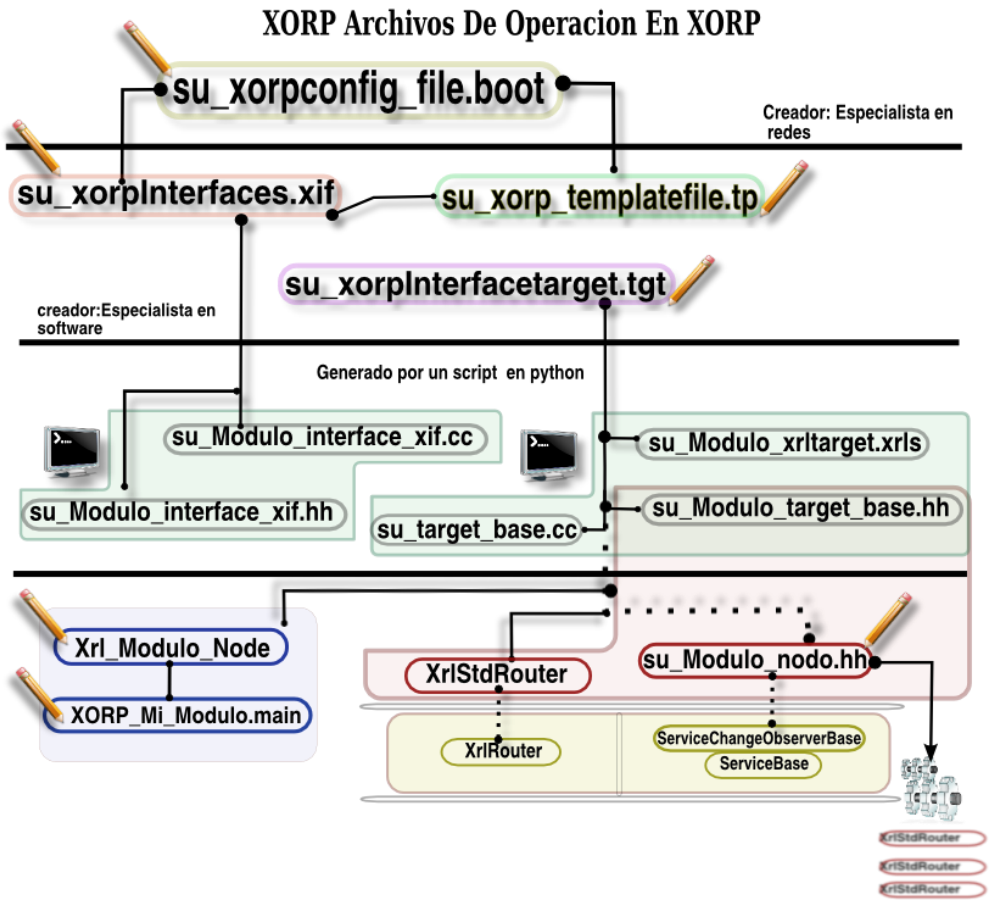


Figura 114 Arbol de archivos de codigo completo para crear un proceso Fuente: Autor

En la figura 114 se muestra los archivos involucrados para la creacion de un proceso allí destacan los archivos con la señalización del lápiz. Creados por el desarrollador de *software*. En amarillo las clases bases para comunicación y controladores de eventos acerca del estado del Módulo. En azul la clase Xrl_Módulo_Node encargada de la logística de tratamiento de errores con el Finder, al enviar solicitudes a un Módulo foráneo, en caso de que el Módulo foráneo este desactivado o algún error en la comunicación y además establecer las operaciones de un Módulo con los demás Módulos foráneos y la clase Main encargada de inicializar el bucle interno del Módulo.

4.4.1 Descripción de la librería XrlRouter

La complejidad detrás de los pedidos de comunicación hacia el Finder, queda encapsulada con la implementación de la clase llamada xrl_router definida en el código fuente en */usr/local/xorp-1-8-CT/xorp/libxipc,* tan sólo observando el tipo de

métodos y definición de los constructores se puede ver que detrás de esta clase esta la complejidad de tareas como envió y recepción de xrl (*unmarshall* el xrl), autentificación xrl y control de errores.

