Facultad de Ingeniería - Universidad de la República

Estudio de algoritmos de localización de Reflectores de Rutas en un Sistema Autónomo de Internet

Proyecto de Grado

Ingeniería en Computación

Gabriel Jambrina
Viviana Solla

Tutor: Eduardo Grampín

2016

Agradecimientos

Este proyecto representó un gran esfuerzo humano, no sólo para las personas que lo realizamos, sino también para nuestras familias y amigos. Por eso queremos agradecer a ellos, la paciencia, el apoyo, el cariño y las palabras de ánimo en los momentos difíciles.

Debemos agradecer especialmente a nuestro tutor Eduardo Grampín por su apoyo, paciencia y orientación a lo largo del proyecto. Por la ayuda que nos brindó para obtener las herramientas que precisábamos y por guiarnos en el proceso de investigación.

También queremos agradecer a los docentes Claudio Risso, Martin Giachino y Carlos Testuri por su cooperación en la investigación del algoritmo Optimal, el acceso al servidor de facultad y al software CPLEX.

Finalmente nos queda agradecer a los compañeros de trabajo, que pasan muchas horas del día con nosotros y nos han brindado su apoyo y cariño a lo largo de este proyecto.

Resumen

El protocolo BGP (*Border Gateway Protocol*) es utilizado hoy en día en todos los Sistemas Autónomos (AS) de Internet. Dentro de cada AS, las sesiones iBGP distribuyen las rutas externas entre los routers del AS. En ASes de gran tamaño, mantener un *full-mesh* (mallado completo) de sesiones entre los routers no es escalable, por lo que los reflectores de rutas (RR) son comúnmente utilizados en este tipo de ASes. Sin embargo, los reflectores de rutas adolecen de algunos problemas como robustez, retardos en la convergencia de las tablas de encaminamiento y reducción de opciones de encaminamiento, entre otros.
En los últimos años se ha comenzado a estudiar el problema de la localización de los RRs como un problema de optimización, donde el objetivo generalmente es minimizar la cantidad de sesiones, aunque también se han estudiado condiciones suficientes de corrección de una topología de RR, que evite los problemas mencionados anteriormente. En un proyecto anterior se han revisado las técnicas y algoritmos de localización existentes, y se ha construido una herramienta para implementar y probar algoritmos de localización. En este proyecto se construyó un ambiente de emulación que se asemeja a la realidad y se realizó una evaluación cuantitativa de los distintos algoritmos sobre una variedad de topologías. Se emularon topologías resultado de aplicar los algoritmos y se inyectaron mensajes de actualización durante una hora. Se midió la cantidad de mensajes generados, cantidad de sesiones y tiempo de convergencia, entre otros. Se concluyó que ninguno de los algoritmos implementados logra una disminución importante con respecto a la cantidad de mensajes trasmitidos. Incluso, la gran mayoría de los algoritmos aumentan considerablemente la cantidad de mensajes enviados provocando, de esta manera, un impacto negativo en la *performance* de el protocolo BGP.

Palabras clave: BGP, *Route Reflector*, *full-mesh*, Sistema Autónomo, Internet

Índice general

Índice de figuras **10**

Índice de tablas **13**

1. Introducción **15**
1.1. Motivación 17
1.2. Descripción del proyecto 19
1.3. Objetivos 20
1.4. Estructura del documento 21

2. Estado del Arte **23**
2.1. Marco Teórico 23
2.1.1. Sistemas Autónomos 24
2.1.1.1. Tipos de Sistema Autónomo 24
2.1.2. Direccionamiento IP 25
2.1.2.1. Clases de Direccionamiento IP 26
2.1.2.2. Enrutamiento entre dominios sin clase 27
2.1.3. Jerarquía de organizaciones encargadas del registro de Internet 28
2.1.4. Protocolos de Enrutamiento Interno y Protocolos de Enrutamiento Externo 30
2.1.5. Full-mesh iBGP 33
2.1.6. Confederaciones BGP 34
2.1.7. Reflexión de rutas 34
2.1.7.1. Comportamiento de un Reflector de Rutas 36
2.1.7.2. Evitar Loops en la información de ruteo 37
2.1.7.3. RR Redundantes 38
2.1.7.4. Impacto en el Proceso de Selección de Rutas . . . 40

2.1.7.5. Consideraciones en la implementación 40
2.1.7.6. Beneficios de los Reflectores de Rutas 40
2.1.7.7. Problemas que acarrean los Reflectores de Rutas . 42
2.2. Algoritmos existentes 47
2.2.1. BGPSep 48
2.2.2. BGPSep_D 51
2.2.3. BGPSep_S 53
2.2.4. Heurística Bates 57
2.2.5. Heurística Zhang 58
2.2.6. Algoritmo Optimal 59
2.3. Revisión de herramientas 67
2.3.1. Totem 67
2.3.2. RRLoc (xTotem) 68
2.3.3. Simuladores/Emuladores 69
2.3.3.1. C-BGP 69
2.3.3.2. MiniNet/MiniNExT 70
2.3.3.3. OMNeT++ 71
2.3.3.4. CORE 72
2.3.4. Inyectores de rutas BGP 72
2.3.4.1. exaBGP 72
2.3.4.2. BGPsimple 73

3. Diseño y Desarrollo **74**
3.1. Ambiente de Trabajo 74
3.2. Diseño e Implementación 79
3.2.1. Traducción gml a xml 82
3.2.2. Generar el ambiente de emulación 82
3.2.3. Base de Datos 84

4. Pruebas y Resultados **88**
4.1. Pruebas 88
4.1.1. Verificación de los resultados obtenidos 90
4.2. Cantidad de Sesiones iBGP 91
4.3. Cantidad de Reflectores de Rutas 94
4.4. Cantidad de mensajes BGP enviados 94
4.4.1. Tamaño promedio de la tabla rib_in 98
4.5. Tiempos de Propagación y promedio de anuncios repetidos 98
4.6. Tiempos de Convergencia 109
4.7. Emulación en servidor de la facultad 115

5. Ingeniería de Software **116**
5.1. Planificación 116
5.2. Problemas Encontrados 120
5.2.1. Problemas enfrentados durante el desarrollo 120
5.2.2. Problemas heredados 121
5.2.2.1. Problemas con el algoritmo Optimal 122
5.2.2.2. Problema con la heurística BatesY 124
5.2.2.3. Problema con la heurística BatesZ 125

6. Conclusiones y Trabajo a Futuro **126**
6.1. Conclusiones 126
6.2. Trabajo Futuro 130

A. Instalación del Ambiente de Trabajo **134**
A.1. Instalación RRLoc 134
A.2. Instalación ambiente de emulación 135
A.2.1. Instalación MiniNExT 135
A.2.2. Instalación Quagga 136
A.2.3. Instalación exabg 136
A.2.4. Instalación bgpdump 136

B. Manual de Usuario **138**
B.1. Posible caso de uso 144

C. Consultas realizadas a la Base **146**

Glosario **150**

Bibliografía **153**

Índice de figuras

1.1. Topología con *full-mesh* de sesiones BGP 18
1.2. Ejemplo de confederaciones 19
1.3. Topogolía con sesiones BGP y RR 20

2.1. Ejemplo de tipos de Sistemas Autónomos 25
2.2. Jerarquía de organizaciones encargadas del registro de Internet . . 29
2.3. Regional Internet Registry 30
2.4. Red sencilla con varios Sistemas Autónomos 31
2.5. Ejemplo de topología con configuración *full-mesh* 35
2.6. Ejemplo de topología con configuración RR 36
2.7. Sistema Autónomo con *cluster* 37
2.8. RR redundantes 38
2.9. Ejemplo de encaminamiento sub-óptimo 43
2.10. Ejemplo de generación de loops 44
2.11. Ejemplo de no determinismo 45
2.12. Configuración iBGP utilizando reflectores de rutas que tiene baja tolerancia a fallas IGP 46
2.13. Configuración iBGP incorrecta 46
2.14. Ejemplo de enrutamiento subóptimo: el tráfico enviado por z sigue el camino IGP (z, rr, x) en vez de (z, y) 61
2.15. Ejemplo de un grafo iBGP y su correspondiente grafo extendido. El camino válido (c_1, rr_1, rr_2, c_2) en G_{bgp} es mapeado a $(c1_{src}, rr1_{src}, rr2_{src}, rr2_{dst}, c2_{dst})$ en G^{ext}_{bgp} 64
2.16. Dos sesiones iBGP candidatas: $label(u, v) = UP$ o $label(u, v) = DOWN$ 65
2.17. Este corte mínimo de flujo máximo inserta la siguiente restricción al problema principal: $up(r_1, r) + down(r_1, r) + up(n, r) + down(n, r) \geq 1$ 66

3.1. Imagen de la topología Abilene vista desde xTotem 76
3.2. Proceso de Emulación 80
3.3. Comando para la obtención de la loc_rib_ipv4 85
3.4. Comando para la obtención de la loc_rib_ipv6 85

4.1. Cesnet Network 89
4.2. Cantidad de Sesiones iBGP (1) 93
4.3. Cantidad de Sesiones iBGP (2) 93
4.4. Cantidad de Sesiones iBGP (3) 94
4.5. Cantidad de mensajes transmitidos (1) 96
4.6. Cantidad de mensajes transmitidos (2) 96
4.7. Tamaño promedio RIB_IN de nodos no reflectores (1) 99
4.8. Tamaño promedio RIB_IN de nodos no reflectores (2) 99
4.9. Tamaño promedio RIB_IN dereflcetores de rutas (1) 100
4.10. Tamaño promedio RIB_IN de reflectores de rutas (2) 100
4.11. Consulta BD por prefijo 77.243.240.0/20 103
4.12. Resultado consulta a base por prefijo 77.243.240.0/20 y con AS-PATH=[65010 8607 6939 12389 42599] 104
4.13. Resultado de consulta a base por prefijo 77.243.240.0/20 [1/2] - Algoritmo BGPPSep 104
4.14. Resultado de consulta a base por prefijo 77.243.240.0/20 [2/2] - Algoritmo BGPPSep 105
4.15. Sesiones iBGP topología Abilene - Algoritmo BGPSep 105
4.16. Resultado de consulta a base por prefijo 77.243.240.0/20 router LOSA - Algoritmo BGPPSep 105
4.17. Consulta para el cálculo del tiempo de propagación - Algoritmo BGPPSep 106
4.18. Promedio de anuncios recibidos con el prefijo P por router - Algoritmo BGPPSep 106
4.19. Tiempo de propagación de un prefijo p (1) 107
4.20. Tiempo de propagación de un prefijo p (2) 107
4.21. Cantidad de anuncios repetidos que recibe un router en promedio (1) 109
4.22. Cantidad de anuncios repetidos que recibe un router en promedio (2) 110
4.23. Cantidad de anuncios repetidos que recibe un router en promedio (3) 110
4.24. Tiempo de convergencia en segundos (1) 112
4.25. Tiempo de convergencia en segundos (2) 113
4.26. Tiempo de convergencia en segundos (3) 114

5.1. Planificiación inicial del proyecto (1) 116

5.2. Planificiación inicial del proyecto (2) 117
5.3. Topología ISTAR luego de la aplicación del algoritmo Optimal. Los RRs están representados en azul. 122
5.4. Topología Forthnet luego de la aplicación del algoritmo Optimal. Los RRs están representados en azul. 124

6.1. Topología Forthnet 127

B.1. TTools 138
B.2. Salida del script “GMLtoXMLconverter.py” - Conexiónes eBGP . 139
B.3. Selección de archivo para convertir a formato adecuado 140
B.4. Ingresar número de Sistema Autónomo 140
B.5. Seleccionar routers de borde 141
B.6. Seleccionar ASN vecino eBGP 141
B.7. Cargar datos en la base 142
B.8. Selecciar trazas para descargar 143

Índice de tablas

3.1. Tabla comparativa de topología Abilene 86

4.1. Información sobre las topologías 90
4.2. Cantidad de sesiones iBGP por topología, por algoritmo - Parte 1 . 92
4.3. Cantidad de sesiones iBGP por topología, por algoritmo - Parte 2 . 92
4.4. Cantidad de Reflectores de Rutas por topología y algoritmo. 95
4.5. Cantidad de mensajes BGP enviados por topología, por algoritmo - Parte 1 97
4.6. Cantidad de mensajes BGP enviados por topología, por algoritmo - Parte 2 97
4.7. Promedio de la rib_in para el conjunto de Reflectores de Rutas - Parte 1 101
4.8. Promedio de la rib_in para el conjunto de Reflectores de Rutas - Parte 2 101
4.9. Promedio de la rib_in para el conjunto de NO Reflectores de Rutas - Parte 1 102
4.10. Promedio de la rib_in para el conjunto de NO Reflectores de Rutas - Parte 2 102
4.11. Tiempo de propagación (en segundos) para un prefijo - Parte 1 . . 108
4.12. Tiempo de propagación (en segundos) para un prefijo - Parte 2 . . 108
4.13. Promedio de updates por nodo para un prefijo - Parte 1 111
4.14. Promedio de updates por nodo para un prefijo - Parte 2 111
4.15. Tiempos de convergencia (en segundos) - Parte 1 113
4.16. Tiempos de convergencia (en segundos) - Parte 2 114

Capítulo 1

Introducción

Nadie puede discutir que Internet es la red que ha tenido mayor crecimiento a lo largo de estos últimos años. El acceso a esta red se ha convertido en un servicio esencial en muchos casos y que influye en gran medida sobre casi todos los ámbitos de la sociedad.
En la actualidad, existen diferentes tipos de dispositivos que se conectan a la red, desde *smartphones* hasta heladeras que se conectan con la página web del supermercado para pedir los suministros que hacen falta en el hogar.
Esto ha modificado el estilo de vida de las personas a tal punto que hoy en día uno puede realizar todo tipo de tareas desde la palma de su mano. Desde el pago de servicios, trámites con distintos entes gubernamentales, compras internacionales y una lista interminable de opciones; pero sin lugar a dudas el uso de la redes sociales en estos últimos años ha sido la explicación del gran crecimiento de esta red.

Internet comenzó siendo una idea de Joseph Carl Robnett Licklider con una visión temprana de una red de ordenadores mucho antes de que fuera construida. En algunos documentos, Licklider describía la idea de un conjunto de computadoras interconectadas entre sí para compartir información, al cual él le llamaba "Red Galactica" ("*Galactic Network*"). Licklider fue el primer director de la Agencia de Proyectos de Investigación Avanzados de Defensa más conocida como DARPA (*Defense Advanced Research Proyect Agency*). Dicha agencia fue la encargada de crear una red de computadoras con el fin de establecer un forma de comunicación entre las diferentes instituciones académicas y estatales, denominada ARPANET. A finales del año 1969 ARPANET comenzaba a tomar forma interconectando los primeros nodos de la red. Ese mismo año, debido a la necesidad de transmitir

información e ideas entre los distintos grupos de investigación que estaban trabajando en el desarrollo de ARPANET se creó un tipo de documento denominado RFC (*Request For Comments*), donde cada RFC describe algún componente de la red.
En 1970 se publica el primer protocolo de comunicación entre dos computadoras de ARPANET denominado NCP (*Network Control Protocol*), predecesor del protocolo TCP (Transport Control Protocol).
En marzo de 1971 se desarrolló el Protocolo de Transferencia de Archivos, FTP (*File Transfer Protocol*), que utilizaba el protocolo NCP como protocolo de comunicación. El protocolo FTP fue publicado por primera vez en el RFC 114. En Marzo de 1972, Ray Tomlinson escribió el software básico de envío y lectura de mensajes de correo electrónico, motivado por la necesidad de los desarrolladores de ARPANET de un mecanismo sencillo y veloz de comunicación.
En los inicios los RFCs se imprimían en papel y se distribuían a través del correo ordinario. Cuando se empezó a usar el protocolo FTP, los RFCs se preparaban como archivos en línea y se accedían utilizando este protocolo.
A media que más universidades se adherían a ARPANET y más crecía esta red, se empezó a notar las distintas limitaciones que tenía el protocolo NCP. Este protocolo no contaba con un método de control de paquetes ni con un método de detección de errores entre los mensajes que se intercambiaban las computadoras por la red. NCP se pensó de esta manera ya que ARPANET sería la red donde se iba a utilizar, y ésta sería tan fiable que no haría falta control de errores. Debido a esta limitaciones y otras más se decidió desarrollar un nuevo protocolo que se denominaría TCP/IP.
A principios de 1983 se sustituye el protocolo NCP por TCP/IP. En 1985 se suma la red de la Fundación Nacional para Ciencia de Estados Unidos (NFSNET, *National Fundation Science*) a ARPANET. Además, otras instituciones se empezaron a interesar en formar parte de la red ARPANET también. En 1990, debido a la gran demanda y el control de la integración de las distintas redes que se sumaban a ARPANET, se determina que se empleará el termino Internet para referirse a lo que hasta ese momento se llamaba ARPANET [1].

Internet ha cambiado desde la red unificada que era en sus inicios, en donde no existía la arquitectura TCP/IP que se utiliza actualmente y en donde cada router mantenía la información total de ruteo. Un evidente problema era el insostenible crecimiento de las tablas de ruteo a medida que la red crecía. Por estos motivos, se hizo inviable mantener a Internet como una única red. Esto provocó que Internet se convirtiera en una red levemente jerárquica compuesta por dominios

administrativos denominados Sistemas Autónomos (AS por sus siglas en inglés: *Autonomous System*).
Un AS es un grupo de redes de direcciones IP que son gestionadas por uno o más operadores de red que poseen una clara y única política de ruteo [2]; la Red Académica Uruguaya (RAU), ANTEL, Google, AT&T son ejemplos de Sistemas Autónomos. Un AS puede ser una pequeña empresa que maneje unos pocos routers en un solo edificio, una gran empresa que maneje decenas de routers en múltiples locaciones o hasta un Proveedor de Servicios de Internet (ISP) que administre cientos o incluso miles de routers.
Actualmente Internet se compone de más de 70000 Sistemas Autónomos diferentes, número que crece rápidamente [3].

El protocolo actualmente utilizado para intercambiar información entre dominios es BGP (*Border Gateway Protocol*). En BGP, sesiones BGP externas (eBGP) son establecidas para intercambiar rutas con ASes vecinos. Las rutas BGP son distribuidas dentro del AS por sesiones BGP internas (iBGP).
Una ruta BGP está compuesta por un prefijo, su *next-hop* (o siguiente salto), y un conjunto de atributos. Los atributos son utilizados en el proceso de decisión BGP (más adelante entraremos en detalle sobre este tema). El *next-hop* es la dirección de un router de borde del dominio. Este router es capaz de reenviar el tráfico hacia el destino al que pertenece el prefijo.
Inicialmente, los routers estaban habilitados a anunciar sólo, en sesiones iBGP, rutas que recibían de sesiones eBGP. En consecuencia, la distribución de las rutas BGP a todos los routers del AS requería configurar un *full-mesh* (mallado completo) de sesiones iBGP dentro del AS. Esto lleva a problemas de escalabilidad en ASes con cientos de routers. Hoy en día, la tendencia es a utilizar *Route Reflectors* (RR, reflectores de rutas) en ASes grandes. Un RR puede reenviar rutas aprendidas en algunas sesiones iBGP a otras sesiones iBGP. Así, permiten una reducción del número de sesiones iBGP establecidas en la red y el número de rutas mantenidas en los routers.

1.1. Motivación

Hemos visto el importante papel que juega BGP en Internet hoy en día ya que permite anunciar a cada subred su existencia al resto de Internet. De esta forma, cuando una subred se conecta a Internet, BGP permite que todos los Sistemas Autónomos de Internet sepan que la subred existe y cómo llegar a ella. Si no fuera por BGP, las subredes estarían aisladas, resultando desconocidas para el resto de Internet. Sin embargo, este protocolo fue por primera vez publicado en el RFC

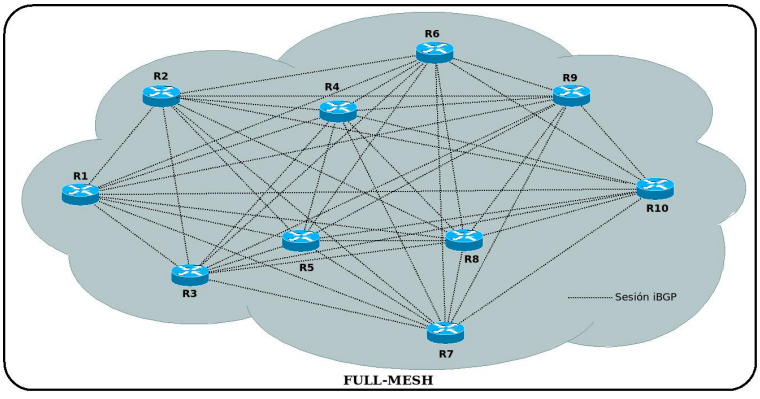


Figura 1.1: Topología con *full-mesh* de sesiones BGP

1105[4] en el año 1989, donde tanto la cantidad de tráfico como la infraestructura de Internet era más pequeña. La pregunta que nos hacemos es: ¿Qué tan bien se adecúa BGP al Internet actual?