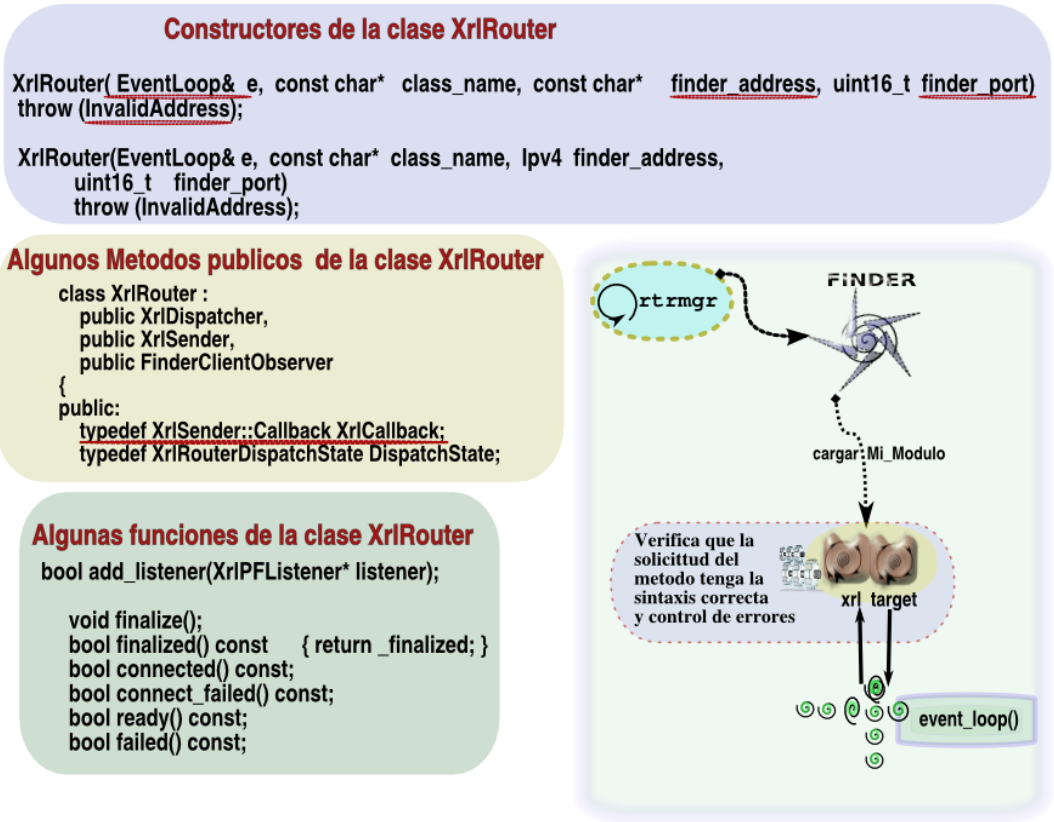


Figura 115 Fragmentos librería XrlXorp fuente: Autor

En el cuadro verde de la figura 115 se puede ver funciones típicas de operaciones en la comunicación tipo *socket.* En el cuadro café se observan los métodos públicos y definición de nuevos tipos relacionados con la clase *XrlSender*, que es una clase base para el transporte de un xrl; Donde esta clase especifica si es aceptable el envió de un xrl (con los correctos parámetros para la ejecución del método solicitado) . Además definición de un tipo callback, para luego ser invocado con el resultado del xrl (implementación) que fue aceptado por el *XrlSender.* En la parte de los constructores se puede observar los requisitos mínimos para iniciar una comunicación tipo *socket* (puerto, dirección), también incluye a la clase *evenLoop*, que desde luego es necesitada para que la solicitud del xrl sea cargada al bucle principal.

4.4.2 Explicación del código auxiliar generado

En la parte referente a la programación de un Módulo en como operan las

interfaces, se puede detectar 3 tareas o también se podrían llamar estados que desarrollan. La primer tarea involucra el código para la recepción del xrl (estado *xrl target*), la segunda la implementación del xrl (estado procesamiento del xrl) y la tercera el envió de un xrl (estado *xrl caller*); para enviar un valor en respuesta a una solicitud. Todas las anteriores tareas hacen parte del desarrollo para la programación de un Módulo, específicamente en la inicialización con el rtrmgr y ampliamente relacionadas con operaciones del *finder* específicamente a la clase XrlRouter.

El conjunto de operaciones y librerías que conforman un Módulo se observan diferencias operacionales como son el código de control (recepción y envió de un xrl), el código de implementación (código para cada nodo) del Módulo y el código para activación del Módulo (librería de errores, ejecución del *evenloop,* manejo de la función *callback,* información relacionada con el finder).

Durante el proceso de comunicación entre módulos en la recepción de un xrl; el xrl es presentado al Módulo en una sintaxis muy particular y compleja. Esta tarea de *unmarshall* es manejada por el código de generación auxiliar específicamente para el Módulo de xrltarget ver figura 114, los motivos de esta sintaxis no son mas que los de seguridad a la comunicación con mensajes xrls.
Además una parte del código auxiliar generado esta relacionado con el código de control específicamente en la definición en código de c++ de los métodos xrls para la comunicación; que incluye a los argumentos del método, una función *callback* y aceptación o negación de un xrl por la clase *XrlSender.*

En el anterior párrafo se puede inferir dos momentos que ocurren en la comunicación de un xrl, el primer momento la definición de métodos *xrltarget* para que sean llamados, permitiendo una operación *unmarshall* de la solitud al Módulo y el segundo momento la intervención de la función *callback,* cuando el Módulo solicita la ejecución de un método foráneo a través de un xrl hacia otro Módulo; necesitando un modo de retorno de esta respuesta, esto es logrado con la función *callback()* disponible en la librería de xorp del código fuente.