El diseño original de BGP requiere que todos los routers que implementan el protocolo intra-dominio (iBGP) mantengan un *full-mesh* (mallado completo) de sesiones, y que los anuncios de BGP solo se propaguen a los vecinos directamente conectados (ver imagen 1.1). Esto crea un problema de escalabilidad de sesiones de $O(n^2)$ (para un AS con n routers BGP, se requieren $n * (n-1)/2$ sesiones), escalabilidad de configuración (las sesiones se deben configurar en todos los routers), escalabilidad de mensajes, y escalabilidad de estado (debido al mecanismo de BGP, cada router debe mantener una copia de la tabla de encaminamiento de cada vecino BGP).

Para paliar este problema la comunidad de Internet propuso dos soluciones:

- *BGP Confederations* (Confederaciones BGP).
- *Route Reflection* (Reflexión de Rutas).

La primera de ellas básicamente consiste en subdividir un AS en pequeños sub-ASes llamados confederaciones BGP, en donde en cada AS se configura un mallado completo (*full-mesh*) de sesiones iBGP (imagen 1.2). Esto se logra asociando un número de Sistema Autónomo (ASN - *Autonomous System Number*) privado (entre 64512 y 65534 para un ASN 16 bits), que sólo es válido en el AS al que pertenece.

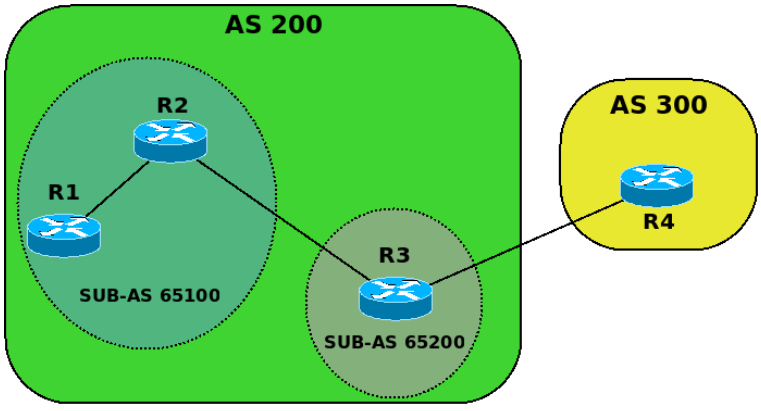


Figura 1.2: Ejemplo de confederaciones

Por otro lado, los reflectores de rutas permiten construir jerarquías en el BGP intra-dominio, relajando la necesidad del *full-mesh* de sesiones (imagen 1.3). Los RRs se han venido implementando exitosamente desde hace dos décadas en Internet, aunque adolecen de algunos problemas como robustez, retardos en la convergencia de las tablas de encaminamiento, reducción de opciones de encaminamiento, encaminamiento sub-óptimo, posibles *loops* y oscilaciones en el reenvío (*forwarding*) de paquetes. En el capítulo de Estado del Arte se verán en detalle estos problemas.

Si bien ambas soluciones tienen enfoques distintos, tanto *BGP Confederations* como *Route Reflection* implementan jerarquías en el funcionamiento de BGP intra-dominio para solventar el problema de escalabilidad.

1.2. Descripción del proyecto

Este proyecto analiza y compara el resultado de distintos algoritmos que determinan la localización de reflectores de rutas dentro de un Sistema Autónomo. En particular, el estudio se enfoca es Sistemas Autónomos de gran escala donde tiene sentido la implementación de reflectores de rutas.

La primera parte del proyecto consta en un estudio general de los reflectores de rutas, donde se describen el comportamiento, los problemas y los beneficios de los mismos. También se hace una comparación de aspectos generales de la

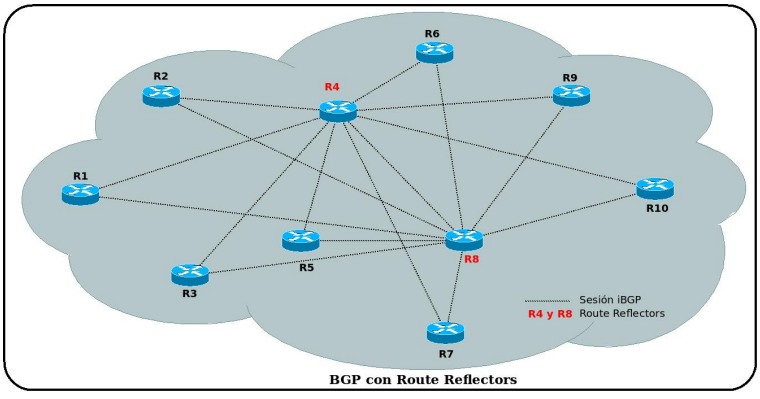


Figura 1.3: Topogolía con sesiones BGP y RR

implementación con *full-mesh* y con la de RRs. En esta sección también haremos un estudio del estado del arte, donde se investigará los proyectos relacionados con esta temática y se estudiarán los algoritmos a utilizar.

Para llevar a cabo esta investigación es necesario un ambiente de trabajo que se asemeje a la realidad para así poder obtener resultados más certeros. Como segunda parte del proyecto, se estudiarán distintas herramientas para determinar cuáles se adaptan mejor a nuestros requerimientos.

Con el objetivo de poder llevar a delante dicho análisis con diferentes algoritmos de localización en distintas topologías dentro del tiempo estipulado para un proyecto de esta índole, se desarrollarán un conjunto de herramientas que automatizan todo el proceso.

1.3. Objetivos

En este proyecto nos centraremos en estudiar una de las soluciones mencionadas en la Motivación, la denominada *Route Reflection*. En un proyecto de grado anterior, se han revisado las técnicas y algoritmos de localización existentes y se ha construido una herramienta para implementar y probar algoritmos de localización de reflectores de rutas (**"RRLOC: a tool for iBGP Route Reflector Topology planning and experimentation"**[10]) que puede ser utilizada para planificación de la red de un operador.
En el documento que habla del RRLOC queda claro que la utilización de los RRs

en las redes de gran tamaño disminuye considerablemente las sesiones iBGP, pero no se ve tan claramente cómo afecta otros factores como son la cantidad de mensajes que se envían en las sesiones iBGP, ni tampoco cómo afecta la velocidad a la que se distribuyen las rutas dentro de un AS. Algo que también queda claro es que la localización de los RRs es de suma importancia a la hora de obtener una mejor *performance*.
Teniendo en cuenta todo lo descrito anteriormente, este proyecto tiene como objetivos realizar una evaluación cuantitativa de distintos algoritmos sobre una variedad de topologías y, en particular, se plantea evaluar aspectos de la mensajería BGP generada para lo cual se deben utilizar emuladores de red.

Uno de los principales objetivos es estudiar la cantidad de mensajes transmitidos para las distintas configuraciones BGP, pero ¿porqué es importante estudiar esto? Cuando decimos mensajes transmitidos estamos hablando también de mensajes procesados por los nodos, por lo tanto cuanto mayor sea este número mayor es la cantidad de mensajes que un router procesa. Esto genera un mayor uso de recursos por lo que podría llega a tener un impacto muy grande si los routers recibieran más mensajes de los que pueden procesar. Esto se vería reflejado en la *performance* de BGP.

1.4. Estructura del documento

El documento se divide en 6 capítulos. En el capítulo 1 se tiene la Introducción donde se plantea y define el problema, se motiva el trabajo, se dejan claros cuáles son los objetivos planteados y se describe la organización general del documento.

En el capítulo 2 se realiza una revisión del Estado del Arte. Se realiza una introducción al marco teórico necesario para entender el proyecto. Se especifican los algoritmos utilizados y se realiza una revisión de las posibles herramientas a utilizar.

En el capítulo 3 se definen las herramientas a utilizar para realizar las emulaciones. Además, se diseña y desarrolla un *framework* que facilitará la ejecución de las tareas previas a las emulaciones.

El capítulo 4 contiene las Pruebas y Resultados. Aquí se definen todas las pruebas realizadas y se muestran todos los resultados obtenidos.

En el capítulo 5 se comenta la planificación realizada al inicio del proyecto y cómo fue variando a lo largo del desarrollo del mismo. Se enumeran los problemas encontrados y, en caso de haber encontrado solución, se explica cómo se resolvió el problema.

Finalmente, en el capítulo 6 se encuentran las conclusiones del proyecto. Además, se comentan los posibles trabajos futuros vinculados a este proyecto.

Al final del documento se pueden encontrar los anexos. Estos contienen información relacionada a la instalación del ambiente de emulación utilizado para realizar las pruebas, un manual de usuario del *framework* desarrollado y las consultas realizadas a la base de datos.

Capítulo 2

Estado del Arte

En este capítulo presentamos un marco teórico con los conceptos más importantes para entender el proyecto, describiremos en detalle los algoritmos existentes para la localización de los Reflectores de Rutas y haremos un estudio de las herramientas disponibles para el desarrollo del mismo, mencionando sus ventajas y desventajas. También presentaremos un estudio de la herramienta que utilizamos para poder ejecutar los algoritmos: xTotem/RRLoc.

2.1. Marco Teórico

Internet esta compuesta por decenas de miles de diferentes redes llamadas Sistemas Autónomos (ASes, por sus siglas en inglés: *Autonomous Systems*). Cada AS representa una entidad administrativa individual con su número de AS único y sus bloques de direcciones IP llamadas prefijos. Routers de diferentes ASes establecen sesiones BGP (*Border Gateway Protocol*) entre ellos para intercambiar actualizaciones de enrutamiento BGP (enrutamiento entre dominios). Este tipo de sesiones son llamadas sesiones eBGP (*external BGP*). También se establecen sesiones entre los routers de un mismo AS (enrutamiento intra dominio) para intercambiar información de enrutamiento. Estas sesiones son conocidas como sesiones iBGP (*internal BGP*).

2.1.1. Sistemas Autónomos

Hasta el año 2007 los números de Sistemas Autónomos estaban definidos por un número entero de 16 bits, lo que permitía un máximo de 65536 asignaciones de Sistemas Autónomos. Debido a la gran demanda, se hizo necesario aumentar el espacio de números de ASes disponibles. Actualmente el número de Sistema Autónomo (ASN, por sus siglas en inglés: *Autonomous System Number*) es una campo de 32 bits, con 4.294.967.296 posibles valores. De este bloque, la IANA (*Internet Assigned Numbers Authority*) tiene reservado para uso privado un bloque de 1023 números contiguos (desde 64512 hasta 65534 inclusive) [5], que están dentro del bloque de números de Sistema Autónomo de 16 bits. La IANA también tiene bloques reservados para uso privado en el rango de números que pertenecen a los Sistemas Autónomos de 32 bits. Estos bloques se detallan en [6].

Los números de Sistemas Autónomos son asignados en bloques por la *Internet Assigned Numbers Authority*. La IANA es un departamento de la ICANN (*Internet Corporation for Assigned Names and Numbers*) responsable de la asignación global de direcciones IP y número de Sistemas Autónomos, administración de los servidores raíz del Sistema de Nombres de Dominio (DNS, por sus siglas en inglés: *Domain Name System*) y otros recursos relativos a los protocolos de Internet. [7]

2.1.1.1. Tipos de Sistema Autónomo

Los Sistemas Autónomos pueden agruparse en tres categorías, dependiendo de sus conexiones y modo de operación como se puede apreciar en la figura 2.1:

- *Stub AS*: Es un AS que se conecta únicamente con un Sistema Autónomo, es decir que tiene un único punto de salida. Esto, en general, es una red corporativa pequeña.

- *Multihomed AS*: Es un AS que se conecta con varios Sistemas Autónomos, pero no soporta el tráfico de tránsito entre ellos. Este es el caso de las redes de grandes empresas.

- *Transit AS*: Es un AS que se conecta con varios Sistemas Autónomos y además permite que se comuniquen entre ellos. La mayoría de los grandes ISP (*Internet Service Provider* o Proveedor de Servicios de Internet) son *Transit AS*.

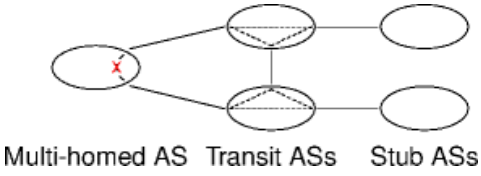


Figura 2.1: Ejemplo de tipos de Sistemas Autónomos

2.1.2. Direccionamiento IP

Actualmente hay dos tipos de direcciones IP: IP versión 4 (IPv4) e IP versión 6 (IPv6). Las direcciones IPv4 son campos de 32 bits. Esto limita el espacio de direcciones IPv4 a un total de 4.294.967.296. Este número parecía lo suficientemente grande cuando se diseñó, pero el crecimiento exponencial de Internet, junto a la falta de coordinación para su asignación durante la década de los 80 sin ningún tipo de optimización, han generado el agotamiento de direcciones IPv4, entre otros problemas. Como consecuencia, se han desarrollado distintas tecnologías que permiten paliar este problema, entre ellas las direcciones IPv6. Las direcciones IPv6 incrementan el tamaño del campo de 32 bits a 128 bits logrando un espacio de direcciones mucho mayor que el de IPv4. El diseño del protocolo agrega múltiples beneficios en seguridad, manejo de calidad de servicio, una mayor capacidad de transmisión y mejora la facilidad de administración, entre otras cosas.

La mayoría del tráfico IP es realizado con direccionamiento IPv4, y aunque se pretende que IPv6 reemplace a IPv4 en un futuro, ambos protocolos coexistirán durante algún tiempo.

Las direcciones IPv4 se pueden clasificar en tres tipos:

- Direcciones IP públicas: Las direcciones IP públicas son aquellas que permiten que cada dispositivo conectado a Internet sea visible para cualquier dispositivo de la red. Cuando accedemos a Internet desde nuestra computadora obtenemos una dirección IP pública suministrada por nuestro proveedor de servicio de Internet. Esta dirección IP es única y permite identificar al dispositivo en la red.
  Cuando un dispositivo se conecta a un servidor de Internet, la dirección IP del servidor será una dirección IP pública. El servidor utiliza la dirección IP pública del usuario para saber dónde enviar la información de vuelta.

- Direcciones IP privadas: Algunos rangos de direcciones IPv4 han sido reser-

vados para la operación de redes privadas. Cualquier organización puede usar estas direcciones IPv4 en sus redes privadas sin la necesidad de solicitarlo a algún Registro de Internet. Estas direcciones, a diferencia de las direcciones IP públicas, no son alcanzables desde Internet. La IANA ha reservado los siguientes tres bloques de direcciones IPv4 para el uso en redes privadas: [8]

  - 10.0.0.0 - 10.255.255.255 (prefijo 10/8)
  - 172.16.0.0 - 172.31.255.255 (prefijo 172.16/12)
  - 192.168.0.0 - 192.168.255.255 (prefijo 192.168/16)

- Direcciones IP especiales y reservadas: Algunas direcciones IPv4 son reservadas para aplicaciones como el *multicasting*. Por más información, ver [9].

2.1.2.1. Clases de Direccionamiento IP

En los inicios de Internet la parte de red de una dirección IPv4 estaba restringida a longitudes de 8, 16 o 24 bits, un sistema de direccionamiento conocido como Direccionamiento con Clases, ya que las subredes con direcciones de 8, 16 y 24 bits se conocían, respectivamente, como redes de clase A, B y C.

- Clase A: Esta clase define las redes de gran tamaño, por ejemplo las de un ISP. En redes de clase A, el valor del primer bit en el primer octeto es siempre 0 y la mascara de red es 255.0.0.0. Esto hace que el espacio de direcciones que define la Clase A incluya las direcciones a partir del 0.0.0.0 hasta la 127.255.255.255. Por lo tanto, hay un total de 128 redes clase A. El rango de IPs 127.X.X.X se reservan para las direcciones IP de *loopback*.

- Clase B: La clase B se utiliza para definir las redes de tamaño mediano. Las direcciones IP comprendidas entre el rango 128.0.0.0 y 191.255.255.255 son parte de esta clase. La máscara de red que definen a éstos rangos es la 255.255.0.0.

- Clase C: Las direcciones de la clase C se utilizan para redes pequeñas y medianas. En redes de clase C, el valor de los tres primeros bits son siempre "110" y la mascara de red es 255.255.255.0. De esta forma, las direcciones IP clase C comprenden el rango de direcciones desde 192.0.0.0 hasta 223.255.255.255. Para identificar a los dispositivos dentro de una red clase C se utiliza el último octeto. Esto significa que hay 2^{21} redes de la clase C con 254 ($2^8 - 2$) dispositivos posibles cada uno, para un total de 2^{29} direcciones IP únicas.

- Clase D: Las direcciones de clase D son direcciones de *multicast*. El *multicast* consiste en que un datagrama sea entregado a varios hosts dentro de la red en lugar de todos (*broadcast*) o uno sólo (*unicast*). Se puede decir que una dirección de clase D o *multicast* identifica un grupo de hosts dentro de la red. Lo que caracteriza a las direcciones clase D es que sus cuatro primeros bits tomarán siempre el valor binario 1110, es decir, el primer octeto tendrá un valor decimal entre 224 y 239. Además, las direcciones clase D utilizan únicamente el primer octeto como identificativo de red y los tres octetos restantes se utilizan como identificativo de grupo de hosts.

- Clase E: Las direcciones de clase E están reservadas para uso experimental en proyectos de investigación en la red. Lo que caracteriza a las direcciones clase E es que sus cuatro primeros bits (los que se encuentran más a la izquierda) tomarán siempre el valor binario 1111, es decir, el primer octeto tendrá un valor decimal entre 240 y 255.

Como se puede apreciar, dentro de cada rango de clases A, B y C, existen bloques de direcciones para uso de redes privadas.

2.1.2.2. Enrutamiento entre dominios sin clase

El requisito de que la parte de subred de una dirección IP tuviera exactamente una longitud de 1, 2 o 3 bytes se volvió problemático a la hora de dar soporte al rápido crecimiento de número de organizaciones con subredes de tamaño medio y pequeño. Una subred con clase C sólo puede contener hasta 254 hosts, que son muy pocos para muchas organizaciones. Sin embargo, una subred de clase B, que puede soportar hasta 65634 hosts, puede ser demasiado grande. Con el direccionamiento con clases, una organización con, por ejemplo, 2000 hosts, era asignada normalmente a una dirección de subred de clase B. Esto llevó a un rápido agotamiento del espacio de direcciones de clase B y una pobre utilización del espacio de direcciones asignadas [10].

La estrategia de asignación de direcciones utilizada en Internet hoy en día se conoce como enrutamiento entre dominios sin clase (CIDR, *Classless Interdomain Routing*) [11]. CIDR sustituye la sintaxis previa para nombrar direcciones IP, las clases de redes (explicadas en el punto anterior). En vez de asignar bloques de direcciones en los límites de los octetos, CIDR utiliza la técnica VLSM (*Variable Length Subnet Mask*) que como indica su nombre, permite utilizar máscaras de subred de longitud variable. De esta forma, CIDR generaliza la noción de direccionamiento de subred.

Para facilitar el ruteo, CIDR permite agrupar bloques de direcciones en una sola entrada de la tabla de rutas. Estos grupos, llamados comúnmente Bloques CIDR, comparten una misma secuencia inicial de bits en la representación binaria de sus direcciones IP.
Los bloques CIDR IPv4 se expresan en notación decimal con puntos como a.b.c.d/x, donde x indica el número de bits de la primera parte de la dirección. Los x bits más significativos de una dirección en el formato a.b.c.d/x constituyen la parte de red de la dirección IP y a menudo se hace referencia a ellos como el prefijo (o prefijo de red).
CIDR también se utiliza con direcciones IPv6, en las que la longitud del prefijo varia desde 0 a 128, debido a la longitud de las direcciones. La sintaxis utilizada en IPv6 es similar a la utilizada en IPv4. El prefijo se escribe como una dirección IPv6, seguida de una barra y el número de bits significativo.

2.1.3. Jerarquía de organizaciones encargadas del registro de Internet

La responsabilidad de la administración tanto del espacio de direcciones de IPv4 e IPv6, como el espacio de números de Sistema Autónomo está distribuida globalmente de acuerdo con la estructura jerárquica que se muestra a en la figura 2.2.

La parte más alta de este sistema hace referencia al *Internet Assigned Numbers Authority* quien delega los recursos a los Registros Regionales (RIRs), que a su vez tienen sus políticas para delegar recursos (IPs, ASNs) a sus clientes. Éstos últimos, entre otros incluyen ISPs y usuarios finales.

Para poder entender un poco mejor cómo funciona esta estructura jerárquica, veamos las siguientes definiciones: [12]

Internet Registry (IR) Un *Internet Registry* (IR) es una organización responsable de la distribución de espacios de direcciones IP a sus miembros o clientes, y del registro de esa distribución. Los IRs están clasificados de acuerdo a su función principal y alcance territorial dentro de la estructura jerárquica.

Regional Internet Registry (RIR) Los *Regional Internet Registries* (RIRs) son establecidos y autorizados por las comunidades regionales respectivas, y reconocidos por el IANA para servir y representar grandes regiones geográficas. El rol principal de los RIRs es administrar y distribuir el espacio de direcciones público

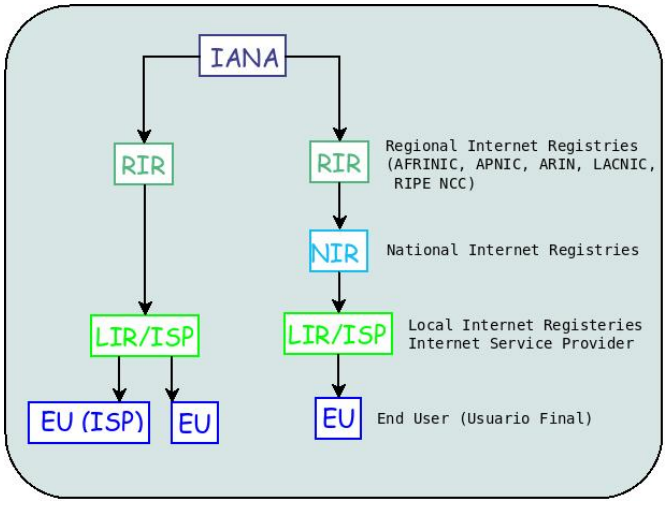


Figura 2.2: Jerarquía de organizaciones encargadas del registro de Internet

de Internet dentro de las respectivas regiones. En la figura 2.3 se puede apreciar la región que abarca cada RIR. [13]

National Internet Registry (NIR) Un *National Internet Registry* (NIR) adjudica, principalmente, espacios de direcciones a sus miembros, los cuales son generalmente LIRs a un nivel nacional. Los NIRs existen mayormente en la región de Asia Pacífico.
Adjudicar significa distribuir el espacio de direcciones a los IRs con el propósito de que ellos realicen la subsiguente distribución.

Local Internet Registry (LIR) Un *Local Intenet Registry* (LIR) es un IR que asigna, principalmente, espacios de direcciones a los usuarios de los servicios de red que éste provee. Los LIRs son generalmente ISPs, cuyos clientes son principalmente usuarios finales y posiblemente otros ISPs.
Asignar significa delegar espacio de direcciones a un ISP o usuario final, para su uso específico dentro de la infraestructura de Internet que ellos operan. Las asignaciones deben ser realizadas solamente para los propósitos específicos documentados por organizaciones específicas y no para ser subasignadas a otras partes.

Registry	Area Covered
AFRINIC	Africa Region
APNIC	Asia/Pacific Region
ARIN	Canada, USA, and some Caribbean Islands
LACNIC	Latin America and some Caribbean Islands
RIPE NCC	Europe, the Middle East, and Central Asia

Figura 2.3: Regional Internet Registry

2.1.4. Protocolos de Enrutamiento Interno y Protocolos de Enrutamiento Externo

A los protocolos que corren dentro un Sistema Autónomo se les denominan protocolos IGP (*Interior Gateway Protocol*). Ejemplos de estos protocolos son RIP (*Routing Information Protocol*) y OSPF (*Open Shortest Path First*). A los protocolos que son utilizados para intercambiar información de enrutamiento entre Sistemas Autónomos se les denomina EGP (*Exterior Gateway Protocol*). La combinación de estos dos tipos de protocolos permite enrutar tanto destinos internos al AS como destinos externos.

Dentro de los protocolos EGP se encuentra BGP. Este protocolo juega un papel crítico en las comunicaciones en Internet, permitiendo en cada Sistema Autónomo:

- Obtener información sobre qué destinos son alcanzables a través de los Sistemas Autónomos vecinos.
- Propagar la información aprendida por los routers de borde a todos los routers internos del Sistema Autónomo.
- Determinar "buenas" rutas a las distintas subredes (no necesariamente determina las mejores rutas).

BGP está sin duda alguna dentro de los protocolos más complejos de Internet, existen libros enteros dedicados exclusivamente a este protocolo, por lo que se hace muy difícil un total dominio del mismo. Básicamente, BGP se comporta como un protocolo vector distancia con la diferencia que, en lugar de mantener el costo hacia cada destino, cada router mantiene el camino utilizado. Por este motivo, algunos libros lo clasifican como un protocolo "vector de rutas". Del mismo modo, cuando

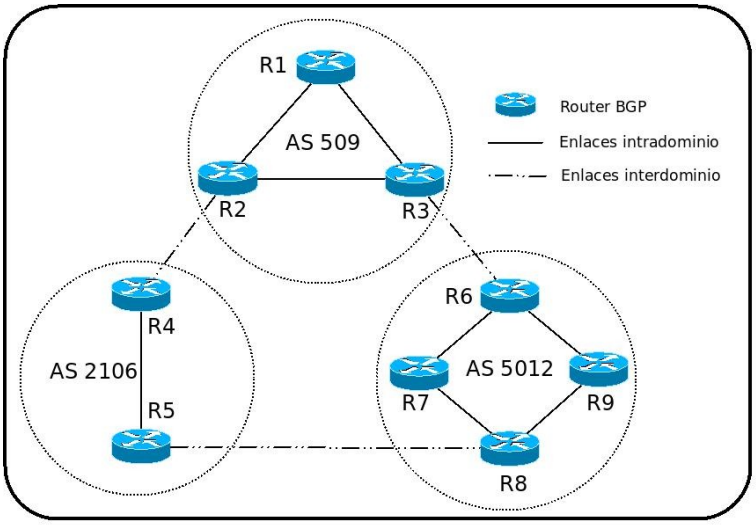


Figura 2.4: Red sencilla con varios Sistemas Autónomos

los routers se envían información de accesibilidad a un determinado destino, ésta contiene la ruta necesaria para llegar al mismo.

BGP utiliza TCP (*Transmission Control Protocol*) como protocolo de transporte en el puerto 179. Los routers que se encuentran en los extremos de la conexión se denominan pares BGP (*peers*), y la conexión TCP con todos los mensajes BGP enviados a través de la misma se denomina sesión BGP. Como se explicó antes, se deben distinguir entre dos tipos de sesiones BGP. Las sesiones que se establecen entre routers de borde de diferentes Sistemas Autónomos se llaman sesiones eBGP, y las sesiones que se establecen entre los routers internos del Sistema Autónomo se denominan sesiones iBGP.

Los routers BGP intercambian información de accesibilidad de la red. Esta información se representa mediante prefijos CIDR, representando cada prefijo una subred o una colección de subredes. Cuando un router anuncia un prefijo, junto con él, incluye una serie de atributos los cuales posteriormente serán utilizados para la elección de las rutas. Estos atributos son:

- AS-PATH (Camino de Sistemas Autónomos): El AS-PATH almacena la serie de Sistemas Autónomos a través de los que ha pasado el anuncio de un prefijo. Cada vez que un router de borde propaga una ruta mediante una

sesión eBGP, éste agrega su ASN al atributo AS-PATH. Esta lista no se modifica si se usa una sesión iBGP. Los routers utilizan el AS-PATH para impedir bucles de anuncios, esto quiere decir que si un router ve su ASN en el atributo AS-PATH rechaza el anuncio. Este atributo también se utiliza para seleccionar entre varias rutas a un mismo destino.

- ORIGIN (Origen): El atributo ORIGIN contiene el mecanismo por el cual se aprendió el prefijo en una primer instancia. Puede tomar los valores IGP, EGP o INCOMPLETE. Como su nombre lo indica, IGP informa que el prefijo ha sido aprendido utilizando un protocolo interno al Sistema Autónomo. De la misma manera, el valor EGP indica que el prefijo se aprendió utilizando un protocolo externo y el valor INCOMPLETE indica que se ha aprendido de otra manera (generalmente indica que es de forma estática). A la hora de la elección de una ruta, se selecciona la que tiene el valor IGP en atributo ORIGIN antes que las de EGP, y éstas últimas antes que las que tienen el valor INCOMPLETE.

- NEXT-HOP (Siguiente salto): El atributo NEXT-HOP contiene la dirección IP de la interfaz del router que debe ser el siguiente salto para llegar al destino. O sea, es la dirección IP de la interfaz del router del Sistema Autónomo vecino por el cual se ha aprendido la ruta a un determinado prefijo. Cuando le llega un anuncio a un router, éste utiliza el protocolo IGP para obtener el mejor camino hacia el NEXT-HOP, según la métrica del protocolo interno. La elección de una ruta por este atributo se basa en el que produzca el menor costo hacia el NEXT-HOP, o sea menor número de saltos.

- LOCAL-PREF (Preferencia local): La preferencia local nos permite indicarle al Sistema Autónomo el trayecto que tiene preferencia para salir del AS con el fin de alcanzar una determinada red. Si hay múltiples puntos de salida del AS, este atributo es usado para seleccionar la salida de una ruta específica. Se prefiere el trayecto con mayor preferencia local. Este atributo es opcional y no transitivo, no se propaga a través de sesiones eBGP pero si se propaga dentro del Sistema Autónomo.

- MED (Métrica): El atributo *Multi-exit discriminator* (MED) o también llamado de una forma más simple como "métrica", es usado como una sugerencia a un Sistema Autónomo externo sobre la ruta preferida al AS que está anunciando la métrica. El término "sugerencia" es utilizado porque el Sistema Autónomo externo que está recibiendo este atributo podría estar usando otros atributos BGP para la selección de rutas. El atributo MED se

envía únicamente a vecinos eBGP, pero no se propagará a ASes terceros. Es un atributo opcional no transitivo con preferencia el menor valor.

El proceso de decisión de BGP para seleccionar la mejor ruta involucra a cada uno de estos atributos. Para nuestro estudio, con el fin de simplificar el comportamiento del proceso de decisión, no vamos a configurar los atributos LOCAL-PREF y MED. Esto es posible ya que son atributos opcionales y por lo tanto no es obligatorio configurarlo en los equipos. Sin embargo, cabe aclarar que estos atributos pueden ser agregados para un futuro estudio.
BGP realiza una ponderación de los atributos de manera que si se tienen varias rutas a un mismo prefijo solo se elija una según un orden. A continuación se muestra el proceso de selección de rutas para un mismo prefijo:

1. Si el siguiente NEXT-HOP no es accesible se descarta la ruta.
2. Se prefieren las rutas con mayor LOCAL-PREF.
3. Se prefieren las rutas con AS-PATH más corto.
4. Se prefieren las rutas con ORIGIN más bajo.
5. Se prefieren las rutas con menor MED.
6. Se prefieren las rutas aprendidas por eBGP antes que las aprendidas por iBGP.
7. Se prefieren las rutas con menor costo hacia el NEXT-HOP.
8. Se prefieren las rutas que ha anunciado el router con menor identificador BGP.
9. Se prefieren las rutas recibidas del vecino con menor dirección IP.

2.1.5. Full-mesh iBGP

Como una forma sencilla de evitar los *routing loops*, el diseño BGP original requería que todos los routers del mismo AS estuvieran conectados en un *full-mesh*, y que la información aprendida desde una sesión iBGP no podía ser reenviada por ningún router. Sin embargo, esta configuración requiere de un total de sesiones iBGP que crece hasta el cuadrado del número de routers BGP dentro del AS. Además, dado que las sesiones iBGP son administradas a través de configuraciones manuales, este requerimiento de un *full-mesh* de sesiones implica cambiar la

configuración de cada uno de los routers del AS en caso de la adición o remoción de un router. Para mitigar este problema de escalabilidad, se propusieron dos arquitecturas alternativas ampliamente utilizadas por grandes ISPs: Reflexión de Rutas y Confederaciones BGP.

2.1.6. Confederaciones BGP

Confederaciones BGP [14] toma el modelo de *"Divide-and-Conquer"* para mitigar el problema de escalabilidad de las sesiones iBGP. Como se explicó antes, esta solución consiste en subdividir un AS en pequeños sub-ASes denominados confederaciones BGP, en donde en cada sub-AS se configura un *full-mesh* de sesiones iBGP. Cuanto menor sea la cantidad de routers en cada sub-AS menor será la cantidad de sesiones iBGP dentro del sub-AS logrando, de esta forma, que la implementación de un *full-mesh* de sesiones iBGP no genere una cantidad significativa de sesiones iBGP. Los sub-ASes dentro del AS se comunican con el resto de los sub-ASes a través de sesiones eBGP. Las Confederaciones BGP evitan la generación de bucles introduciendo dos nuevos atributos: "AS CONFED SET" y "AS CONFED SEQ". Más adelante veremos que los Reflectores de Rutas también necesitan introducir dos nuevos atributos para evitar la generación de bucles.

Esta separación del AS en distintos sub-ASes se logra asociando un ASN privado (entre 64512 y 65534 para un ASN 16 bits), que sólo es válido en el AS al que pertenece. Aunque es necesario configurar sesiones eBGP entre los ASes de la confederación, los ASN privados no se anunciarán a través de estas sesiones, logrando así que la subdivisión en sub-ASes sea completamente invisible fuera del AS.

Si bien el subdividir grandes AS permite reducir significativamente la cantidad de sesiones intra-dominio en BGP, las confederaciones BGP acarrean algunos problemas como: encaminamiento sub-óptimo, posibles *loops* y agrega complejidad en cuanto al funcionamiento de BGP.

2.1.7. Reflexión de rutas

La reflexión de rutas [15] permite construir jerarquías en el BGP intra-dominio evitando la necesidad del *full-mesh* de sesiones. Este enfoque permite a un router BGP enseñar rutas aprendidas mediante una sesión iBGP. A estos routers se les denomina "Reflectores de Rutas" (*"Route Reflector" o "RR"*).
Este enfoque representa un cambio en el concepto clásico de BGP en donde cada par iBGP no podía retransmitir lo que había aprendido a través de otro, evitando así que se generen *loops*.

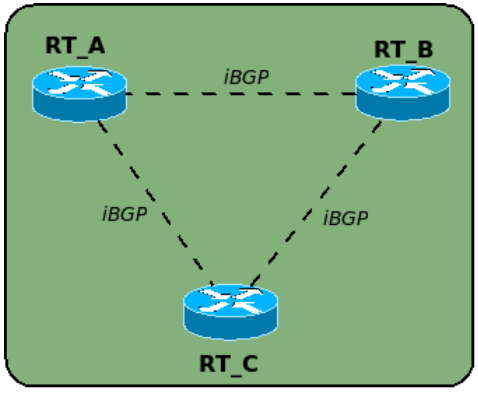


Figura 2.5: Ejemplo de topología con configuración *full-mesh*

Esta solución fue diseñada para satisfacer los siguientes criterios:

- Simplicidad (*Simplicity*): Cualquier alternativa debe ser simple de configurar y fácil de entender.
- Fácil Transición (*Easy Transition*): Tiene que ser posible la transición desde una configuración *full-mesh* sin la necesidad de cambiar drásticamente la topología o el Sistema Autónomo.
- Compatibilidad (*Compatibility*): Tiene que ser posible, para un par iBGP que no entienda el concepto "Reflexión de Rutas", poder seguir siendo parte del Sistema Autónomo original sin perder ninguna información de ruteo de BGP.

Para explicar el concepto de Reflexión de Rutas vamos a hacer una comparación con el modelo clásico de BGP.

En la figura 2.5 se puede apreciar un AS con tres routers (RT_A, RT_B, RT_C) los cuales establecen un *full-mesh* de sesiones iBGP como en el modelo BGP clásico. En este caso, cuando uno de los routers aprenda alguna ruta nueva y ésta sea elegida como la mejor, se la enseñará a los otros dos routers del AS a través de la sesión iBGP que mantiene con cada uno. De esta forma, todos los routers del AS aprenden la ruta enseñada.

En la figura 2.6 se establecen dos sesiones iBGP; una entre RT_A y RT_C, y otra entre RT_B y RT_C. En este caso, si RT_A aprende una ruta y ésta es

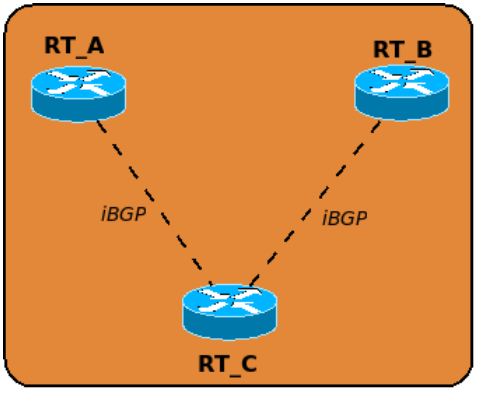


Figura 2.6: Ejemplo de topología con configuración RR

elegida como la mejor, la única forma de que RT_B aprenda esta nueva ruta es a través de RT_C. Para que esto suceda, RT_C tiene que retransmitir lo aprendido por RT_A. En este caso, a RT_C se le denomina "Reflector de Ruta" o "RR".

Los pares iBGP de un Reflector de Rutas se clasifican en dos grupos:

- Pares que son clientes (*Client peers*).
- Pares que no son clientes (*Non-Client peers*).

El RR junto con sus clientes forman un *cluster*. Los routers que no son clientes de ningún RR deben mantener un *full-mesh* entre si, en cambio los que son clientes no necesitan estar conectados de esta manera.
En la imagen 2.7 se ilustran estos conceptos.

2.1.7.1. Comportamiento de un Reflector de Rutas

Cuando un RR recibe una ruta desde un par, éste selecciona el mejor camino. Después que el mejor camino es seleccionado, se hace lo siguiente dependiendo del tipo del par de donde provino la ruta:

- Una ruta que provino de un *Non-Client peer*: Se enseña a todos sus *Client peers*.
- Una ruta que provino de un *Client peer*: Se enseña a todos sus *Non-Client peers* y también a los *Client peers*.

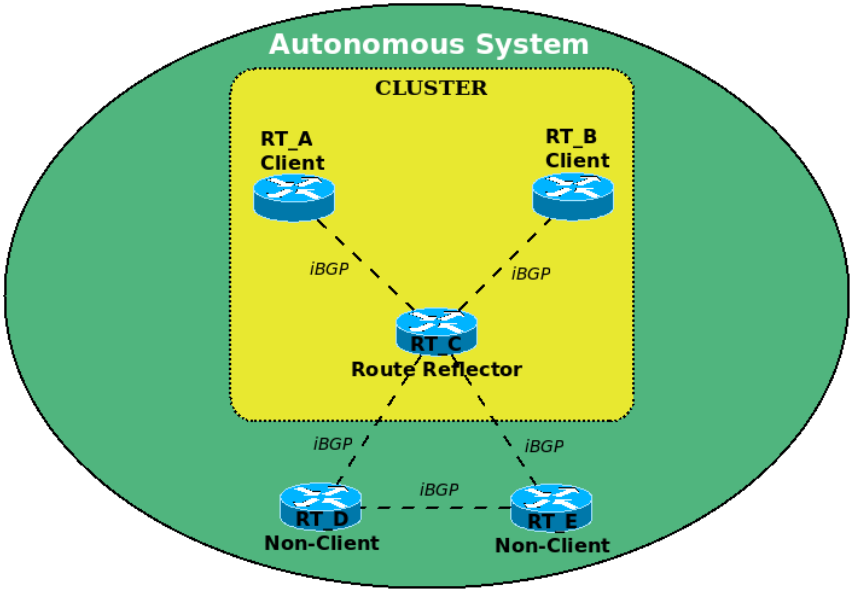


Figura 2.7: Sistema Autónomo con *cluster*

- Una ruta que provino de un par eBGP: Se enseña a todos sus *Non-Client peers* y también a los *Client peers*.

Un AS puede tener más de un RR. Un RR trata a otro RR como cualquier otro par iBGP. Un RR puede pertenecer tanto al grupo de los *Client peers* como al de los *Non-Client peers* de otro RR.

En un AS pueden existir pares iBGP que no entiendan el concepto de *Route Reflection*. A estos los llamaremos pares BGP convencionales. El esquema de *Route Reflection* permite a estos routers formar parte tanto del grupo *Non-Client peers* como del *Client peers*. Esto permite realizar un migración gradual desde un modelo clásico de BGP al modelo de BGP con Reflectores de Rutas.