4.5 TRABAJOS ACADÉMICOS RELACIONADOS CON XORP

Para los desarrolladores principales, creadores de la compañía XORP.inc, no cabe duda que una de las razones de desarrollo de XORP como una herramienta para cubrir el espacio que hay en el proceso de desarrollo de nuevos algoritmos de enrutamiento o modificación de los existentes en el laboratorio de redes y tener la facilidad de llevarlos a un ambiente de producción.

A continuación una descripción de trabajos académicos relacionado con XORP:

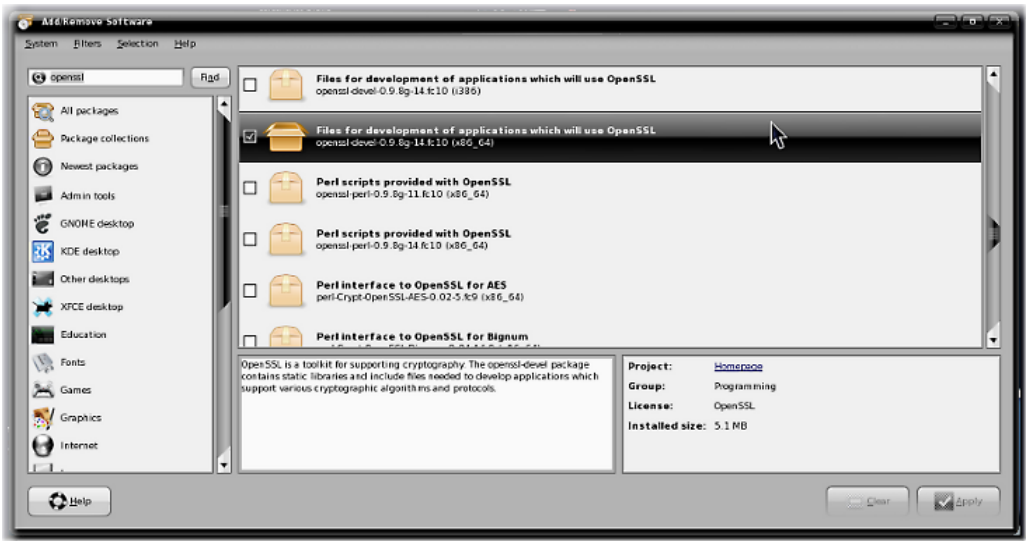
- *HLP: A Next Generation Inter-domain Routing Protocol:* El objetivo de la investigación es proponer una alternativa a BGP, con el desarrollo de un nuevo algoritmo de enrutamiento llamado HLP (Hibry-Link-Path). Un prototipo de este algoritmo fue desarrollado como Módulo en XORP.

- Mejoramiento e Implementación De un Algoritmo De Prevención Bucles Transitorios Durante La Convergencia De OSPF: El trabajo presentado para obtener el titulo de maestría informática. Crea un nuevo Módulo llamado "*OSPF_LOOPFREE*", que trabaja conjuntamente con OSPF, el cual aplica un algoritmo para mejorar el comportamiento en redes que utilizan a "link-state protocol". Para realizar el estudio el investigador cambio la métrica de una ruta en la LSBD, de este modo genera, un bucle transitorio. Luego de recrear el bucle transitorio pone en acción un algoritmo para evitar que el bucle transitorio aparezca.

- *Implementing NAT Support Into The XORP:* Este trabajo detallado en la tesis para el departamento de Ciencias de la Computación, de la Universidad de Copenague, Dinamarca. Describe la creación de un Módulo llamado XORP_NAT. No se muestran resultados de la implementación, sólo describe el marco teórico necesario para construir el Módulo. Aduce falta de tiempo y complejidad de la implementación en C++.

Anexo A. Instalación de 1.8XORP-CT en un Sistema Operativo Fedora 13

La instalación se realizo a un fedora 13 *Goddard,* los paquetes necesarios en la instalación son: openssl-devel, ncurses-lib , libpcap-devel, C++ suport to te GNU compiler Collection, te standard c++ libraries, glibc-devel, scons, boost c++ libraries, boost-devel y git.
El Package Manager se encuentra ubicado en el menú del escritorio en system /Administrations/ add/remove software.



. utilidad add remove software fuente: autor

El Package Manager para GNOME ofrece una descripción del paquete a instalar, aquí una corta definición de estos:
openssl: El conjunto de herramientas de openssl, provee soporte para la comunicación segura entre maquinas. Manejo de certificados, algoritmos de

encriptación y protocolos.
ncurses:Provee una API de programación para la creación de interfaces de usuario basadas en texto
libpcap:Provee un acceso al manejo de la red a bajo nivel, enfocado a monitoreo y estadística
boost:Conjunto de librerías libres que trabajan en conjunción con C++ Standard Library
scons:Es una herramienta de construcción de software.
Esta utilidad permite buscar e instalar paquetes firmados exclusivamente para fedora 13. Para los usuarios de Ubuntu, abrir consola y digitar: sudo apt-get install scons, libboost-regex-dev, libboost-regex1.40-dev, libssl-dev, libpcap-dev, libncurses5-dev.