2.1.7.2. Evitar Loops en la información de ruteo

Cuando una ruta es retransmitida puede generar *loops*. Por esta razón el método de Reflexión de Rutas define los siguientes atributos para poder detectar y

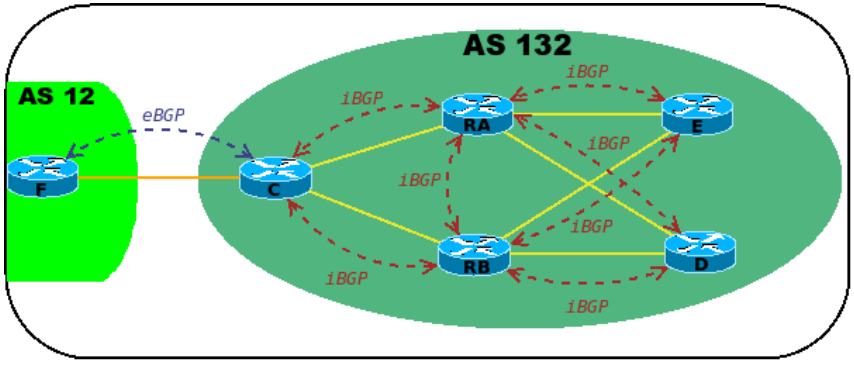


Figura 2.8: RR redundantes

evitar este problema:

- ORIGINATOR_ID es un atributo de 4 bytes y es creado por cualquier RR cuando retransmite una ruta. Este atributo contendrá el identificador BGP del originador de la ruta en el AS local. Un par BGP no debe crear un atributo ORIGINATOR_ID si ya existe uno. Un router que reciba una ruta con su identificador BGP en el atributo ORIGINATOR_ID debe descartar esta ruta.

- CLUSTER_LIST es un atributo que contiene una secuencia de CLUSTER_ID. Representan el camino por el cual se ha retransmitido una ruta. Cuando un RR retransmite una ruta, debe agregar el valor CLUSTER_ID local en el atributo CLUSTER_LIST. Si la secuencia CLUSTER_LIST esta vacía, este debe crear una nueva. Usando este atributo un RR puede saber si esa ruta ya ha pasado por el mismo *cluster*. Si se encuentra su CLUSTER_ID en la secuencia CLUSTER_LIST, el RR debe ignorar la ruta.

Ambos atributos son opcionales y no transitivos (no se propagan a través de sesiones eBGP).

2.1.7.3. RR Redundantes

En el caso que un *cluster* de clientes tenga un sólo RR, el mismo será identificado con el identificador BGP del RR (*"bgp router-id"* de la configuración BGP). Sin embargo, esto representa un solo punto de falla dado que si el router que actúa

como Reflector de Rutas se cae, no habrá ningún router que propague las rutas en ese *cluster*. Para evitar esto, se pueden configurar varios RR en un mismo *cluster*. En este caso, todos los RR del mismo *cluster* pueden ser configurados con el mismo identificador de 4 bytes denominado *CLUSTER_ID*. Esto permite que se puedan desechar rutas que provienen de otros RR en el mismo *cluster*. Para entender esto un poco mejor veamos la figura 2.8.

- Caso 1: RA y RB poseen diferente *cluster id*:
  Supongamos que C recibe una ruta *"p"* de F y se lo anuncia a los reflectores de rutas RA y RB.
  RA crea un nuevo atributo *cluster list*, inserta su *router id* y setea el atributo *originator id* con el *router id* de C. Luego, RA anuncia *"p"* a los routers RB, D y E.
  RB recibe este anuncio y descubre que su *cluster id* no esta en el *cluster list*. Por lo tanto, acepta el anuncio y agrega su *cluster id* a la lista.
  RB tambíen recibe el anuncio enviado por C. En ese momento RB crea un nuevo atributo *cluster list*, inserta su *router id* y también setea el atributo *originator id* con el *router id* de C. Luego, RB corre el algoritmo de mejor camino, selecciona la ruta aprendida directo por C y se lo anuncia a los routers RA, D y E.
  Como RA y RB usan diferente *cluster id* los routers D y E reciben dos copias del mismo anuncio.

- Caso 2: RA y RB poseen el mismo *cluster id*:
  Supongamos que C recibe la ruta *"p"* por F y se lo anuncia a los reflectores RA y RB.
  RA crea un nuevo atributo *cluster list*, inserta su *cluster id* (el mismo que tiene RB) y setea el atributo *originator id* con el *router id* de C. Luego, RA anuncia *"p"* a los routers RB, D y E.
  RB recibe este anuncio y lo descarta ya que su *cluster id* ya esta en la *cluster list*. RB recibe el anuncio enviado por C. RB crea un nuevo atributo *cluster list*, inserta su *cluster id* (el mismo que tiene RA) y también setea el atributo *originator id* con el *router id* de C. Luego, RB se lo anuncia a los routers RA, D y E.
  Cuando RA recibe el anuncio enviado por RB lo descarta ya que su *cluster id* esta en la *cluster list*.
  De esta manera, los reflectores RA y RB mantienen solo una copia de la ruta.

2.1.7.4. Impacto en el Proceso de Selección de Rutas

Estos nuevos atributos impactan en el proceso de selección de rutas de BGP que hemos visto antes. Más específicamente, en el punto "Se prefiere la ruta que ha anunciado el router con menor identificador BGP", debe tratar el ORIGINATOR_ID como el identificador BGP del router que ha anunciado la ruta. Además, es necesario agregar la siguiente regla entre la mencionada antes y la siguiente del proceso de decisión de BGP:
"Un par BGP debe preferir la ruta que tenga el atributo CLUSTER_LIST de menor largo".

2.1.7.5. Consideraciones en la implementación

Cuando un RR retransmite una ruta, éste no debe modificar ninguno de los siguiente atributos:

- NEXT_HOP
- AS_PATH
- LOCAL_PREF
- MED

La modificación de alguno de estos atributos puede llevar a obtener rutas con *loops*.

2.1.7.6. Beneficios de los Reflectores de Rutas

A continuación se describen algunos de los beneficios que se obtienen al tener una topología con Reflectores de Rutas.

Reduce la cantidad de Sesiones iBGP

Los Reflectores de Rutas pueden efectivamente disminuir la cantidad de sesiones iBGP en un Sistema Autónomo, pero ¿cuántas sesiones menos se pueden tener con RRs?. Un router que no sea RR necesita unas pocas sesiones iBGP (típicamente dos para tener redundancia) con Reflectores de Rutas. Sin embargo, los RRs generalmente tienen un número grande de sesiones iBGP. Por lo tanto, la cantidad de sesiones iBGP depende de la cantidad de Reflectores de Rutas y el grado de redundancia que se le quiera dar a los routers que no son RR.

Reduce el costo operacional

Crear, modificar o remover sesiones BGP requiere un costo operacional que muchas veces no se puede apreciar a simple vista. Por ejemplo, en el caso de *full-mesh*, cuando se quiere agregar un nuevo router a un Sistema Autónomo uno puede llegar a pensar que simplemente debe configurar el nuevo router y configurar los enlaces punto a punto con cada uno de los routers vecinos. La realidad es que también se debe modificar la configuración BGP de cada uno de los routers del AS. Esto puede ser un trabajo tedioso si estamos hablando de un ISP de nivel 1 (Tier 1) [16]. En cambio, cuando hablamos de una topología con RRs, este nuevo cambio simplemente requiere la modificación de las configuraciones de BGP para los RRs que establecen sesiones iBGP con este nuevo router, sin tener ningún impacto en el resto de los routers del AS.

Capacidad de implementación incremental

La Reflexión de Rutas permite la coexistencia de los RRs con los routers BGP convencionales. Esto quiere decir que un router BGP convencional B puede ser un *client peer* de un RR, o ser un *non-client peer* de un RR (en cuyo caso B también debe estar conectado con el resto los RRs). Esto permite realizar una migración gradual de una red con el modelo *full-mesh* de sesiones iBGP a una con un modelo de Reflexión de Rutas.

Reduce el tamaño de la RIB-in

Un router BGP mantiene tres diferentes tipos de tablas de ruteo: Adj-RIB-in (o simplemente RIB-in), Loc-RIB y Adj-RIB-out (o RIB-out). Las tablas RIB-in contienen información de ruteo no procesada que le ha sido enseñada por cada unos sus pares BGP. Luego de aplicar el proceso de decisión BGP a partir de las tablas RIB-in, el router decide un único mejor camino hacia cada destino D y lo almacena en la tabla local BGP conocida como Loc-RIB. Si el router es capaz de verificar el siguiente salto para alcanzar el destino D (usando una ruta aprendida por un protocolo IGP, una ruta estática o una red directamente conectada), se almacena dicha ruta en la FIB (*Forwarding Information Base*). Finalmente, la tabla RIB-out contiene las rutas enviadas a cada uno de sus vecinos luego de aplicar las políticas BGP a las rutas almacenadas en la Loc-RIB. Cada router mantiene una tabla RIB-in y una tabla RIB-out por cada vecino BGP, a diferencia de la tabla Loc-RIB que se almacena únicamente una tabla por router.

Los autores que manifiestan que las topologías BGP con Reflectores de Rutas disminuyen el tamaño total de la RIB-in en comparación con el modelo *full-mesh*

(por ejemplo [17], [18] y [19]), argumentan que este comportamiento se da debido a la disminución de la cantidad de tablas RIB-in que mantienen los routers. Dado que la cantidad de tablas RIB-in aumenta proporcionalmente con la cantidad de sesiones BGP que tiene un router R, si R tiene n vecinos y cada uno envía p prefijos, el tamaño total de la RIB-in es del orden de $n \times p$. Con un *full-mesh* de sesiones i-BGP, n es igual a la cantidad de routers del AS menos uno. Dado que los RRs disminuyen la cantidad de sesiones iBGP, sobre todo en los routers clientes de los reflectores, el tamaño total de la RIB-in disminuye.

Este punto aparece en varios documentos como uno de los beneficios de los Reflectores de Rutas por lo que nos pareció importante mencionarlo. De todas formas, creemos que los argumentos proporcionados para dicha afirmación no son suficientes y que incluso esta afirmación puede no ser cierta. Uno de los objetivos que tiene este proyecto es estudiar la veracidad de esta afirmación y realizar una completa argumentación sea cual sea la conclusión obtenida.

Reduce la cantidad de actualizaciones BGP

Los Reflectores de Rutas reciben actualizaciones a través de todos sus vecinos, pero debido a que BGP sólo propaga la mejor ruta hacia cada destino, cada reflector reenviará a sus clientes únicamente aquellas actualizaciones que cambien la selección de la mejor ruta. Esto produce que aquellas actualizaciones que contienen rutas que no son catalogadas como la mejores, no sean recibidas por todos los routers, algo que no sucede en el modelo *full-mesh*.

Es por esta razón y por la disminución de sesiones BGP, que varios autores sostienen que los Reflectores de Rutas disminuyen la cantidad de actualizaciones que son enviadas dentro del Sistema Autónomo. Sin embargo, consideramos que no es suficiente información para validar dicha afirmación. Queda claro que los Reflectores "filtran" o "descartan" algunas actualizaciones y evitan que se propaguen por todo el Sistema Autónomo, pero lo que no queda tan claro es que este comportamiento sea tan significativo como para afirmar que haya una disminución con respecto a la cantidad de actualizaciones.

2.1.7.7. Problemas que acarrean los Reflectores de Rutas

Los RRs se han venido implementando exitosamente desde hace dos décadas en Internet, aunque adolecen de algunos problemas como robustez, retardos en la convergencia de las tablas de encaminamiento, reducción de opciones de encaminamiento, encaminamiento sub-óptimo, posibles *loops* y oscilaciones en el reenvío (*forwarding*) de paquetes.

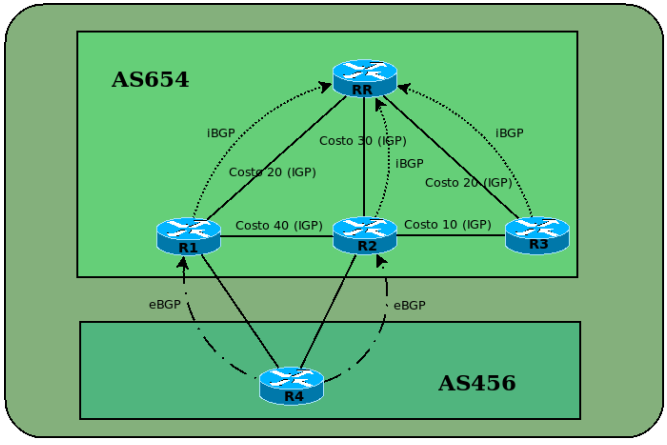


Figura 2.9: Ejemplo de encaminamiento sub-óptimo

Encaminamiento Sub-óptimo

Un RR seleccionará el mejor camino para alcanzar un prefijo de destino utilizando su información de ruteo local, y propagará estos caminos elegidos a sus clientes. Es muy probable que no todos los mejores caminos elegidos por el reflector de rutas sean mejores caminos para sus clientes. Por ejemplo, observando la figura 2.9 imaginemos que el router $R4$, que pertenece al AS 456, enseña a través de una sesión eBGP el prefijo *"p"* a los routers $R1$ y $R2$, que pertenecen al AS 654. Dado que tanto $R1$ como $R2$ son clientes del router RR, ambos le enseñarán a éste router el prefijo *"p"* aprendido, utilizando la sesión iBGP que cada uno de ello comparte con RR. Como el costo del protocolo interno al AS entre RR y $R1$ (costo 20) es menor que el costo entre RR y $R2$ (costo 30), el reflector de rutas RR seleccionará como la mejor ruta al prefijo *"p"* la que se llega a través del router $R1$. Luego, RR le enseñará esta ruta tanto a $R2$ como a $R3$. $R2$ ha aprendido una ruta hacia ese prefijo por una sesión eBGP y, como hemos visto, se prefiere las rutas aprendidas por una sesión eBGP que por una iBGP, por lo tanto, la descarta. $R3$, en cambio, no tiene otra ruta para llegar al prefijo *"p"*, esto hará que tome esta nueva ruta, como la mejor para llegar a este prefijo, teniendo en realidad una mejor ruta a través de $R2$.

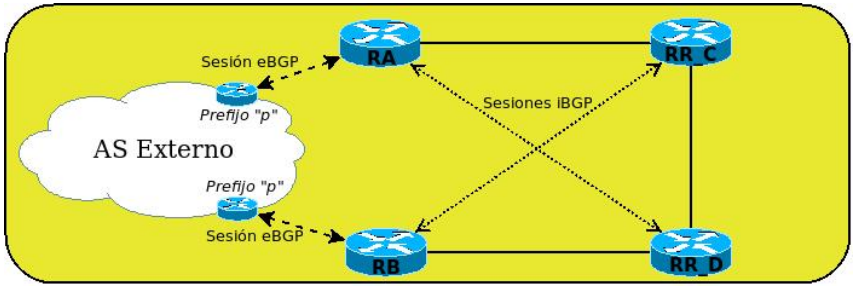


Figura 2.10: Ejemplo de generación de loops

Generación de Loops

Como se puede apreciar en la figura 2.10 los routers RA y RB aprenden un mismo prefijo *"p"* a través de una sesión eBGP con otro AS. Luego RA y RB le enseñan a RR_D y RR_C respectivamente una ruta para llegar al prefijo *"p"*. Cuando RR_C quiera encaminar tráfico hacia el prefijo *"p"*, lo hará por intermedio del router RB. Para esto, el camino posible de RR_C a RB es el siguiente: $[RR\_C, RR\_D, RB]$. El problema surge cuando RR_D reciba el tráfico de RR_C con destino a *"p"*. El camino que RR_C conoce es por intermedio del router RA, pero el camino para llegar de RR_D a RA es el siguiente: $[RR\_D, RR\_C, RA]$. Esto provoca un *loop* ya que el tráfico se encaminará de RR_C a RR_D, y de RR_D a RR_C.
Por lo tanto, un mal diseño de la red con reflectores de rutas puede provocar que se generen *loops*, y en consecuencia, se pierda tráfico.

No determinista

Cuando hablamos de que los reflectores de rutas pueden provocar que el protocolo BGP no sea determinista nos referimos a que dependiendo de cómo se transmita un determinado anuncio, los routers pueden tomar distintas decisiones de ruteo.

Para entender mejor este comportamiento, veamos la figura 2.11. Los routers RA y RB aprenden una ruta a un prefijo *"p"* por una sesion eBGP. Luego, éstos lo enseñan por intermedio de sesiones iBGP a los routers RR_C y RR_D respectivamente. Separemos esta situación en 2 posibles casos:

- Caso 1) El router RA enseña primero al router RR_C la ruta al prefijo *"p"*

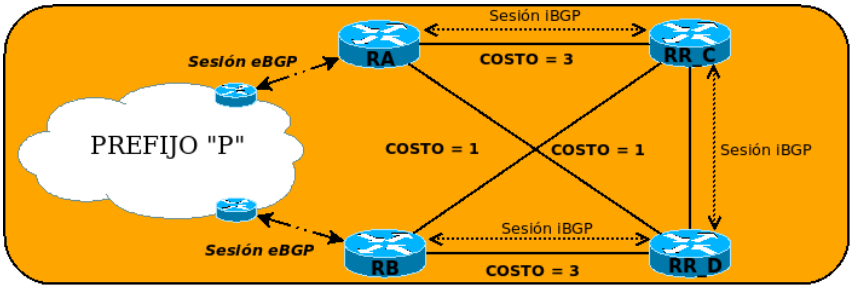


Figura 2.11: Ejemplo de no determinismo

y luego éste se la enseña al router RR_D. Éste último le enseña la ruta al router RB pero éste prefiere la ruta aprendida por eBGP. Luego RB le enseña la ruta aprendida por la sesión eBGP con otro AS al router RR_D. Como el costo entre RR_D y RB (costo 3) es mayor que el costo entre RR_D y RA (costo 1), RR_D descarta el anuncio de proveniente de RB.

- Caso 2) Supongamos ahora que es RB quien enseña primero la ruta al prefijo "P". Manteniendo el razonamiento del "Caso 1", RR_D y RR_C mantendrán una ruta al prefijo "P" a través del router RB.

Robustez frente a fallas IGP

En configuraciones con reflectores de rutas arbitrarias, las propiedades de correctitud como *loop-free forwarding* pueden ser violadas ante cambios en los costos en los enlaces IGP y/o fallas en routers o enlaces. Por ejemplo, se considera la configuración iBGP de la figura 2.12. $R1$, $R2$ y $R3$ son reflectores de rutas, mientras que $C1$ y $C2$ son los clientes. Ambos, $C1$ y $C2$, eligen a $R3$ como su siguiente salto hacia el destino "d" y no hay desviaciones en la ruta hacia $R3$. Cuando $R3$ falla, la topología, ahora equivalente a la de la figura 2.13, tiene un *forwarding loop*. Por lo tanto, es difícil garantizar el *loop-free forwarding* y el encaminamiento óptimo en las topologías iBGP con reflectores de rutas en casos en los que un enlace o nodo fallan. Por la misma razón, es particularmente difícil construir una topología con reflectores de rutas que tenga redundancia, porque si un reflector de rutas falla, el reflector de rutas de *"backup"* puede terminar causando *forwarding loops*.

Por más información sobre los problemas que acarrean los Reflectores de Rutas se puede consultar [20], [21], [22] y [23]

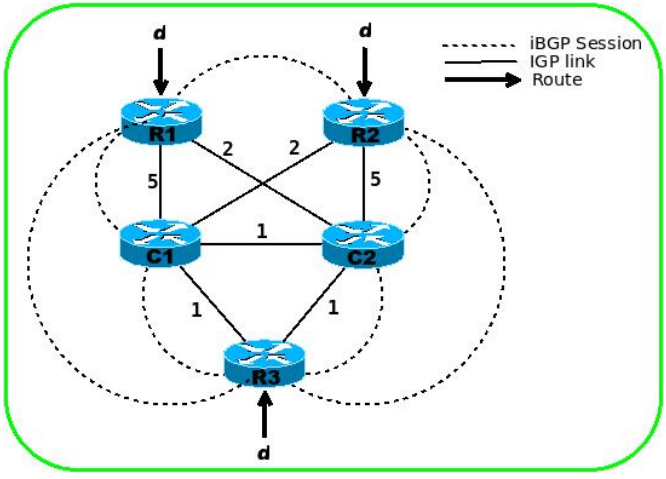


Figura 2.12: Configuración iBGP utilizando reflectores de rutas que tiene baja tolerancia a fallas IGP

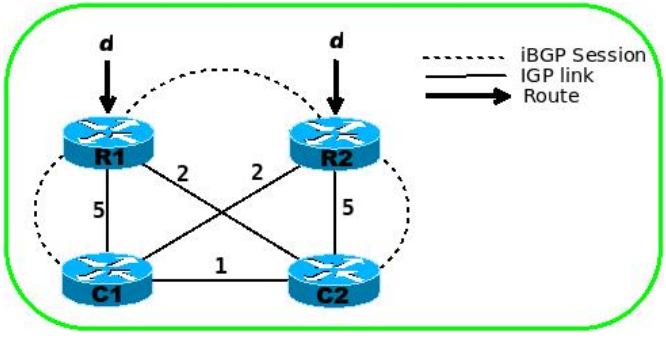


Figura 2.13: Configuración iBGP incorrecta

La construcción de configuraciones iBGP correctas con reflectores de rutas en redes reales con un gran número de routers BGP no es una tarea trivial. En los últimos años, se ha comenzado a estudiar el problema de la localización de reflectores de rutas como un problema de optimización, donde el objetivo generalmente es minimizar la cantidad de sesiones. También se han estudiado condiciones suficientes de corrección de una topología con RRs que evite los problemas mencionados anteriormente.

2.2. Algoritmos existentes

Como ya se ha explicado, Internet está compuesto por muchos Sistemas Autónomos, que son redes o grupos de redes bajo una administración común. Un router de borde de un AS utiliza BGP para intercambiar rutas en una sesión BGP externa (eBGP) con un router par. Cada router eBGP elige la mejor ruta a cada destino, y luego debe difundir las rutas a cada prefijo a los otros routers del AS. Los routers en el AS obtienen información acerca de todas las posibles rutas a cada destino, y eligen su mejor opción.

La opción más utilizada para resolver este problema de diseminación de rutas intra-AS, como se explicó antes, es establecer sesiones BGP entre los routers dentro del AS, en un modo llamado BGP interno (iBGP). Los diseñadores de BGP propusieron el uso de una configuración iBGP *full-mesh*.

La configuración *full-mesh* satisface las siguientes propiedades deseables de correctitud:

- Visibilidad completa: La trasmisión de la información a los routers debe ser "completa" en el sentido de que, para cada destino externo, cada router elige la misma ruta que hubiera elegido si hubiera visto las mejores rutas desde todos los demás routers eBGP en el AS.

- *Loop-free forwarding*: Luego de la trasmisión de las rutas eBGP aprendidas, las rutas resultantes (y los posteriores caminos de reenvío de los paquetes enviados a través de esas rutas) elegidas por todos los routers deben estar libres de *deflections* (desvíos) y *forwarding loops*.

- Robusto frente a fallas IGP: El mecanismo de transmisión de rutas debe ser robusto ante fallas en los nodos, links o cambios de costos IGP. Estos cambios no deberían traer como consecuencia la violación de la propiedad anterior de correctitud.

Desafortunadamente, como ya se explicó, la configuración *full-mesh* no escala: un Sistema Autónomo con n routers necesita tener $n*(n-1)/2$ sesiones iBGP. Un proveedor de servicios de Internet con cientos de routers requerirá muchos miles de sesiones iBGP.

Esta falta de escalabilidad de la configuración *full-mesh* ha sido un problema, y ha llevado a diferentes propuestas para mejorar la misma. La técnica más común utilizada hoy en día (como se ha explicado antes) son los Reflectores de Rutas (un subconjunto de routers BGP son designados como reflectores de rutas, proporcionando las mejores rutas a otros routers configurados como sus clientes).

Existen diferentes algoritmos que construyen topologías iBGP correctas y escalables utilizando reflectores de rutas, algunas de ellas, se detallan a continuación.

2.2.1. BGPSep

El algoritmo BGPSep construye una configuración iBGP que satisface las propiedades de visibilidad completa, *loop-free forwarding* y robusto frente a fallas IGP. Utiliza la noción teórica de conjuntos separadores para generar éstas configuraciones iBGP. Un conjunto separador (*graph separator*) es un conjunto de vértices cuya eliminación separa el grafo en dos o más componentes conexas.
El problema de encontrar el conjunto separador óptimo de un grafo es, en general, un problema *NP-hard*. Sin embargo, para muchas familias de grafos son conocidos algoritmos rápidos y prácticos para encontrar pequeños conjuntos separadores.

Según [24] el número de sesiones iBGP con BGPSep es significativamente más bajo que en una configuración *full-mesh* (un factor entre 2.5 y 5x en topologías de gran tamaño).

El Algoritmo Simplificado:

Sea $G = (V, E)$ el grafo IGP inducido por los routers eBGP, donde V es el conjunto de routers eBGP. A continuación se describe cómo construir una configuración básica iBGP que satisface las propiedades de visibilidad completa, *loop-free forwarding* y robustez frente a fallas IGP, sin requerir una configuración iBGP *full-mesh*. Más adelante se optimizará esta solución simplificada.

- Paso 1: Considerar el grafo separador S de G. Hacer todos los routers de S Reflectores de Rutas.

- Paso 2: Para cada $u, v \in S$, configurar los routers u y v como pares iBPG. Haciendo esto, se forma un *full-mesh* en el nivel superior de la jerarquía de Reflectores de Rutas.
- Paso 3: Hacer que cada router de $V - S$ sea un cliente de cada Reflector de Rutas en S.
- Paso 4: Para cada componente G_i en la que S separa a G, establecer una sesión iBGP entre los routers de G_i (es decir, construir una configuración *full-mesh* en cada componente G_i).

A pesar de que esta construcción básica satisface las propiedades de correctitud y tiene una menor cantidad de sesiones iBGP comprada con el *full-mesh* (los routers en las diferentes componentes conexas no necesitan conectarse entre ellos, sólo necesitan conectarse a los reflectores de rutas en S), ésta configuración sigue utilizando *full-mesh* dentro de las componentes individuales. Para reducir aún más la cantidad de sesiones iBGP, se observó que el problema de evitar una configuración *full-mesh* dentro de G_1 y G_2 sin violar las propiedades de correctitud, es solo una instancia más pequeña del problema original que se comenzó a resolver en G. Por lo tanto, podemos aplicar recursivamente el mismo algoritmo dentro de cada una de las componentes conexas.
La recursión puede terminar una vez que las componentes son lo suficientemente pequeños para ser una configuración *full-mesh*.

El Algoritmo Completo:

El algoritmo 1 muestra el pseudocódigo del algoritmo recursivo centralizado, BGPSep. El mismo toma un grafo $G = (V, E)$ formado por los routers BGP y produce un conjunto I de sesiones iBGP que deben ser establecidas entre los routers. Cada elemento de I denota una sesión iBGP de la forma (u, v, t) donde u y v son los routers entre los cuales la sesión iBGP es establecida y t indica el tipo de sesión. Si $t =$"client", entonces la sesión iBGP entre u y v es una sesión *cliente-Reflector de Ruta* (donde u es el cliente del reflector de ruta v). Si $t =$"peer", entonces la sesión iBGP entre u y v es una sesión iBGP normal sin clientes. La recursión se detiene cuando el componente tiene uno o dos routers.
El algoritmo utiliza el procedimiento *Graph-Separator* (algoritmo de partición de grafos), que toma el grafo G y retorna el conjunto separador del grafo S.

A pesar de que la recursión mostrada en el algoritmo 1 termina cuando cada componente tiene uno o dos routers, se puede modificar fácilmente para que el mismo

Entrada: Grafo IGP G, Conjunto de routers BGP V
Salida: Conjunto I de sesiones iBGP
si $|V| = 1$ **entonces**
  $I = \emptyset$;
si no, si $|V| = 2$ **entonces**
  $\{u, v\} \leftarrow V$;
  $I = \{(u, v, peer)\}$;
en otro caso
  /* Paso 1: Elige un grafo separador $S \subseteq V$. Los routers en S son reflectores de rutas */
  $S \leftarrow Grafo - Separador(G)$;
  $G_1, \ldots, G_m \leftarrow$ componentes de $V - S$;
  /* Paso 2: Crear un full-mesh entre el conjunto de reflectores de rutas */
  para cada $u, v \in S, u \neq v$ **hacer**
    $I = I \cup \{(u, v, peer)\}$;
  fin
  para cada G_i **hacer**
    /* Paso 3: Hacer que todos los routers de cada componente G_i sean clientes de cada route reflector */
    para cada $u \in G_i, v \in S$ **hacer**
      $I = I \cup \{(u, v, client)\}$;
    fin
    /* Paso 4: Aplicar recursivamente BGPSep en cada componente */
    $I_i = BGPSep(G_i)$;
    $I = I \cup I_i$;
  fin
fin
retornar I;

Algoritmo 1: BGPSep

termine la recursión en una etapa más temprana y realizar un *full-mesh* de routers en cada componente. En la práctica, es deseado que el número máximo de niveles de recursión (que es igual al número de niveles de la jerarquía de RR resultante) sea un parámetro definido por el usuario.

Existe una clase de falla en la que se quiebran las propiedades de correctitud de este algoritmo: las fallas iBGP, donde sólo cambia la configuración iBGP sin que cambie la topología IGP subyacente. Esto es, cuando falla la función BGP de un router o una sesión BGP entre routers pares, pero la función de *forwarding* sigue intacta.
Según los autores, si esto sucede, no se puede seguir asumiendo que los nodos en el grafo separador son todos reflectores de rutas, por lo que las garantías de correctitud se quiebran.

Finalmente, los autores aclaran que el algoritmo BGPSep no es un algoritmo incremental, y debe volverse a ejecutar cuando se agregan nodos y/o links a la red. [24]

2.2.2. BGPSep_D

El algoritmo BGPSep_D es una mejora del algoritmo BGPSep planteado anteriormente. A pesar de que BGPSep reduce la cantidad de sesiones iBGP con respecto al *full-mesh*, el mismo no reduce el número de sesiones iBGP de sus reflectores de rutas de nivel superior. Esto es, el nodo de grado máximo de la topología iBGP se mantiene igual que en una configuración *full-mesh*. Esto no es deseable ya que actualizaciones concurrentes desde múltiples pares con una frecuencia alta pueden generar problemas. Por ejemplo, un número grande de actualizaciones por segundo puede causar que un router envíe fuera de tiempo un mensaje KEEPALIVE generando así un reinicio de la sesión.

El Algoritmo BGPSep_D:

Se encontró que existen muchos vértices de grado uno en las topologías IGP de algunos Sistemas Autónomos grandes. Basados en esto, los autores modificaron el algoritmo BGPSep removiendo gradualmente éstos vértices de grado uno del grafo IGP. Si no existen más vértices de grado uno en el subgrafo producido, entonces se aplica BGPSep. A éste algortimo lo llamaron BGPSep_D. Su pseudocódigo se puede ver en el Algoritmo 2.

Entrada: Grafo IGP G, Conjunto de routers BGP V
Salida: Conjunto I de sesiones iBGP

```
/* Paso 1: Remover los nodos de grado uno gradualmente
   */
```

$I = \emptyset$;
$pending = true$;
$G' = G$;
mientras $pending == true$ **hacer**
  $G = G'$;
  $pending = false$;
  para cada $u \in G.V$ **hacer**
    si $d_G(u) == 1$ **entonces** /* $d_G(u)$ es el grado de u en G */
      $v = adj_G(u)$; /* v es el nodo adyacente a u */
      $I = I \cup \{(u, v, client)\}$; /* u es cliente de v */
      $G' = G' - \{u\}$;
      $pending = true$;
  fin
fin

```
/* Paso 2: Aplicar BGPSep sobre el subgrafo generado G'
   */
```

$I_S = BGPSep(G')$;
$I = I \cup I_S$;
retornar I;

Algoritmo 2: BGPSep_D

La idea de BGPSep_D es simple. Si el grado del nodo u es uno en el grafo IGP, entonces u será cliente de su nodo adyacente v. Si v tiene visibilidad completa, siguiendo las reglas de los reflectores de rutas y de selección de rutas BGP, el nodo u tendrá visibilidad completa.

Finalmente, los autores probaron que el algoritmo modificado BGPSep_D cumple las propiedades de visibilidad completa, *loop-free forwarding* y robusto frente a fallas en los nodos y/o links. Además, mostraron que la configuración obtenida con BGPSep_D reduce el grado máximo de la topología iBGP entre un 9 % y un 50 % comparado con BGPSep. Por más información ver [25].

2.2.3. BGPSep_S

El algoritmo BGPSep_S construye una configuración iBGP tomando en consideración el grado de los vértices, los separadores de vértices y los caminos más cortos entre los vértices en el grafo IGP subyacente. Los autores prueban que este algoritmo garantiza visibilidad completa en situaciones normales.
Como ya se explicó antes, el algoritmo BGPSep reduce el número de sesiones iBGP comparado con una configuración *full-mesh*. Sin embargo, no reduce el número de sesiones iBGP en los reflectores de rutas de nivel superior, manteniendo así el grado máximo de la topología iBGP igual al del *full-mesh*. También se mencionó que esto no es deseable, ya que puede acarrear problemas. La configuración iBGP generada con BGPSep_S disminuye el grado máximo de la topología entre un 27 % y 68 % comparado con el *full-mesh*.

Visibilidad Completa y Configuración iBGP:

Para entender la idea y características del algoritmo, esta sección discute la relación entre configuraciones iBGP, visibilidad completa, *forwarding loops* y caminos sub-óptimos.
En primer lugar, se describen algunas notaciones, definiciones y lemas.

Sea G el subgrafo IGP inducido por los routers BGP de una red en un Sistema Autónomo. Sea V el conjunto de routers BGP. Sea d cualquier destino. Para cada router A, sea $Egress_d(A)$ el mejor router de egreso que A hubiera elegido si hubiese visto las mejores rutas desde cada router eBGP en el AS.

Definición 2.2.1 (Signaling chain)**.** Una *signaling chain* entre dos routers A y B es definida como el conjunto de routers $A(= R_0), R_1, R_2, \ldots, R_r, B(= R_{r+1})$, tal que para $i = 1 \ldots r$,

- R_i es un reflector de rutas, y

- Al menos uno de R_{i+1} o R_{i-1} es un cliente del reflector de rutas R_i.

Definición 2.2.2 (Signaling chain de incremento monótono)**.** Una *signaling chain de incremento monótono* entre dos routers A y B en una *signaling chain* $S : A(= R_0), R_1, R_2, \ldots, R_r, B(= R_{r+1})$, tal que para $i = 1 \ldots r + 1$, R_{i-1} es cliente del reflector de rutas R_i.

Definición 2.2.3 (Signaling chain de disminución monótona)**.** Una *signaling chain de disminución monótona* entre dos routers A y B en una *signaling chain* $S : A(= R_0), R_1, R_2, \ldots, R_r, B(= R_{r+1})$, tal que para $i = 1 \ldots r + 1$, R_i es cliente del reflector de rutas R_{i-1}.

Definición 2.2.4 (Concatenación de Signaling chains)**.** Dadas dos *signaling chains* $S_1 : R_0, R_1, R_2, \ldots, R_k$ y $S_2 : R'_0, R'_1, R'_2, \ldots, R'_l$, si $R_k = R'_0$ y $R_0, R_1, R_2, \ldots, R_k(= R'_0), R'_1, R'_2, \ldots, R'_l$ es una *signaling chain*, entonces decimos que S_1 se puede concatenar con S_2. Denotamos esta concatenación de S_1 y S_2 como $S_1||S_2$.

Definición 2.2.5 (Superposición de una Signaling chain en un camino iBGP)**.** Dada la *signaling chain* $S : A(= R_0), R_1, R_2, \ldots, R_r, B(= R_{r+1})$ y el camino iBGP P de A a B, si para $i = 0 \ldots r + 1, R_i \in P$ entonces podemos decir que S se superpone a P, o que P es superpuesto por S.

Definición 2.2.6 (Signaling chain de camino más corto)**.** Para una *signaling chain* $S : A(= R_0), R_1, R_2, \ldots, R_r, B(= R_{r+1})$, si existe un camino más corto IGP P desde A hasta B tal que S se superpone a P, entonces podemos decir que S es la *signaling chain* de camino más corto entre A y B.

Lema 2.2.7. *Para cualquier destino d, si existe una signaling chain de camino más corto entre los routers A y* $Egress_d(A)$*, entonces A aprende la mejor ruta para el destino d vía* $Egress_d(A)$*.*

Usando el lema anterior podemos obtener el siguiente teorema:

Teorema 2.2.8. *Una configuración iBGP garantiza que la propiedad de visibilidad completa se satisface ante cambios IGP arbitrarios si y sólo si, para cualquier camino IGP desde un router BGP a otro router eBGP, existe una signaling chain que se superpone en ese camino.*

El Algoritmo BGPSep_S:

Uno de los objetivos del algoritmo es encontrar una configuración iBGP que garantice la visibilidad completa, evitando de esta manera los *forwarding loops* y los caminos sub-óptimos. De acuerdo con la sección anterior, se debe encontrar una configuración iBGP tal que exista una *signaling chain* de camino más corto desde un router a todos sus posibles egresos.

Como se explicó en el algoritmo BGPSep_D, en un grafo IGP, los vértices de grado uno tendrán visibilidad completa si son clientes de su único vecino (asumiendo que el vecino tiene visibilidad completa).

Además, si en un grafo IGP se puede encontrar un conjunto separador (conjunto de vértices que si se remueven parten al grafo original en componentes conexas), entonces cualquier camino que comience en una componente y termine en otra componente diferente deberá pasar por uno o más routers en el conjunto separador. Si se construye una topología iBGP realizando un *full-mesh* entre los routers del conjunto separador, construyendo una configuración *full-mesh* dentro de cada componente conexa y creando otras sesiones iBGP necesarias de manera que exista una *signaling chain* de camino más corto entre cualquier router en una componente y un router dentro el conjunto separador, entonces existirá una *signaling chain* de camino más corto entre todo par de vértices.

Si se toman uno o más vértices de las componentes conexas y se agregan al conjunto de vértices del separador, se obtiene un super-conjunto del conjunto separador, que sigue siendo un conjunto separador. Obviamente, el máximo grado de las topologías iBGP basadas en los diferentes separadores será diferente. Los autores esperan poder encontrar un separador óptimo tal que el máximo grado de la topología iBGP generada es mínimo. Sin embargo, en la práctica, encontrar este separador óptimo en grafos IGP grandes es una tarea difícil. En cambio, luego de encontrar un conjunto separador se utiliza una heurística para encontrar un super-conjunto del conjunto separador. Primero se halla el camino más corto desde cualquier router en las componentes a cualquier router en el separador. Luego agregamos los vértices en ese camino al super-conjunto, excepto el vértice inicial.

La Descripción del Algoritmo:

El algoritmo BGPSep_S se muestra en los Algoritmos 3 y 4.

BGPSep_S toma un grafo IGP $G = (V, E)$ formado por routers BGP y produce un conjunto I de sesiones iBGP que deben ser establecidas entre los routers. Cada elemento en I denota una sesión iBGP y es de la forma (u, v, t), donde u

Entrada: Grafo IGP G, Conjunto de routers BGP V
Salida: Conjunto I de sesiones iBGP

```
/* Paso 1: Remover los nodos de grado uno gradualmente
   */
I = ∅;
pending = true;
G' = G;
mientras pending == true hacer
    G = G';
    pending = false;
    para cada u ∈ G.V hacer
        si d_G(u) == 1 entonces /* d_G(u) es el grado de u en G
          */
            v = adj_G(u) ;        /* v es el nodo adyacente a u */
            I = I ∪ {(u, v, client)} ;        /* u es cliente de v */
            G' = G' − {u};
            pending = true;
    fin
fin
/* Paso 2: Seleccionar un grafo separador S ⊆ G'.V     */
S = Graph − Separator(G');
G_1, ..., G_m ← componentes de G'.V − S;
/* Paso 3: Encontrar un super-conjunto S+ de S         */
S+ = S;
para cada u ∈ G_i, v ∈ S hacer
    si u ∉ S+ entonces
        P = Shortest − Path(u, v);
        para cada w ∈ P hacer
            si w ≠ u entonces
                S+ = S+ ∪ {u};
        fin
fin
/* Paso 4: Realizar un full-mesh entre los routers en
   S+                                                   */
para cada u, v ∈ S+, u ≠ v hacer
    I = I ∪ {(u, v, peer)};
fin
```

Algoritmo 3: BGPSep_S

```
/* Paso 5: Hacer que cada router en G'.V − S+ sea un
   cliente de algunos reflectores de rutas en S+      */
para cada u ∈ G'.V − S+, v ∈ S+ hacer
    P : u(= R0), R1, R2, ..., Rr, v(= Rr+1) ← Shortest − Path(u, v);
    i = 1;
    mientras Ri ∉ S+ hacer
        i++;
    fin
    I = I ∪ {(u, Ri, client)};
fin
/* Paso 6: Realizar un full-mesh entre los router de
   Gi − S+                                            */
para cada u, v ∈ Gi − S+ hacer
    I = I ∪ {(u, v, peer)};
fin
retornar I;
```

Algoritmo 4: BGPSep_S continuación

y v son los routers entre los cuales la sesión iBGP será establecida y t es el tipo de sesión iBGP. Si $t =$ "client", entonces la sesión iBGP entre u y v es una sesión cliente-reflector de rutas (donde u es cliente del reflector de rutas v). Si $t =$ "peer", entonces la sesión iBGP entre u y v es una sesión iBGP normal.
El algoritmo utiliza el procedimiento *Graph-Separator*, que es un algoritmo de partición de grafos que toma un grafo G como entrada y devuelve un grafo separador S. Además, el algoritmo utiliza otro procedimiento llamado *Shortest-Path* para encontrar el camino más corto entre dos vértices. [26]

2.2.4. Heurística Bates

La heurística Bates recomienda configurar en la topología iBGP con uno o más reflectores de rutas por cada punto de presencia (PoP - *Point of Presence*) en la red. Todos los routers en el PoP son clientes de los reflectores de rutas de ese PoP. Adicionalmente, los autores exigen que se realice un *full-mesh* de sesiones iBGP entre los reflectores de rutas. Por otra parte, recomiendan que se configure un *full-mesh* de sesiones iBGP entre todos los routers en un PoP. [27]
En la implementación de esta heurística, se consideró la topología IGP entera como un sólo PoP. Por lo tanto, solo se ingresa la cantidad de reflectores de rutas

que tendrá la topología iBGP.