Después de esto, se descarga 1.8XORP-CT, una opción para la descarga es con la utilidad, git ; se abre la consola y digita el siguiente comando:
git clone http://github.com/greearb/XORP.ct.git /usr/local, esta utilidad le asegura que los archivos que residen en servidor de git, son efectivamente una copia fiel a los que llegaran a la carpeta /usr/local.
Siguiente paso:
Antes de iniciar la instalación manual se debe copiar una librería de openssl, aquí el registro del bug, en la siguiente figura sin esta librería en la ubicación predeterminada, no podrá compilar. Para fedora 12 no hay registro de tal bug.

Registro del bug #559953 openssl Fuente:www.spinics.net/lists/fedora-package-announce/msg41855.html

Para corregir esto se hace lo siguiente.
En consola: cp /usr/lib/libcrypto.so /lib

El siguiente paso es crear el usuario XORP y el grupo XORP, este paso es importante porque da al usuario capacidades de utilizar la línea de comando de XORP, el Módulo xorpsh.
El XORP-ct team, creó un script en BASH, que se encarga de crear el usuario y el grupo, y se ejecuta de la siguiente manera: CD /usr/local/XORP; luego ./xorp_install.bash
asegúrese que esté ubicado en la carpeta XORP, en /usr/local/ de otro modo le generara un error.
Ya terminado los pasos de la pre-instalación, se comienza con la construcción del binario y su instalación. Aquí los pasos a seguir:

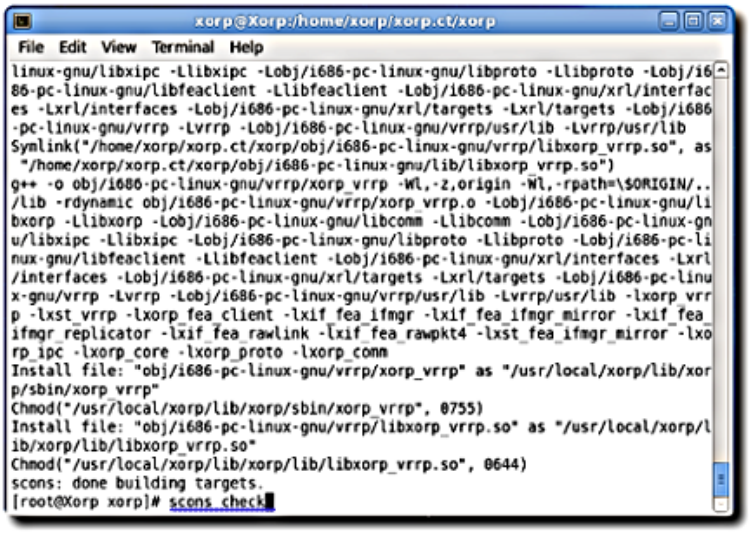
1 .La construcción del binario, XORP-CT nos ofrece una serie de opciones de construcción, en las cuales se puede descartar protocolos y funcionalidades de XORP, con el fin de enfocar la instalación a un ambiente embebido, aquí las opciones de construcción[1]:

"scons disable_ipv6=yes disable_fw=yes disable_warninglogs=yes \
disable_tracelogs=yes disable_fatallogs=yes disable_infologs=yes \
disable_assertlogs=yes disable_errorlogs=yes enable_olsr=no \
enable_bgp=no enable_ospf=no enable_vrrp=no enable_policy=no \
enable_rip=no optimize=size enable_fea_dummy=no enable_XORPsh=no \
disable_profile=yes enable_ustl=no"

Las opciones las puede visualizar con el comando "scons - -help" desde consola. La instalación que se está documentado se realizo completa, no se utilizaron ninguna de las opciones anteriores.

Seguido se pasa a digitar en consola cd /usr/local/XORP , luego "scons", después de la construcción del binario y su compilación, se prosigue a implementar la instalación, se digita en consola "scons install".

Toma aproximadamente con un procesador core 2 duo de 2Ghz, 20 minutos. Observara la siguiente salida, después de haber terminado de instalar.



Salida de consola después de una exitosa instalación Fuente: Autor

2. A continuación, si no ha tenido ningún problema con la instalación, tendrá la posibilidad de ejecutar una serie de pruebas de resistencia pertinentes a

1 Team XORP. 6-2-2010.XORP-CT is realese. XORP-users@XORP.org esta disponible en Internet en .http://mailman.icsi.berkeley.edu/pipermail/XORP-users/2010-June/003963.html

establecer la capacidad del enrutador, probando diferentes protocolos de enrutamiento, especialmente para BGP, estas pruebas son llamadas "*XORP test* harnesses".
Las pruebas diseñadas son posibles, porque XORP permite por medio de *scripts* generar llamados a los diferentes módulos de XORP, que es una interesante facultad que tienen los diferentes módulos.

La documentación de XORP-ct es generada por medio de un *script* ubicado en el directorio /usr/local/XORP/docs, ejecutando el comando "scons" y luego "scons install" tendremos la documentación principal de XORP, allí residen papers, manuales y presentaciones. En este directorio tendrá la documentación no actualizada para la versión 1.7, pero aun vigente, ya que los mayores cambios se produjeron en la forma de instalación, reducción del tamaño de la instalación y resolver algunos bugs de los protocolos presentes, pero el funcionamiento y conceptos de como la herramienta funciona no han cambiado

Anexo B. Instalación De XORP Con El CD-LIVE XORP-ct.

La instalación de UBUNTU a partir del CD-live de xorp.ct lleva los siguientes pasos

1. Para dar inicio a la instalación completa en el disco duro, debemos disponer de la clave de súper usuario, para el Cd-live de LANforge es lanforge.

Autorización del súper usuario Fuente: Autor

2. A Continuación, la selección de la zona horaria.

Zona horaria Fuente: Autor

3. Seguido, escogemos la mejor configuración del teclado

configuración teclado Fuente: Autor

4. Luego, pasamos a configurar la partición del disco, tenemos dos opciones, la primera es escoger una partición por defecto en la cual, el programa escoge la mejor opción dependiendo del tamaño del disco y de la memoria RAM de la maquina, la otra opción es en la que el usuario escoge, el tamaño de los diferentes grupos de archivos y la partición *swap*, además si en llegado caso se tienen dos particiones y una de ellas posee un Windows o otra distribución Linux, tendrá la opción de instalarla al lado. Como el objetivo de la instalación es tener un enrutador avanzado, la opción a escoger, es la que viene por defecto instalación, en todo el disco duro.

Partición del disco donde se va a realizar la instalación
Fuente: Autor

5. A continuación, se presenta una ventana, en la cual se necesita información sobre el usuario, el nombre del equipo y clave para acceder.

Información del usuario y del equipo Fuente: Autor

6. Seguido aparecerá una ventana de información, que describe, la configuración de la instalación que se va realizar.

Ventana de información de la instalación a realizar fuente: Autor

7. Luego, se le da click a *install* y nos mostrara una ventana de información del proceso de la instalación.

progreso de la instalación. *Fuente: Autor*

Como ultima parte del procedimiento de instalación, presenta un dialogo indicando, si quiere seguir probando el CD-live de Lanforge o reiniciar para empezar a usar la distribución desde el disco duro.

El Cd live-LANforge nos ofrece interactuar previamente con los módulos de XORP-ct, permitiendo familiarizarnos con los componentes de configuración del enrutador. El principio de la creación de este software sería didáctico pero con la posibilidad de habilitarlo para uso de producción y de investigación.

Anexo C. Comandos de Ayuda General Para Tener Una Instalación Exitosa

Los problemas más comunes durante la compilación son, la ausencia de una librería o la búsqueda de esta por el compilador en una ubicación errónea.
Aquí algunos comandos que pueden ayudar a dar solución a algunos errores de compilación.

Los enlaces simbólicos son útiles en el caso cuando se requiere un archivo y este no está presente en la ubicación deseada, y no se quiere tener una copia de este archivo en otra ubicación por que podría comprometer la seguridad sistema, el comando es el siguiente:
ln -s [archivo para enlace simbólico] [ubicación del enlace simbólico];

Con el comando “find” podrá solicitar la ubicación de un archivo, iniciado su búsqueda, desde el nodo de la carpeta donde se digito el comando hasta que termine el árbol de carpetas:
find -name [nombre del archivo a buscar];

Algunos diseños del archivo de compilación no aceptan enlaces simbólicos para

certificar la presencia del archivo y luego utilización por la aplicación, la única opción es crear una copia:
cp [archivo a copiar] [destino del archivo copiado];

Durante la compilación es habitual la creación de arboles completos de carpetas, pero durante algún error de compilación este proceso se puede interrumpir y estas carpetas que proceden de una compilación errónea podría generar más errores, sino se borran después de haber corregido el error. Para ello el comando rm, se utiliza, para hacer un borrado progresivo con las opciones -fr, hasta borrar todos los elementos de la carpeta objetivo:
rm -fr [carpeta de archivos a borrar];

También es recomendable hacer un *make clean* o un *scons clean*.
Una forma rápida de saber si un paquete ha sido instalado y desea saber su ubicación el comando which [nombre del paquete];

Cada comando presenta una documentación a la cual se puede acceder en consola, allí figura las opciones con las que puede ejecutar y las funciones del comando. Para solicitar esta ayuda, en consola digita man [comando del cual necesita información] o info[comando del cual necesita información] y para salir de la documentación con la tecla “q”.