Diseño iBGP "Bates1" (BatesY):

Es una variante de la heurística Bates. El router más conectado de cada PoP es seleccionado para ser el reflector de rutas. Cada router es cliente del reflector de rutas en su PoP. Se establece un *full-mesh* de sesiones iBGP entre los reflectores de rutas de los distintos PoPs. Finalmente, se establece un *full-mesh* de sesiones iBGP entre los routers de cada PoP.

Diseño iBGP "Bates2" (BatesZ):

Es otra variante de la heurística Bates. Con propósito de obtener redundancia, dos reflectores de rutas son seleccionados en cada PoP. Estos dos routers son los más conectados en el PoP. Todos los routers en el PoP son clientes iBGP de los dos reflectores de rutas. Se configura un *full-mesh* de sesiones iBGP entre los reflectores de rutas. No requiere *full-mesh* de sesiones entre los routers (no RRs) del PoP.

2.2.5. Heurística Zhang

La heurística Zhang se caracteriza por tener múltiples niveles de reflectores de rutas, generando así una configuración iBGP jerárquica. Los routers que son clientes de los reflectores de rutas del nivel superior pueden ser reflectores de rutas para routers que están en niveles inferiores. Zhang y Bartell realizaron una serie de recomendaciones para el diseño de topologías iBGP jerárquicas [18]. Ellos sugieren que los reflectores de rutas del nivel superior deben establecer un *full-mesh* de sesiones iBGP. Sin embargo, esto no es necesario para los reflectores de rutas de niveles inferiores.
Usualmente se definen dos o tres niveles en la jerarquía. Para ilustrar mejor la heurística se explicará cómo es una configuración con dos niveles de reflectores de rutas (RRs). En el nivel inferior, los routers del PoP (que no son RR) son clientes de dos RRs del PoP del nivel siguiente. Éstos dos RRs son clientes de dos RRs de nivel superior. Además, se realiza una configuración *full-mesh* entre todos los RRs de nivel superior de los PoPs.[27]
La heurística Zhang es utilizada comúnmente por los operadores de red.

2.2.6. Algoritmo Optimal

El algoritmo Optimal propone una solución al diseño de topologías iBGP con reflectores de rutas que converge al mismo estado final que con una configuración *full-mesh*, pero minimizando el número de sesiones iBGP. Este tipo de topologías son llamadas *fm-optimal*.

Antes de presentar el algoritmo, se resumirán las principales ventajas e inconvenientes de las configuraciones iBGP *full-mesh* y con reflectores de rutas. El *full-mesh* es determinista y siempre resulta en un enrutamiento óptimo. La convergencia es rápida y la red es lo más robusta posible. Sin embargo, ésta configuración no escala, son necesarias muchas sesiones iBGP, y agregar o remover routers implica un costo adicional de configuración significativo. Además, un cambio en la mejor ruta desencadena actualizaciones en todos los routers. Por otro lado, en la configuración con reflectores de rutas, la escalabilidad es mejorada en términos de costo adicional de configuración, convergencia, y tamaño de las tablas de enrutamiento. Sin embargo, como ya se ha explicado, se pierde diversidad de rutas, que puede llegar a introducir encaminamiento sub-óptimo y no determinista, oscilación de rutas, desviación de rutas o *forwarding loops*. Además, el comportamiento de las topologías con configuraciones que utilizan reflectores de rutas bajo fallas y cambios en la topología IGP es poco clara.

El algoritmo presentado generará una configuración iBGP que respetará simultáneamente los siguientes requerimientos:

- ***Fm-optimality*:** La configuración con reflectores de rutas es una alternativa escalable a la configuración iBGP *full-mesh*. No se compromete la selección de rutas óptima que se elegirían con el *full-mesh* utilizando reflectores de rutas.
- **Correctitud:** Verificar la correctitud está probado que es un problema *NP-hard*. Sin embargo, gracias a la *fm-optimality*, se puede probar que la red está libre de *loops* (porque está libre de desviaciones) y es determinista, por lo que es correcta.
- **Fiabilidad:** Se diseñó una topología iBGP que sigue el grafo IGP todo lo posible. Sólo se establecerán sesiones *multi hop* (sesiones que atraviesan otros routers BGP además de los los dos routers BGP finales de la sesión) si es necesario.
- **Robustez:** Se construye una topología robusta que soporta fallos en enlaces IGP y routers. Además, incluso luego de una falla en un enlace o la eliminación de un router, la topología se mantiene *fm-optimal*.

- **Escalabilidad:** Se construye una topología que tenga la menor cantidad de sesiones iBGP posible.

Todo lo que se presentará a continuación está basado en el artículo *Designing Optimal iBGP Route-Reflection Topologies*. Por más información ver [28].

Terminología:

Grafo IGP. Sea $G_{igp} = (V_{igp}, E_{igp})$ la topología física de la red. Cada vértice de V_{igp} representa un router y cada arco ponderado (u, v) de E_{igp} caracteriza el enlace físico y su métrica IGP. Se denota con $dist : V_{igp} \times V_{igp} \longrightarrow \mathbb{N}$ la función que devuelve el costo del camino más corto entre dos routers.

Grafo BGP. Denotamos con $\mathcal{N}$ el conjunto de posibles *next-hops* en rutas aprendidas por el Sistema Autónomo (AS), y con $\mathcal{R}$ el conjunto de routers que corren BGP dentro del AS. El grafo $G_{bgp} = (V_{bgp}, E_{bgp})$ describe la topología con reflectores de rutas. $V_{bgp} = \mathcal{R} \cup \mathcal{N}$. E_{bgp} denota el conjunto de sesiones BGP entre los routers. Cuando dos routers comparten una sesión iBGP, se agregan dos aristas entre los routers etiquetadas con *UP* desde el cliente hacia uno de sus reflectores de rutas, *DOWN* desde el reflector de rutas hacia uno de sus clientes, o *OVER* entre pares (ver imagen 2.15).

Se asume que los routers de borde del AS (ASBR) serán los *next-hops* BGP para las rutas ($\mathcal{N} \subseteq \mathcal{R}$). Denotamos por $\mathbb{L} = \{UP, OVER, DOWN\}$ los tipos de sesiones BGP y por $label : E_{bgp} \longrightarrow \mathbb{L}$ la etiqueta de un determinado enlace. Además, denotamos como $sym : \mathbb{L} \longrightarrow \mathbb{L}$ la función que devuelve la etiqueta simétrica para una determinada etiqueta: $sym(UP) = DOWN,\ sym(DOWN) = UP,\ sym(OVER) = OVER$.

Un camino BGP en G_{bgp} es un *camino válido* si está compuesto por cero o más arcos UP, seguido de cero o un arco OVER, seguido de cero o más arcos DOWN. Cualquier secuencia de etiquetado iBGP válido verifica la expresión regular $(UP)^*(OVER)?(DOWN)^*$.

Sea $(n, r) \in \mathcal{N} \times \mathcal{R}$ un par *next-hop* router dado. Cuando consideramos este par (n, r), asumimos que:

- existe un prefijo p y varias rutas (llamadas rutas concurrentes) hacia este prefijo en el AS las cuales no empatan en el costo IGP hacia el *next-hop*,

- n es el *next-hop* BGP más cercano a r en el grafo IGP.

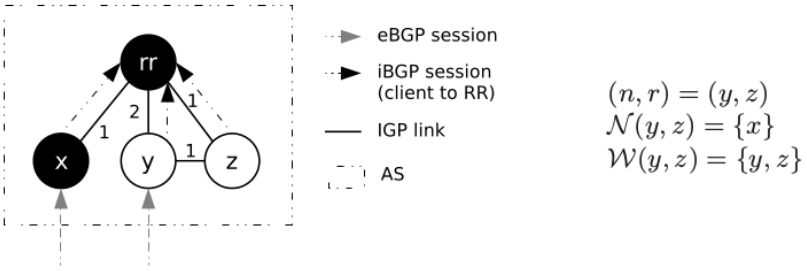


Figura 2.14: Ejemplo de enrutamiento subóptimo: el tráfico enviado por z sigue el camino IGP (z, rr, x) en vez de (z, y)

Se trata de asegurar que r siempre pueda aprender la ruta anunciada por su *next-hop* BGP más cercano n. Únicamente tenemos que considerar el siguiente conjunto de *next-hop* concurrente:

$$\mathcal{N}(n, r) = \{n' \in \mathcal{N}, dist(r, n') > dist(r, n)\}.$$

Si existe un camino iBGP válido desde n a r tal que cada router w de ese camino selecciona la ruta anunciada por n, entonces r aprende la ruta anunciada por n. Llamamos *router blanco* a un router que verifica esta propiedad. El conjunto de routers blancos relacionados con un par dado (n, r) es definido por:

$$\mathcal{W}(n, r) = \{w \in \mathcal{R} / \forall n' \in \mathcal{N}(n, r), dist(w, n) < dist(w, n')\}$$

Notar que n y r siempre pertenecen a $\mathcal{W}(n, r)$. Es más, $\mathcal{N}(n, r) \cap \mathcal{W}(n, r) = \emptyset$.

Llamamos *camino blanco* a cualquier camino iBGP integrado únicamente por routers blancos. Si para cada $(n, r) \in \mathcal{N} \times \mathcal{R}$ existe al menos un camino blanco válido, entonces la topología se dice que es *fm-optimal*. Notar que la *fm-optimality* es independiente del prefijo. Este criterio asegura un buen "comportamiento" de la red para cualquier conjunto de rutas BGP concurrentes.

La figura 2.14 proporciona un breve ejemplo ilustrando la *fm-optimality*. En esta topología, (y, rr, z) es un camino iBGP válido desde y a z. Sin embargo, $rr \notin \mathcal{W}(y, z)$. (y, rr, z) no es un camino blanco. De hecho, rr puede seleccionar la ruta anunciada por x, y z sera incapaz de aprender la ruta anunciada por su *next-hop* más cercano y.

Diseño del algoritmo:

Como entrada se necesitan, el conjunto $\mathcal{N}$ de *next-hops* BGP, el conjunto de routers BGP $\mathcal{R}$, y la topología IGP (G_{igp}).

1. En la primer etapa, se presenta una propuesta para resolver el problema cuando no existen fallas, llamada *caso nominal*. Para cada par $(n, r) \in \mathcal{N} \times \mathcal{R}$, se construye un problema satélite que será satisfecho si y sólo si existe al menos un camino *fm-optimal* desde n a r.

2. En la segunda etapa, se detalla como introducir restricciones para que el algoritmo sea robusto a fallas. Se construye un problema satélite para cada tupla (n, r, f) con una falla dada f.

No se consideran sesiones OVER. Cada sesión OVER se puede convertir en una sesión UP o DOWN sin invalidar ningún camino iBGP.

CASO NOMINAL

Problema Principal

Variables. Para cada sesión iBGP candidata (u, v), $(u, v) \in \mathcal{R}$, $u \neq v$, se definen dos variables de tipo 0-1: $up(u, v)$ (igual a 1 si $label(u, v) = UP$, 0 en otro caso), y $down(u, v)$ (igual a 1 si $label(u, v) = DOWN$, 0 en otro caso).

Función Objetivo. Se trata de diseñar una topología iBGP los más cercana posible a la topología IGP, minimizando el número de sesiones iBGP. Se denota con F la función objetivo y está definida de la siguiente manera:

$$F = min(\sum_{(u,v)\in\mathcal{R}}(R(u, v) * (up(u, v) + down(u, v))))$$

donde $R(u, v)$ caracteriza el número de saltos IGP necesarios para establecer una sesión iBGP desde u hasta v.

Restricciones. El problema principal está compuesto por dos conjuntos de ressstricciones:

- Restricciones de Dominio: Cada par $(u, v) \in \mathcal{R} \times \mathcal{R}$ está conectado por 0 o 1 sesiones iBGP, y $label(u, v) = sym(label(v, u))$. Esto conduce a las siguientes restricciones lineales:
  - $\forall u, v \in \mathcal{R}, up(u, v) + down(u, v) \leq 1$.

- $\forall u, v \in \mathcal{R}, up(u, v) = down(v, u)$.

- Restricciones de flujo máximo y corte mínimo: Al principio, este conjunto de restricciones es vacío. Más adelante se describe cómo se agregan restricciones a este conjunto.

En cada iteración *it*, el problema principal consulta a los problemas satélites para testear la *fm-optimality* de su correspondiente par (n, r). Cada consulta insatisfecha a un problema satélite inserta una nueva restricción de flujo máximo y corte mínimo al problema principal. El conjunto de restricciones de flujo máximo y corte mínimo asegura la propagación de rutas entre cualquier par (n, r) a través del grafo IGP. Si todos los problemas satélites son satisfechos, la resolución del problema retorna una solución *fm-optimal* que minimiza la función objetivo F.

Problemas Satélites

Conceptos de Grafo Extendido. Para garantizar que sólo pueden ser construidos caminos iBGP válidos, se utiliza la transformación de grafos introducida en [29]. Se transforma cada vértice de V_{bgp} en un *meta-nodo* compuesto por dos nodos (llamados *nodo origen* y *nodo destino*) y un arco (llamado *arco interno*), de acuerdo con la figura 2.15. La manera en la que se enlazan dos meta-nodos depende de la relación iBGP entre los dos routers. En el grafo extendido podemos construir únicamente caminos iBGP válidos. Llamamos *meta-arco* al arco que conecta dos vértices pertenecientes a diferentes meta nodos. En el grafo, cada meta-arco es mapeado con la sesión iBGP y los dos routers que establecen esta sesión iBGP. Se denota como $[u, v, rel]$ al meta-arco que es mapeado a los meta-nodos u y v, y a la sesión iBGP rel. Cada camino válido en G_{bgp} desde $s \in V_{bgp}$ a $t \in V_{bgp}$ es entonces mapeado a exactamente un único camino en el grafo extendido desde s_{src} a t_{dst}, donde s_{src} es el nodo origen de s y t_{dst} el nodo destino de t.

Grafos Satélites. Para asegurar la *fm-optimality* de un par dado (n, r), se construye un problema satélite. Para cada problema satélite se construye un grafo satélite $G_w(n, r)$. Cada vértice del grafo pertenece a $\mathcal{W}(n, r)$. Para reducir el número de sesiones iBGP candidatas, sólo se consideran las sesiones (u, v) que verifican las siguientes propiedades:

1. $u, v \in \mathcal{W}(n, r)$: la ruta BGP enviada por n sólo atraviesa routers que no ocultan la ruta;

2. $dist(n, u) \leq dist(n, v)$ y $dist(v, r) \leq dist(u, r)$: un mensaje BGP que cruza el arco (u, v) incrementa su distancia a n y decrementa su distancia a r.

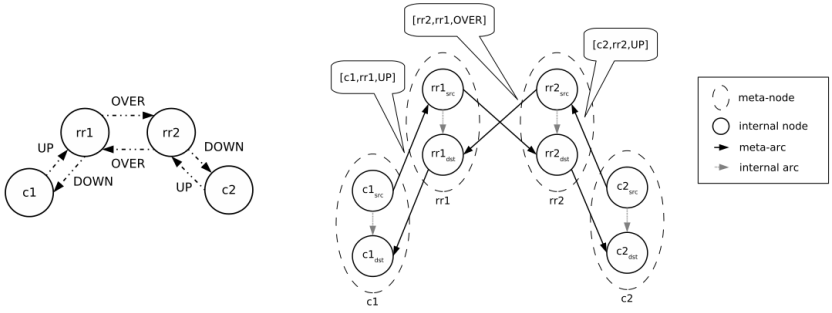


Figura 2.15: Ejemplo de un grafo iBGP y su correspondiente grafo extendido. El camino válido (c_1, rr_1, rr_2, c_2) en G_{bgp} es mapeado a $(c1_{src}, rr1_{src}, rr2_{src}, rr2_{dst}, c2_{dst})$ en G^{ext}_{bgp}

Las sesiones iBGP que no verifican el punto 1 podrían causar que las rutas *fm-optimal* no sean propagadas, por lo que no se quieren utilizar esas sesiones. El punto 2 previene que los mensajes iBGP anunciados por n sigan caminos muy largos desde n y r. Notar que la configuración de una topología iBGP *full-mesh* permanece como posible solución porque la sesión directa entre n y r está permitida.

Así, el grafo $G_w(n,r) = (\mathcal{W}(n,r), E_w(n,r))$ reúne los candidatos a caminos blancos iBGP capaces de satisfacer el par (n,r). Si para todo $(n,r) \in \mathcal{N} \times \mathcal{R}$, existe al menos un camino válido desde n a r en $G_w(n,r)$, entonces la topología iBGP es *fm-optimal*.

Problemas Satélites. Para cada par (n,r) se construye el grafo extendido $G^{ext}_w(n,r)$ desde $G_w(n,r)$ y todos los meta-arcos candidatos. Se denota por n_{src} al nodo origen del meta-nodo n y r_{dst} al nodo destino del meta-nodo r. Se asignará para cada arista $(i,j) \in G^{ext}_w(n,r)$ una capacidad de arco, como se muestra en la figura 2.16.

- Si i y j pertenecen al mismo meta-nodo, instalamos en (i,j) la capacidad de arco infinito.
- De otra manera, (i,j) es un meta-arco. Sea $rel \in \{UP, DOWN\}$ la relación iBGP asignada a (i,j), r_i el meta-nodo mapeado a i, y r_j el meta-nodo mapeado a j: Si en la sesión iBGP rel esta establecida desde r_i a r_j, instalamos

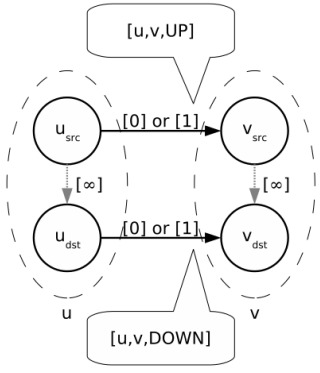


Figura 2.16: Dos sesiones iBGP candidatas: $label(u, v) = UP$ o $label(u, v) = DOWN$

en el arco (i, j) la capacidad igual a 1 (0, en caso contrario). Por lo tanto, hay al menos un meta-arco desde el meta-nodo r_i al meta-nodo r_j con capacidad igual a 1.

Si el flujo máximo enviado desde n_{src} (el origen) a r_{dst} (el destino) es mayor o igual a 1, entonces el par (n, r) es satisfecho. De lo contrario, no puede existir flujo del origen al destino. Se busca el flujo máximo de corte mínimo en este grafo. Se denota como $C(n, r, it)$ al conjunto de meta-arcos que cruzan este corte durante la iteración it. Se inserta la siguiente restricción de flujo máximo y corte mínimo al problema principal:

$$\sum_{[r_i,r_j,rel]\in C(n,r,it)}(rel(r_i, r_j)) \geq 1.$$

La figura 2.17 provee un ejemplo del problema de flujo máximo y corte mínimo. En este ejemplo $\mathcal{W}(n, r) = \{n, r_1, r\}$ y las iteraciones previas han determinado que $label(n, r_1) = UP$, $label(n, r) = NOT$. Cuando el satélite relacionado con (n, r) es consultado, debido al problema de flujo máximo y corte mínimo se inserta la siguiente restricción lineal al problema: $up(r_1, r) + down(r_1, r) + up(n, r) + down(n, r) \geq 1$.

FALLAS IGP

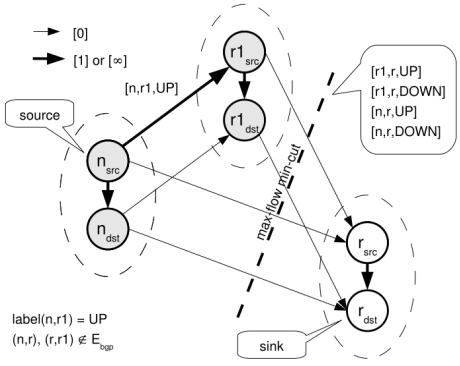


Figura 2.17: Este corte mínimo de flujo máximo inserta la siguiente restricción al problema principal: $up(r_1, r) + down(r_1, r) + up(n, r) + down(n, r) \geq 1$

Ahora se pasará a detallar cómo tener en cuenta los fallos IGP. Un router BGP utiliza su camino más corto IGP para establecer las sesiones con sus vecinos BGP. Cuando un fallo IGP ocurre, los routers BGP actualizan sus árboles de caminos más cortos IGP y restablecen las sesiones BGP rotas de acuerdo a sus nuevos árboles de caminos. Si la conectividad IGP entre dos pares BGP no está funcionando, la sesión BGP se caerá. Se denota como $f = (V^f_{igp}, E^f_{igp})$ a una falla IGP, donde $V^f_{igp} \subseteq V_{igp}$ es el conjunto de routers involucrados y $E^f_{igp} \subseteq E_{igp}$ el conjunto de aristas IGP involucradas en la falla. Se denota como $\emptyset$ la falla vacía.

Metodología. Una falla IGP f consiste en recalcular todos los costos IGP entre cada par de routers de la misma componente conexa. Para cada falla IGP considerada, se aplica el mismo razonamiento del "Caso Nominal" para cada componente conexa. Consideremos un par (n, r) tal que n y r pertenecen a la misma componente conexa C. Sólo se considerarán en $G_w(n, r, f)$ los vértices blancos pertenecientes a C. Una sesión iBGP puede ser establecida solamente si los dos routers pertenecen a la misma componente conexa. Por lo tanto, sólo se consideran las fallas IGP tales que n y r pertenezcan a la misma componente conexa y $n \notin V^f_{igp}$, $r \notin V^f_{igp}$.

Agregación de Satélites. Si construimos un satélite (n, r, f) por cada falla IGP (incluido $\emptyset$), notamos que hay muchos satélites redundantes. Por ejemplo, si f no afecta al par (n, r), no tiene sentido construir un satélite para (n, r, f), ya que (n, r, f) y $(n, r, \emptyset)$ insertarán las mismas restricciones de flujo. Consideremos dos

fallas f y f' para un par dado (n, r). Sean $G_w(n, r, f)$ y $G_w(n, r, f')$ los grafos de flujos correspondientes. Si $G_w(n, r, f) \leq G_w(n, r, f')$, las restricciones introducidas por $G_w(n, r, f)$ serán más restrictivas que las de $G_w(n, r, f')$. Por lo que podemos omitir sin problemas el satélite $G_w(n, r, f')$.

2.3. Revisión de herramientas

Con el objetivo de estudiar los algoritmos de localización de Reflectores de Rutas se investigaron las herramientas a utilizar para la realización de simulaciones/emulaciones de diferentes topologías de redes. Se buscaron requerimientos mínimos, como ser, que soporte ASN de 32 bits, IPv6, entre otros.

En esta sección se explicará qué herramientas se estudiaron para la realización de este proyecto. Más adelante se detallará cuáles fueron las que se utilizaron y las razones por las cuáles se eligieron las mismas.

2.3.1. Totem

TOTEM es una herramienta *open-source* que provee un *framework* donde los investigadores pueden integrar sus algoritmos de ingeniería de tráfico (*traffic engineering* - TE). Éstos algoritmos pueden ser aplicados en modelos de redes reales. También, da la posibilidad a los operadores de redes de experimentar los algoritmos de TE desarrollados en la herramienta en sus propias redes. [30]

TOTEM es una de las pocas herramientas *open-source* para el estudio de TE inter e intra dominio en redes IP o MPLS, proporcionando métodos robustos y estables para la optimización de la métrica IGP, entre otros.

La herramienta puede ser utilizada para simular cómo el tráfico será ruteado en una red que utiliza SPF (*shortest path first*), CSPF (*constrained shortest path first*) o algún otro algoritmo de ruteo de TE. También puede simular escenarios "qué pasa si?" (*what-if*) para ayudar a entender los efectos de cambios de métricas, fallos, cambios en el tráfico o de políticas BGP.

El *open-framework* provisto por la aplicación permite una rápida integración de nuevos métodos. Los principales beneficios son:

- Reuso de componentes existentes como los módulos de topología o de matriz de tráfico.
- Comparación con algoritmos existentes en la herramienta.

- Integración de nuevos algoritmos de TE con los ya existentes, como el simulador BGP.

TOTEM incluye C-BGP, un eficiente simulador de decisión BGP. Esta herramienta puede ser utilizada para evaluar el impacto de políticas de entrada/salida en las tablas de ruteo de diferentes AS. También puede ser usada para experimentar con procesos de decisión modificados y atributos BGP adicionales. Como se verá cuando se hable de C-BGP, el mismo no está actualizado y no soporta IPv6 ni tampoco ASN de 32 bits, por lo que se puede utilizar el simulador C-BGP de TOTEM bajo ciertas restricciones.

2.3.2. RRLoc (xTotem)

RRLoc es una extensión de TOTEM diseñada como un *framework* para implementar y testear algoritmos de localización de Reflectores de Rutas. [31]

Algunas de las funcionalidades que fueron añadidas a TOTEM son:

- Una representación en forma de grafo de la topología donde los nodos son los routers iBGP y los links las sesiones iBGP. Esta vista fue integrada a la GUI de TOTEM. En las configuraciones con reflectores de rutas, se pueden distinguir los mismos de los routers BGP comunes por el color (Azules: RRs - Verdes: Routers BGP comunes).
- La posibilidad de exportar la configuración IGP+iBGP de la topología en formato C-BGP lista para utilizar en el simulador.
- Asignación automática de direcciones IP a los nodos de la red, previniendo así, los posibles errores de la asignación manual. Muy útil para las simulaciones.

Además, se implementaron los siguientes algoritmos de localización de reflectores de rutas que ya fueron explicados en detalle en la sección anterior:

- BGPSep
- BGPSepD
- BGPSepS
- Optimal
- Heurística Bates

- Heurística BatesY
- Heurística BatesZ
- Heurística Zhang

También se encuentra implementada la configuración *full-mesh* para poder, de esta manera, realizar comparaciones con los algoritmos antes mencionados.

2.3.3. Simuladores/Emuladores

A continuación se describen las principales características y desventajas de los simuladores y emuladores estudiados para este proyecto.
Cabe aclarar que se tuvieron en cuenta otros (NS3, J-SIM, SSFnet), sin embargo, aquí se detallan los que fueron probados y analizados en más profundidad.

2.3.3.1. C-BGP

C-BGP es un simulador de BGP. Tiene el objetivo de calcular el resultado del proceso de decisión de BGP en redes compuestas por varios routers. Para este propósito, se tiene en cuenta la configuración de los routers, las rutas BGP recibidas del exterior y la topología de la red. Soporta el proceso de decisión BGP completo, reflectores de rutas y atributos experimentales, como las *communities*, entre otros. [32]

Durante el diseño y modelado de C-BGP, se omitieron algunas partes del protocolo BGP. Esto incluye, por ejemplo, los de la capa de transporte (TCP), el establecimiento de la conexión, los mensajes de mantenimiento de conexión y algunas partes de la Máquina de Estados Finita de BGP. [33]

C-BGP puede utilizarse como una herramienta de investigación con procesos de decisión modificados y atributos de ruteo BGP adicional. También puede ser utilizado por un operador de un ISP para evaluar el impacto de cambios lógicos y/o topológicos en las tablas de ruteo de sus routers. Cambios en la topología incluyen fallas en links o routers. Cambios lógicos incluyen cambios en la configuración de los routers como ser políticas de ruteo de paquetes entrantes o salientes, o cambios en los pesos de los links IGP.

Tiene la gran ventaja de que puede ser utilizado en topologías de gran escala, del mismo orden de magnitud que Internet.

C-BGP es un simulador a eventos discretos, con la particularidad que cada corrida del simulador con los mismos datos de entrada, devuelve la misma salida. Esto es una desventaja para el simulador. En las redes reales, todos los routers

de la topología envían y reciben mensajes BGP en un mismo instante de tiempo. En el caso de C-BGP esto no es así, por lo que **no** se puede pensar que este simulador es completamente realista. Tomemos como ejemplo un mensaje BGP de actualización; hay un nuevo prefijo disponible por una cierta ruta. El mensaje llegará al router de borde del Sistema Autónomo y éste lo enviará a sus vecinos. Mientras envía la actualización a los vecinos, llega un mensaje para dar de baja ese prefijo. El procedimiento del router de borde en una red real es dejar de enviar a los vecinos la actualización. A los que ya le envió el mensaje, le mandará un mensaje para dar de baja la ruta; pero a los que no le llegó a mandar la actualización, no les enviará nada. Este caso particular en C-BGP no se dará de la misma manera. El router de borde primero enviará la actualización de la ruta a todos sus vecinos, para luego recibir la baja de dicha ruta. Al recibir la baja de la ruta, enviará a todos sus vecinos la baja de la misma. Esto genera más tráfico del que debería, por lo que no nos dará una medición real.

Otra desventaja encontrada en las pruebas con C-BGP son que el simulador no soporta IPv6 ni ASN de 32 bits en su última versión oficial.

2.3.3.2. MiniNet/MiniNExT

Mininet es un emulador de red. Corre una colección de hosts, switches, routers y links en un único kernel de Linux. Utiliza virtualización para que un sólo sistema parezca una red completa, corriendo el mismo kernel, sistema y código de usuario. Un host Mininet se comporta como una máquina real; por lo que se puede ingresar a él a través de ssh y se pueden correr programas arbitrarios (incluyendo aquellos que vienen instalados por defecto en una máquina Linux). Los programas que se corren pueden enviar paquetes a través de lo que parece una interfaz Ethernet real con una velocidad de link y delay dadas. Los paquetes son procesados por lo que parece ser un switch o router real. [34]

Algunas ventajas de Mininet:

- Es rápido. Levantar una red simple lleva unos pocos segundos.
- Se pueden crear topologías personalizadas.
- Se pueden correr programas reales. Cualquier programa que corra sobre Linux se puede correr en Mininet, desde web servers, herramientas que monitorean la ventana TCP, hasta Wireshark.
- Es fácil de usar. Se puede crear y correr un experimento Mininet escribiendo scripts de Python simples (o complejos, si se quiere).

- No es pesado, es decir, no necesita un equipo con una determinada capacidad. Se puede correr en un laptop.

Una desventaja encontrada es que Mininet utiliza un único kernel de Linux para todos los hosts virtuales; esto significa que no se puede correr un software que dependa de BSD, Windows u otro kernel de otro sistema operativo.

MiniNExT es una extensión de Mininet que facilita la creación de redes complejas en Mininet.[35]

Los host y contenedores MiniNExT proporcionan un mayor aislamiento, incluyendo:

- PID, identificadores de procesos: aisla los procesos de cada contenedor, facilitando el análisis y soporte de la aplicación.
- UTS, identificadores de hosts: cada host tiene su propio nombre, simplificando el análisis y el *debugging*.
- Cada host tiene su propio directorio */var/log* y */run*.

Además, hace más fácil expresar las configuraciones comunes con:

- *Service helpers*: hace que sea más fácil ejecutar y administrar los servicios en los hosts.
- *Network helpers*: hace que sea más fácil configurar las interfaces de *loopback* y redes NAT.
- *Mount management*: hace que sea más fácil proporcionar a los hosts con configuraciones de aplicaciones individuales.

2.3.3.3. OMNeT++

OMNeT++ (*Objective Modular Network Tested in C++*) es un simulador de eventos discretos basado en componentes, *open source*, extensible y modular. Se puede utilizar para simular tanto redes cableadas, como redes inalámbricas. Es mayormente utilizado con fines educativos o de investigación. [36]

El simulador ofrece un IDE basado en Eclipse, un entorno de ejecución gráfico y una gran cantidad de herramientas. Es un simulador de propósito general capaz de simular cualquier sistema compuesto por dispositivos que interactúan entre ellos. Las simulaciones pueden ser corridas sobre una variedad de interfaces de

usuario. Las interfaces de usuario más utilizadas para propósitos de demostración y *debugging* son las gráficas y la animada.

No se realizó un estudio en profundidad de este simulador ya que, a pesar de tener un entorno gráfico que facilita mucho las cosas, es difícil de implementar. Utiliza un lenguaje llamado NED (*Network Description*) para describir la estructura del modelo a simular.
Otra desventaja, es que es un simulador a eventos discretos como C-BGP, por lo que no se estaría simulando exactamente la realidad.

2.3.3.4. CORE

CORE (*Common Open Research Emulator*) es una herramienta para emular una red en una o más máquinas. Éstas redes emuladas se pueden conectar a redes reales. Utiliza Linux Containers (LXC) como método de virtualización. Provee una interfaz gráfica para crear las redes y ofrece funcionalidades útiles para inspeccionar el estado de los elementos de la red virtual, para correr las aplicaciones y crear tráfico en la red. Además, el usuario puede guardar la configuración de la red en un archivo y/o cargar redes a partir de archivos de configuración. [37]

Una desventaja encontrada es que no se puede modificar ningún elemento durante el momento de ejecución. Es decir no se pueden borrar routers, dar de baja links, etc. [38]
Además, a pesar de aceptar el formato XML (*eXtensible Markup Language*) como formato de entrada para las topologías, los archivos generados por la herramienta RRLoc no son válidos para CORE.

2.3.4. Inyectores de rutas BGP

Se estudiaron dos herramientas para realizar la inyección de rutas: exaBGP y BGPsimple. A continuación, se describen sus principales características.

2.3.4.1. exaBGP

ExaBGP es una herramienta desarrollada en *Python*, diseñada para permitir la inyección de rutas arbitrarias en una red incluyendo IPv6 y FlowSpec. Además, se puede utilizar para mitigar ataques de denegación de servicio y para monitorear la red. [39]

Utiliza un archivo de configuración y un *script* para crear los mensajes de actualización BGP. El router que inyectará las rutas al AS deberá correr el programa con el archivo de configuración para poder comenzar a inyectar las mismas.

Una desventaja de la herramienta es que, una vez que se termina de ejecutar el *script* de inyección, la aplicación envía un mensaje para dar de baja todas las rutas con *next-hop* él mismo al par al que le envió las actualizaciones. Por lo tanto, si se quiere tomar medidas, por ejemplo cantidad de mensajes enviados, se deberá tener la precaución de no dejar que finalice la aplicación hasta tanto no tener las mediciones hechas (utilizando un *time-sleep*, por ejemplo).

2.3.4.2. BGPsimple

BGPsimple es un *script* de *Perl* que permite establecer una sesión BGP con un par, monitorear los mensajes recibidos de ese par, y enviar actualizaciones a partir de un conjunto predefinido de atributos.

El *script* fue escrito principalmente para tomar un archivo con información de ruteo BGP (en formato TABLE_DUMP_V2) e inyectar estas rutas al par BGP. Los mensajes de actualización son guardados en un *log*. Sin embargo, las tablas *adj-rib-in* y *adj-rib-out* no se mantienen.
Es muy fácil de utilizar y poner en funcionamiento, sin embargo, tiene algunas desventajas: no soporta ASN de 32 bits, ni IPv6. [40]

Capítulo 3

Diseño y Desarrollo

El objetivo de este proyecto es estudiar los algoritmos de localización de Reflectores de Rutas. Para lograr este objetivo, se diseñó e implementó un ambiente de trabajo que permite realizar pruebas con las diferentes topologías (luego de haberles aplicado el algún algoritmo) y sacar conclusiones en base a los resultados obtenidos.

Es de suma importancia poder realizar las pruebas en una red lo más parecida a una red real, para así obtener resultados más acertados. Por esto, resulta un punto crítico encontrar una herramienta que permita lograrlo, y a su vez consuma la menor cantidad de recursos (para así poder realizar las pruebas simulando Sistemas Autónomos reales de gran tamaño).

En este capítulo se va a definir el ambiente de trabajo, qué características son necesarias que tengan las herramientas a utilizar y se justificará porqué se eligieron dichas herramientas. Además, se diseñará y desarrollará un *framework* para interactuar con las herramientas y automatizar algunas tareas.

3.1. Ambiente de Trabajo

A la hora de realizar la búsqueda de una herramienta de emulación/simulación necesitamos tener bien claro cuáles son las características que nos interesan que tenga la herramienta que utilizaremos.
Un aspecto importante que debemos considerar es la inyección de trazas BGP en el AS generado: la herramienta que utilicemos deber permitir esto, ya que es sumamente importante para conseguir el nivel de realismo que precisamos.

Se definió, por tanto, que la herramienta deberá tener las siguientes características:

- Debe soportar IPv6.
- Debe soportar ASN de 32 bits.
- Deberá tener la capacidad de poder inyectar rutas o en su defecto, deberá ser posible agregarle alguna herramienta sencilla de inyección de rutas.
- Debe consumir pocos recursos. De esta manera se podrán simular topologías reales en computadoras personales.
- Debe soportar protocolos de ruteo (BGP y al menos uno de los protocolos IGP).
- Debe permitir la creación de topologías personalizadas.

Con todas estas características en mente se seleccionó Mininet con la extensión MiniNExT como nuestro emulador. Luego de realizada la elección se pasó a la etapa de pruebas para verificar que realmente este emulador cumplía con nuestras exigencias.

Para agilizar la implementación de las redes en MiniNExT utilizamos una *API*(*Application Programming Interface*) de *Python*. Esta interfaz nos brinda una serie de funciones que simplifican la configuración de toda la red. Se puede encontrar documentación de esta API en el manual *Mininet Python API Reference Manual* en [41].

Para las pruebas, en una primera instancia seleccionamos una topología simple como lo es "Abilene" (ver figura 3.1). A continuación, se configuró la topología para poder ser utilizada en el emulador. Luego, debido a que xTotem sólo asigna una dirección IP por cada router (la dirección de *loopback*) y no por cada interfaz, tuvimos que asignar una por una las direcciones IP de cada interfaz. Dicha asignación se hizo con IPs privadas para asegurarnos que no se repitan.

Una vez realizada esta tarea pasamos a configurar los protocolos de ruteo. Como protocolo IGP seleccionamos el protocolo OSPF. Perfectamente se pudieron haber configurado rutas estáticas, pero esto era muy poco automatizable y exigía un trabajo mayor. Luego, pasamos a el foco principal del estudio y configuramos el protocolo BGP. Para esto, como primer caso de pruebas se implementó un *full-mesh* de sesiones iBGP.

Para contemplar varios escenarios, configuramos dos routers que se conectan a

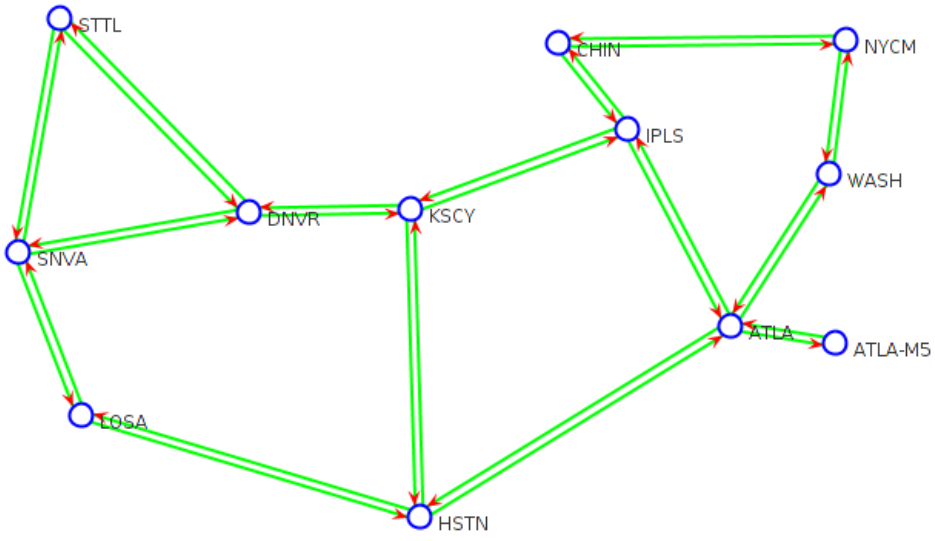


Figura 3.1: Imagen de la topología Abilene vista desde xTotem

nuestro AS. Éstos routers son el origen de las trazas BGP, pertenecen a dos ASes diferentes y establecen sesiones eBGP con los routers de borde de nuestro AS.

Una vez pronta la configuración de todo el Sistema Autónomo, levantamos el emulador y empezamos a realizar las pruebas para ver el comportamiento del mismo. Para esto, obtuvimos trazas BGP de la página del **Centro de Coordinación de redes IP europeas** [42] y, con la ayuda de la herramienta *exabgp*, inyectamos una de estas trazas para verificar que se propagaba por todo el AS.

Una de las características importantes que debe tener el emulador es la de brindar información necesaria en cuanto a los mensajes que participan en el protocolo BGP, en particular en las sesiones iBGP. Para poder obtener dicha información, lo que hicimos fue almacenar en cada router dos archivos, uno que contiene los mensajes BGP que recibió, tanto *updates* como *withdraws*, y otro que contiene *snapshots* de la FIB cada cierto tiempo.
Con la ayuda de estos archivos nos dimos cuenta de algunos factores que estábamos pasando por alto. Por ejemplo, como los dispositivos de nuestra red tienen interfaces con direcciones IPv4, todos los mensajes BGP que contienen direcciones IPv6 son descartados por el router de borde de nuestro AS. Por lo tanto, no son

procesados por el resto de los routers. De esta prueba, concluimos que era necesario realizar la asignación de direcciones IPv6 a las interfaces de los routers. Esto es realmente importante ya que se intenta simular redes reales e IPv6 es parte de las redes de todo el mundo hoy en día.

Siguiendo con las pruebas, realizamos inyección de trazas BGP por dos puntos separados de nuestro AS, donde cada una de estas trazas enseñaba la misma ruta. De esta forma, pudimos observar el comportamiento del emulador y ver que funcionaba correctamente. En particular, comprobamos que el proceso de decisión de rutas para un mismo prefijo se produce como se vio en la sección 2.1.4 de este documento.