ANEXO D. Script para crear las interfaces virtuales

```
#!/bin/bash
# tap-setup.sh

tap_up() {

#   tunctl   - t {INTERFACE-NAME}  -u {USUARIO-NAME}  ******

#                                          __TAP00__
tunctl -t tap00 -u root

ifconfig tap00 $tap00ip netmask 255.255.255.0 up

# ********************************************************
#                                          __TAP22__
tunctl -t tap22 -u root

ifconfig tap22 $tap22ip netmask 255.255.255.0 up
```

```
#  ********************************************************
#                                          __TAP01__
tunctl -t tap01 -u root

ifconfig tap01 $tap01ip netmask 255.255.255.0 up

echo "1" > /proc/sys/net/ipv4/ip_forward

ifconfig -a

          }
tap_down(){

tunctl -d tap00
tunctl -d tap01
tunctl -d tap22

ifconfig -a
                }

if [[ $EUID -ne 0 ]]; then

  echo "Debe correr este script como superusuario" 1>&2

  exit 1
else

case "$1" in

start)printf "digite la direccion ip de la interface tap00: "

     read tap00ip

     printf "digite la direccion ip de la interface tap22: "

     read tap22ip

     printf "digite la direccion ip de la interface tap01: "

     read tap01ip

     tap_up

         ;;
```

```
stop)
        tap_down
        ;;
*)
        echo "Usage: $0 {start|stop}"
        ;;
esac
fi
exit 0
```

ANEXO E. Codigo de ejecución para el protocolo RIP, llamado ripZ.boot

```
        /* $XORP$ */
interfaces {
    interface eth0_rename {
        vif eth0_rename {
            address 20.40.10.1 {
                prefix-length: 24
            }
        }
    }
    interface eth1 {
```

```
        vif eth1 {

            address 20.30.10.2 {

                prefix-length: 24

            }
        }
    }
}

fea {

    unicast-forwarding4 {
        disable: false
    }
}

policy {

    policy-statement export-connected {

        term 100 {
            from {
                protocol: "connected"
            }
        }
    }
}
protocols {
    rip {
        export: "export-connected"

        interface eth1 {
            vif eth1 {
                address 20.30.10.1 {
                    disable: false
/*
                    authentication {
                        simple-password: "password";
                            md5 @: u32 {
                                password: "password";
                            start-time: "2006-01-01.12:00"
                        end-time: "2007-01-01.12:00"
                        }
```

```
                }
*/
            }
        }
    }
  }
}
```

Anexo F. Codigo para ejecutar BGP, redistribuyendo una ruta estatica en BGP, con el nombre del archivo bgpZ.boot

```
/* $XORP$

interfaces {
    interface eth0 {
        vif eth0 {
            address 66.7.8.1 {
                prefix-length: 24
            }
        }
    }
  interface eth1 {
        vif eth1 {
            address 55.6.7.2 {
                prefix-length: 24
            }
```

```
        }
    }
}

policy {
    policy-statement  to-bgp {
  term include {
       from  {
         protocol: "static"

        /*    network4 == 66.7.8.0/24
              network4 != 55.6.7.0/24
              network4 == 0.0.0.0/8
           }
   to {
          neighbor: 0.0.0.0
       }
  then {
          reject*/
             }
          }
      }
 }
fea {
    unicast-forwarding4 {
          disable: false
    }
}
protocols {
    static {
      route 77.8.9.0/24 {
          next-hop: 66.7.8.2
            }
          }
        bgp {
     export: "to-bgp"

          bgp-id: 55.6.7.2
          local-as: 65000

          peer 66.7.8.2  {

              local-ip: 66.7.8.1
              as: 65000
             next-hop: 66.7.8.1
```

```
        }
    }
}
```

ANEXO G. Codigo de la VM "fedora_xorp0" para ejecutar OSPF con el nombre de ospf.boot

```
/* $XORP$ */

interfaces {

    interface eth0_rename {

        vif eth0_rename {

            address 20.30.10.1 {

                prefix-length: 24
            }
        }

 vif vlan1 {
    vlan {
```

```
            vlan-id: 01
              }
           address 20.30.10.4 {
                    prefix-length: 24
                    }
              }
           }

    interface eth1 {
           vif eth1 {
              address 20.40.10.1 {
                    prefix-length: 24
              }
           }
    }
  }
fea {

    unicast-forwarding4 {
           disable: false
    }
}
protocols {

    ospf4 {

           /*Policy para connected*/

            export: "export-connected"

            router-id: 20.0.0.0

         area 0.0.0.0 {

         interface eth0_rename {

          vif eth0_rename {

           address 20.30.10.1 {

                    }

                 }

            }
```

```
        }
    }
}
policy {
    policy-statement export-connected {
        term 100 {
            from {
                protocol: "connected"
            }
        }
    }
}
```

GLOSARIO

Archivo de extensión .boot: Archivos pertenecientes a la parte operativa de XORP, este archivo específicamente se detalla la configuración del enrutador.

Archivo de extensión .tp-template: Archivo encargado de describir como se debe configurar un proceso en particular, además, describe detalladamente las operaciones de comandos muy particulares una función que desempeña un nodo configurado que además esta ligado a una llamada xrl.

Archivo de extensión .tgt: Este archivo se definen los métodos foráneos declarados e implementados por archivos .xif de otros módulos, necesarios para definir un Módulo en particular.

Archivo de extensión .xif: Archivo donde se describe los métodos xrl, definidos con una sintaxis muy particular, para ser direccionales por solicitudes xrls. Este archivo representa una interface entre los métodos en programación de un Módulo y la posibilidad de acceso al usuario hacia estos.

Border Gateway Protocol-BGP: Es un protocolo de enrutamiento que pertenece a la *generic Exterior Gateway Protocols*

Enrutamiento Estático: Configurar una ruta manualmente.

Exterior Gateway Protocol: Es una clasificación que se le da a los protocolos de enrutamiento, que funcionan para enrutar entre Sistemas Autónomos.

Finder: Módulo de inter-comunicación de XORP, utilizado como un enrutador virtual de las solicitudes de comunicación entre los Módulos de XORP.

Interfaces XRL: Conjunto de Metodos con los parámetros necesarios para ejecutar una funcionalidad dada, que además son direccionales por llamadas xrl.

Interfaces de red TUN/TAP: Son Interfaces de Red creadas en software que residen en el nucleó de un sistema operativo, una vez creadas actúan como cualquier otra interface física.

IPC: Son métodos genéricos de comunicación entre dos o mas módulos que operan en diferentes niveles

Link-State Adver-LSA: Son mensajes generados por el protocolos de enrutamiento OSPF, son cinco mensajes cada mensajes es utilizado por la definición de una área, para enviar información entre enrutadores de una misma área o entre aéreas.

Link-State DataBase-LSD: Es uno de los mensajes que pertenecen al protocolo OSPF, que informa que rutas tiene la base de información de enrutamiento .

Live CD: Es un CD-room que es autoejecutable, para el proyecto XORP
es un software que permite ejecutar XORP sin necesidad de instalarlo en el disco duro.

netcat: Utilidad de red, con múltiples funciones entre estas comunicación socket TCP/UDP, muestreo de puertos, escritura y lectura de archivos.

Open Shortest Path First: Es un protocolo de enrutamiento IGP, basado en el algoritmo Link-state, utilizado en redes de media y alta escala.

Route Information Protocol-RIB : Es un protocolo de enrutamiento utilizado en pequeñas y mediana redes de escalabilidad.

Routing Daemons: Son “demonios” que se comunican entre sistemas para

propagar la tabla de enrutamiento del nucleó de un sistema operativo.

Sistemas Autónomos: Es un conglomerado de enrutadores mantenidos por empresas privadas, para mejor administración

softroute: Utiliza demonios de enrutamiento para crear, modificar la tabla de enrutamiento, con la premisa que el sistema base es por *software* y no por *hardware.*

VirtualBox: Aplicación que sirve para crear y manejar maquinas virtuales.

VLAN : Virtual LAN

xorp_rtrmgr: Proceso de XORP llamado *router manager,* encargado de inicializar los procesos configurados en archivo de ejecución de extensión .boot.

XRL: Es una forma de direccionar Metodos con una sintaxis parecida a la que existe para ubicar una página web, sólo que este ubica un método en un archivo de extensión .xif.

xrl target: Hace parte de las interface xrl, sólo que estos métodos, tienen como característica que solicitados desde un Módulo.

LISTA DE ANEXOS

Anexo F. Código para ejecutar BGP, redistribuyendo una ruta estática en BGP, con el nombre bgpZ.boot

Anexo G. Código de la VM "fedora_xorp0" para ejecutar OSPF con el nombre de ospf.boot

BIBLIOGRÁFIA

XORP.

Eddie Kohler.XORP: An Open Platform for Network Research.ICSI Center for Internet Research, Berkeley, California.2003.de Internet: research.microsoft.com/apps/pubs/default.aspx?id=73275.Fecha: 2010-10-6.

Pavlin Radoslavov, Designing Extensible IP Router Software, University College, London:UCLA,International Computer Science Institute, 2005.Internet: www.xorp.org/papers.html. Fecha: 2010-6-15.

XORP, Inc. XORP User Manual.version1.6.January 7, 2009. de Internet: www.xorp.org. Fecha: 2010-7-4.

XORP, Inc. XORP Router Manager Process (rtrmgr).version1.6.January 7, 2009.de Internet: www.xorp.org. Fecha: 2010-5-4.

XORP Inter-Process Communication Library Overview..version1.6.January 7, 2009.de Internet: www.xorp.org. Fecha: 2010-6-4.

XORP, Inc. XORP Forwarding Engine Abstraction.version1.6. January 7,2009. de Internet : www.xorp.org. Fecha: 2010-8-4

Mail List xorp-user.de internet: http://mailman.icsi.berkeley.edu/pipermail/xorp-users/. Fecha: 2010-5-2

Mail List xorp-hacker.de Internet: http://mailman.icsi.berkeley.edu/pipermail/xorp-hackers/. Fecha : 210-5-2

XORP wiki page.de Internet: http://xorp.run.montefiore.ulg.ac.be/start. Fecha: 2011-3-22.

XORP tesis.

Kristen Nielsen.Implementing NAT support into the XORP project. Marts 29, 2008. Department of Computer Science University of Copenhagen, Denmark
. De internet: http://www.krn.dk/thesis/xorp-nat-speciale-final-vers-1-1-20080329.pdf. Fecha: 2010-11-3

Nguyen Van Nam. Amélioration ET Implémentation D'un Algorithme D'évitement des Boucles Transitoires Durant LA Convergence D'ospf. Université catholique de Louvain, Institut de la Francophonie pour, IP Networking Lab (INL), Belgium

.De internet: http://www.ifi.vnu.edu.vn/site_data/ rapports/stages-promo12/stage-nguyen_van_nam.pdf. Fecha: 2010-12-5.

CLICK

Eddie Kohler.The Click Modular Router.submitted to the Department of Electrical Engineering and Computer Scienc in partial fulfillment of the requirements for the degree of Doctor of Philosophy .February 200. de internet: http://read.cs.ucla.edu/click/publications. Fecha: 2011-2-20.

Virtualización

Manual de Usuario Oracle VirtualBox.2004-2011 Oracle. De Internet: http://www.virtualbox.org/manual/UserManual.html. Fecha: 2010-6-4

Interface de red virtuales Tun/Tap. De Internet : http://vtun.sourceforge.net/tun/faq.html.

MASSIMO cafaro.Grids, Clouds and Virtualization
. Springer-Verlag London Limited 2011.De internet: http://www.springer.com/series/4198

Evaluación de plataformas de virtualización para experimentación
de servicios multimedia en redes IP.J. E. López de Vergara. Departamento de Ingeniería Informática, Universidad Autónoma de Madrid, Madrid, España. De Internethttp://biblioteca.espe.edu.ec/upload/Revista_WFuertes_JLopez_de_Vergara_Final.pdf. Fecha :2011-05-1.

Redes.

Guide to IP Layer Network Administration with Linux Version 0.4.4. 2003. de Internet : http://linux-ip.net/. Fecha: 2010-7-6.

Deepankar Medhi Y Karthikeyan Ramasamy.Network Routing Algorithms, Protocols, and Architectures. USA:Morgan Kaufmann Publishers, 2007.

KOZIEROK Charles M. The TCP/IP Guie. 2005. De internet:http://www.tcpipguide.com/. Fecha: 2010-7-9.

BURNS Bryan, Security power tools.USA: O'Reilly Media, Inc, 2007,pág 324

CONCLUSIONES Y RECOMENDACIONES

- XORP es una gran herramienta para la investigacion de redes en el campo de enrutamiento, gracias a su arquitectura Modular, permite extender procesos modificandolos o creando nuevos, esta caracteristica es la que lo hace sobre salir de los demas proyectos de enrutamiento avanzado ya sea de *software* libre o propietario. Pero creo que esa misma cualidad de extensibilidad, lleva inherentemente un grado de complejidad en la producción de un nuevo Módulo, para ser utilizada por un usuario con conocimientos medios de programación.

- En la emulación de una pequeña red para probar protocolos de enrutamiento de XORP, permitiendo familiarizase con las operaciones de configuración de XORP. Cuando llegue el momento de llevar a XORP a un ambiente de producción sera muy natural y rapido configurar protocolos de enrutamiento.

- XORP no emplea su propio Motor de envio de paquetes. sería interesante emplear a XORP para que trabajara conjuntamente con una NetFPGA; XORP proveeria la ruta y la NetFPGA se utilizaría para el plano de envio.

- XORP no tiene actualmente soporte para QoS, el integrarle al nucleo de Linux por medio de *scripts* soporte para QoS. Para que XORP opere con esta facultad que poseen los enrutadores modernos.

- En el presente trabajo se introdujo un pequeña práctica de CLICK para operaciones de tratamiento de paquetes en el plano de envio. Un proyecto donde se profundise sobre este *software* experimental, para utilizarlo con XORP o otro *software* de enrutamiento como Quaanga.

- Documentar a XORP en un ambiente de produccion sobre una plataforma embebida.

- Con la emulación de una red descrita en este trabajo se podria dar comienzo a un laboratorio de enrutamiento; Con cinco portatiles y cada uno con cinco instancias de maquinas virtuales como enrutadores, se puede emular el comportamiento de enrutadores para protocolos BGP y OSPF.

- Documentar la instalación y uso de XORP para la Nube. Una alternativa para la creación de un laboratorio de redes con un bajo costo.