Para poder estudiar con más detalle los datos que arrojó esta primera emulación, guardamos en una base de datos los mensajes que se generaron a raíz de las trazas BGP. De esta manera, con una simple consulta a la base podemos comparar las entradas en la tabla de *forwarding* de cada router, verificando el correcto funcionamiento de la herramienta.

Terminada la primera instancias de pruebas, procedimos a implementar la topología con los Reflectores de Rutas. Para obtener la localización de los Reflectores utilizamos la herramienta xTotem y el algoritmo BGP-SepD. Una vez obtenida la localización de los RRs, configuramos los mismos y realizamos las mismas pruebas que habíamos realizado para la topología con *full-mesh* para verificar su correcto funcionamiento. Uno de los chequeos realizados fue ver que los routers de la topología configurados con Reflectores de Rutas aprendían los mismos prefijos que la topología con configuración *full-mesh*.

En resumen, en la primera etapa se realizaron pruebas para ir encaminando el proyecto, se estudiaron las maneras de obtener la información necesaria para utilizar durante el desarrollo del mismo además de entender mejor qué información es la que queremos obtener. También se investigó cómo obtener esta información de las herramientas que utilizamos (MiniNExT). Sin embargo, para realizar estas primeras pruebas fueron necesarios varios pasos (crear routers, enlaces, asignar direcciones IP para cada interfaz, crear las tablas en la base de datos y cargar esta base) que si bien no son complejos, consumen mucho tiempo realizarlas manualmente considerando que las pruebas realizadas fueron relativamente simples.
El objetivo de este proyecto es estudiar la mensajería BGP en distintos Sistemas Autónomos en donde la utilización de los Reflectores de Rutas pueda ser una mejora, por lo tanto estamos hablando de topologías muy grandes. Para que sea viable este estudio, considerando que por cada topología existen distintas localizaciones

de los RR dependiendo el algoritmo que se le haya aplicado a la red, es necesario la automatización del proceso de ingresar los datos a la emulación a partir de la salida de la herramienta xTotem. También es deseable que una vez corridas las simulaciones, toda la información obtenida se pueda cargar en la base de datos automáticamente de manera que lo único que sea manual sea la realización de las consultas a la base para luego realizar las comparaciones pertinentes.
En la siguiente sección se explican las herramientas desarrolladas y cómo ayudan a mejorar el tiempo que insumen ciertas tareas y reducen las posibilidades del error humano. Sin embargo, antes de pasar a la siguiente sección se contestarán dos preguntas importantes que nos planteamos durante el desarrollo del proyecto.

¿Porqué un emulador para estudiar BGP?

Hasta el momento hemos visto una gran cantidad de conceptos teóricos acerca de BGP y en particular de *Route Reflection*. Hemos visto que la cantidad de sesiones iBGP disminuye cuando utilizamos Reflectores de Rutas y los problemas que introduce este cambio de paradigma. Sin embargo, en este proyecto no es únicamente esto lo que nos interesa, consideramos que lo más importante es analizar la *performance* del protocolo BGP en la práctica. Para esto, es fundamental una herramienta que permita generar un ambiente lo más real posible.

La emulación es una tecnología clave en la investigación de redes de gran tamaño donde experimentar con *hardware* genera un alto costo y no siempre es viable. Las redes implementadas con *hardware* son difíciles de configurar y requieren de mucho tiempo. Sin embargo, es casi tan difícil llevar a cabo un experimento de gran escala con *hardware* que realizarlo con una emulación. La emulación elimina problemas tales como la necesidad de conectar físicamente cables, pero es necesaria la configuración de muchos dispositivos, incluyendo los routers y *switches*. La configuración de router es particularmente difícil en redes complejas. A pesar de esto, la emulación se ha vuelto popular debido al incremento de la habilidad de correr múltiples routers virtuales en una sola computadora. Además, las computadoras personales hoy en día tienen un poder de procesamiento considerable que permite emular toda una red, con sus protocolos de ruteo y decenas de routers sin ningún tipo de problema.

La simulación es otra herramienta a tener en cuenta para experimentos de gran escala, pero dada la necesidad que surge de llevar a cabo experimentos que se asemejen a la realidad, la emulación es la herramienta más acertada para un estudio de estas características. La emulación permite experimentos más realistas que las simulaciones pero como contrapartida emular requiere mayores recursos.

La configuración manual es la raíz del problema en la emulación, ya que intro-

duce la posibilidad del error humano. Vamos a estudiar redes a gran escala, que pueden contener cientos de routers, lo que implica cientos o miles de archivos de configuración. Sin embargo, la cantidad de datos diferentes entre estos archivos es a menudo relativamente pequeño, y por lo general, limitado a áreas tales como la configuración de IP, los atributos de la comunidad, los cambios en los pesos de los enlaces y la política de enrutamiento. La mayoría de los cambios de configuración complejos están reservados para las implementaciones de políticas BGP en el borde de la red.

¿Porqué utilizamos Quagga?

Quagga es una suite de software que permite a las familias de Sistemas Operativos Unix (entre ellos, Linux) funcionar como routers [43]. El mismo provee implementaciones de protocolos como *Open Shortest Path First* (OSPF), *Routing Information Protocol* (RIP), *Border Gateway Protocol* (BGP) e *IS-IS*. *Quagga* no es el único software que implementa protocolos de ruteo para plataformas Unix, XORP [44] y BIRD (*BIRD Internet Routing Daemon*) [45] son otros de los más conocidos. En cuanto a popularidad y a semejanza con *Quagga*, BIRD debe ser el competidor más cercano. Si bien tanto *Quagga*, XORP y BIRD cumplen con lo necesario para poder realizar nuestro estudio, hemos elegido *Quagga* como suite de software para *Routing*. La decisión de utilizar *Quagga* se basó en la experiencia personal que hemos tenido con el software y, dado su similitud con *CISCO IOS*, hace muy fácil trabajar con *Quagga* a aquellos que trabajamos con equipamiento de esta marca. *Quagga* se puede ver como una colección de pequeños demonios donde cada uno tiene una tarea específica, como por ejemplo, correr un protocolo de ruteo como OSPF o BGP. El *core* de *Quagga* es principalmente el demonio *Zebra*, que actúa como una capa de abstracción al *kernel* de Linux y presenta la API Zserv. Esta API es utilizada por los demonios encargados de correr los protocolos de ruteo para almacenar la información de ruteo aprendida.

3.2. Diseño e Implementación

Si dividimos el proyecto en etapas podríamos decir que la primera etapa fue la de elección de herramientas y la realización de pruebas para verificar que la elección de las herramientas fue correcta.

Con las herramientas a utilizar ya definidas y con los objetivos claros nos queda por ver cómo vamos a realizar nuestro estudio de manera eficiente. Hasta el momento hemos visto que el realizar una simple emulación con una topología

sencilla requiere de mucho tiempo ya que se necesita realizar la configuración de toda la red de un Sistema Autónomo, con todo lo que esto implica. Por lo tanto, la siguiente etapa de nuestro trabajo consistirá en desarrollar las herramientas necesarias para que, junto con las elegidas antes, podamos realizar un estudio completo aprovechando al máximo nuestro tiempo. En el diagrama de la figura 3.2 se trata de mostrar, de manera sencilla, qué es lo que necesitamos desarrollar, cómo va ser la integración de todas la herramientas y cuál es la funcionalidad de las mismas.

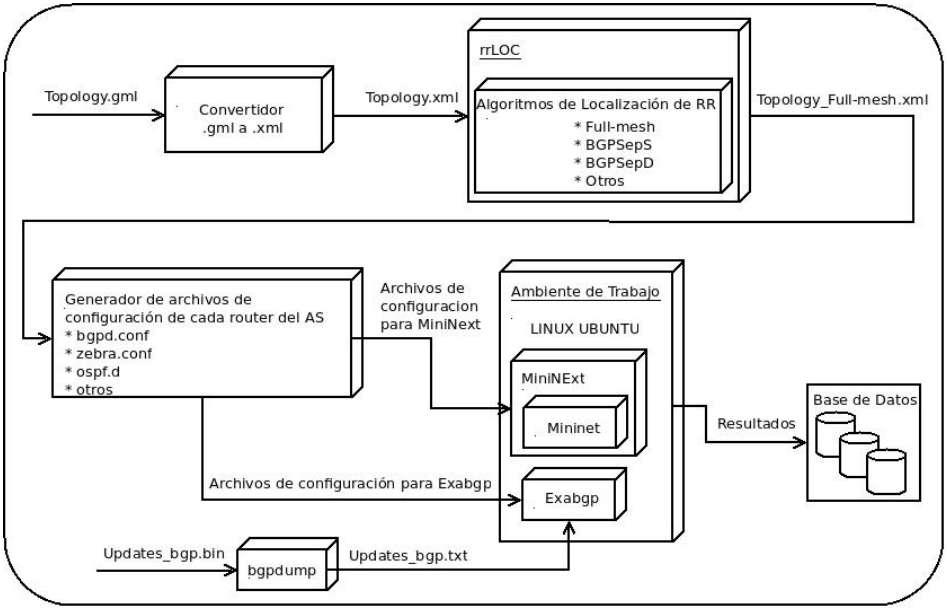


Figura 3.2: Proceso de Emulación

Digamos que el proceso para llevar a cabo una emulación consta de los siguientes pasos:

- Como primera instancia se debe obtener una topología. La misma debe tener el formato adecuado para que sirva como entrada en la herramienta RRLoc. Existen distintas páginas donde se pueden conseguir topologías reales de Sistemas Autónomos de Internet. El formato más común para describir una red que se extiende por diferentes zonas geográficas es el formato GML (*Geo-*

graphy Markup Language), el cual utiliza una gramática XML para modelar información geográfica. Para estos casos, necesitaremos algo que convierta el archivo GML en uno que sea aceptado por la herramienta RRLoc.

- Luego simplemente necesitamos ejecutar RRLoc con la topología en el formato adecuado como entrada. El resultado será un archivo con extensión xml con las sesiones iBGP definidas según el algoritmo que se quiere estudiar.
- Ahora entramos en la parte más complicada o más tediosa que es configurar toda la red. Como hemos mencionado anteriormente esto requiere definir todos los enlaces entre cada router, asignar las direcciones IPv4 e IPv6 a cada interfaz, configurar OSPF, configurar BGP según el algoritmo que se desea estudiar y, por último, definir la ubicación de los archivos donde se va almacenar la información relevante. Este trabajo hay que realizarlo para cada uno de los routers del AS, el cual puede tener cientos.
- Antes de poder realizar la emulación debemos obtener las trazas BGP y convertirlos a texto plano para poder así inyectar los mensajes con la ayuda de la herramienta Exabgp.
- Una vez la topología definida pasamos a emular la red. La emulación consistirá en establecer las sesiones iBGP, correr el protocolo de ruteo OSPF y luego inyectar trazas BGP desde diferentes routers de borde durante una hora.
- Finalmente, los resultados serán almacenados en una base de datos para su posterior análisis.

El conjunto de herramientas que hemos desarrollado para este proyecto conforman un *framework* que facilita las pruebas en un ambiente virtual que se asemeja mucho a la realidad. Si bien este *framework* está desarrollado pensado en el estudio del comportamiento de BGP, es fácilmente adaptable para el estudio de otros protocolos de red o, incluso, para otros tipos de estudios independientemente del protocolo que se utiliza. Por ejemplo, la carga de determinados enlaces ante posibles fallas en la red, como puede ser la falla de un equipo o la caída de un enlace.

Cabe destacar que el desarrollo de las herramientas se realizó de tal forma que sea compatible con la herramienta RRLoc pero que no dependa de la misma. Con esto queremos decir que podemos integrar distintos algoritmos de localización de Reflectores de Rutas a nuestro *framework*, aunque no hayan sido implementados por la herramienta RRLoc.

3.2.1. Traducción gml a xml

Como mencionamos anteriormente, hemos obtenido una gran número de topologías reales en "The topology Zoo" [46] para poder realizar las pruebas. Los archivos que describen dichas topologías son archivos .gml (GML), los cuales contienen todos los datos necesarios para poder correr los algoritmos de localización. Pero estos datos tienen un formato distinto al aceptado por la herramienta RRLoc. Debido a la gran pérdida de tiempo que puede consumir la traducción de cada uno de los archivos gml decidimos automatizar esta tarea. El script *GMLtoXMLconverter.py* realiza justamente esta traducción. El único parámetro de entrada que se necesita es la topología en formato .gml y devuelve su correspondiente traducción a .xml. Con el archivo .xml somos capaces de correr los algoritmos de RRLoc [31]. Además, dado que la herramienta MiniNExT sólo acepta nombres de nodos de hasta 9 caracteres y sin espacios, este script se encarga de recortar y/o quitar los espacios en los nombres de nodos que no cumplan con estos requerimientos.

3.2.2. Generar el ambiente de emulación

Teniendo ya la topología con una determinada implementación de BGP, podemos utilizar nuestro *framework* para generar el ambiente de pruebas correspondiente, con todo lo que esto implica. El script llamado *emulGen.py* es el encargado de generar toda la infraestructura con la configuración de cada elemento que interviene en las pruebas, desde routers, enlaces, sesiones, demonios, protocolos, hasta la inyección de las trazas BGP. A partir del archivo .xml resultado de la aplicación del algoritmo en la herramienta RRLoc, se generan todos los archivos necesarios para la emulación. Además, el script se encarga de asignar todas las direcciones IPv4 e IPv6 para los enlaces entre nodos (dado que RRLoc sólo asigna direcciones IPv4 a las interfaces de *loopback*).
Básicamente, el script genera dos archivos para la herramienta MiniNExT, dos archivos por cada router inyector y una carpeta por cada router de la topología con 5 archivos de configuración cada una.
Los archivos necesarios para la herramienta MiniNExT son los siguientes: *start.py* y *topo.py*. El archivo *topo.py* contiene la información relacionada con la topología. En este archivo se define la ruta para acceder a las carpetas de los routers (ubicados en la carpeta "*.../configs/*"), los nombres con los que se identifican los routers y sus correspondientes direcciones IPv4 de *loopback*, los enlaces que existen entre los routers y las rutas hacia los archivos de configuración de cada router de la topología. El archivo *start.py* inicializa la topología y levanta la red en MiniNExT. Además, luego de cargar la red entra en modo consola, donde si se escribe

la palabra *exit* se levanta el *exabgp* para cada uno de los routers inyectores y se comienza la inyección de rutas. Una vez que se escribe la palabra *exit*, se sale del modo consola para luego entrar en este modo nuevamente. Una vez pasada la hora de inyección y luego de que converge el protocolo BGP se realiza el último *exit* en donde se sale del modo consola, se guardan las tablas *Loc-RIB_ ipv4* y *Loc-RIB_ ipv6*, y se termina la emulación en MiniNExT.
Por cada router inyector se generan dos archivos: *announce-routes-from_ X.py* y *archivo-X.conf*, donde X es un número entero a partir de 1 y sirve para distinguir el router inyector. El archivo *archivo-X.conf* define la dirección IP y el número de AS del vecino BGP al que se le van a inyectar las rutas (perteneciente a la topología que se está probando), el id del router inyector, su dirección IP y su número de AS. Además, se especifica el directorio donde se encuentra el proceso de *python* para inyectar las rutas. El archivo *announce-routes-from_ X.py* define el proceso que inyecta la rutas. Dentro del mismo, se especifica dónde se encuentra el archivo con las trazas a inyectar, luego se recorre este archivo y se van inyectando una a una las trazas, cambiando el *next-hop* por la dirección IP del router inyector y agregando en el *AS-Path* el número de AS del mismo.
Finalmente, se tiene una carpeta por cada router. Cada una de estas carpetas está identificada con el nombre del router y contiene 5 archivos: *bgpd.conf*, *daemons*, *debian.conf*, *ospfd.conf* y *zebra.conf*.

- El archivo *daemons* contiene la lista de demonios de *quagga* que deben ser iniciados. En nuestro caso, éstos demonios son: *bgpd*, *zebra* y *ospfd*.

- El archivo *bgpd.conf* contiene la configuración para el demonio de BGP. Entre los parámetros que se definen en este archivo están las direcciones IP de *loopback* y número de AS de los vecinos que mantienen una sesión BGP con el nodo (puede ser una sesión iBGP o una eBGP). Además, se define dónde se guardan los datos de la tabla *RIB_ IN* y la tabla *table_ dump*, y cada cúanto tiempo.

- El archivo *debian.conf* contiene la configuración para poder acceder remotamente a los demonios *zebra*, *bgpd* y *ospfd*.

- El archivo *ospfd.conf* contiene la configuración del protocolo OSPF. En este archivo definimos el parámetro "ospf router id" el cual es el identificador del router dentro del protocolo OSPF. En la misma red no puede existir un router-id repetido, de lo contrario puede generar problemas de ruteo. Además, en este archivo archivo también definimos las redes que se van a enseñar a través del protocolo OSPF.

- El archivo *zebra.conf* contiene la configuración para el demonio *zebra*. En este archivo definimos la interfaces del router y le asignamos las direcciones IP a cada una.

Además de las configuraciones explicadas antes, el script *emulGen.py* puede recibir un archivo de texto con los costos de los enlaces. Este archivo contiene en cada línea un link (en formato router1-router2) y el valor del costo del enlace separado por un espacio. Por defecto, el costo de todos los enlaces vale 10.

3.2.3. Base de Datos

Una vez realizada la emulación en Mininet es hora de analizar los resultados obtenidos. Estos resultados son almacenados en la carpeta */var/log/mininext*. En esta ubicación existe una carpeta con el nombre de cada router de la topología, y dentro de cada una de estas carpetas están los datos que nos interesa analizar. Para nuestro estudio almacenamos cuatro archivos por router:

- *rib_ in*: Contiene todos los mensajes BGP recibidos por el router.
- *loc_ rib_ ipv4*: Contiene los prefijos IPv4 aprendidos por BGP luego de aplicar el algoritmo de decisión de BGP.
- *loc_ rib_ ipv6*: Contiene los prefijos IPv6 aprendidos por BGP luego de aplicar el algoritmo de decisión de BGP.
- *table_ dump*: Contiene la tabla de ruteo del router en un determinado punto del tiempo.

Particularmente los archivos *rib_ in* y *table_ dump* son generados por *Quagga* y están en formato binario. Para su análisis es necesario convertirlos a texto plano. Gracias a la herramienta *bgpdump* esto es posible. *bgpdump* es una librería de C que traduce los archivos binarios generados por *Quagga* a formato legible.

A diferencia de los mensajes BGP, la Loc-RIB no se almacena en ningún archivo por lo que es necesario obtenerla de alguna manera. Para esto utilizamos la API de Python que provee MiniNExT, la cual brinda funciones que posibilitan la obtención de esta tabla. Como se explicó antes, la Loc-RIB la separamos en dos tablas: loc_rib_ipv4 y loc_rib_ipv6. Ambas tablas se obtienen utilizando los comandos de las figuras 3.3 y 3.4 por cada router de la topología.

Para poder realizar un mejor análisis, lo que se hizo fue crear una base de datos que contuviera todos los mensajes descritos antes. Si bien uno puede ver la cantidad de mensajes que recibe cada router a través de un archivo, es necesario,

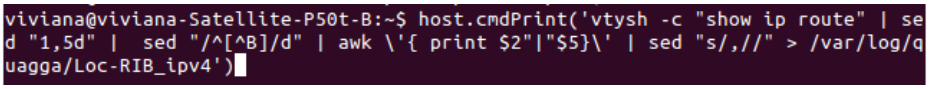


Figura 3.3: Comando para la obtención de la loc_rib_ipv4

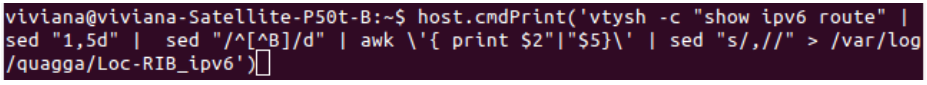


Figura 3.4: Comando para la obtención de la loc_rib_ipv6

para facilitar el trabajo de comparación, el cargar todos estos mensajes en una base de datos. Para esto, desarrollamos otra herramienta en *Python* que nos permite cargar estos valores en la base la cual denominamos *loadDB.py*. Este script toma todos los archivos generados en la emulación, los traduce con la herramienta *bgpdump* y los almacena en la base de datos. Este proceso puede demorar varias horas dependiendo de la cantidad de mensajes que se hayan enviado. Teniendo los datos de esta forma, se pueden hacer distintos tipos de consultas sin recorrer todos los archivos para comparar los resultados.

Utilizamos *mysql* como Sistema de Gestión de Base de Datos. Los datos los almacenamos en dos discos externos de 1TB cada uno.
Con el fin de separar los datos obtenidos para cada topología, decidimos crear una base de datos por topología, y cuatro tablas por implementación de BGP. Las tablas tienen los mismos nombres que los archivos en donde se almacenan los datos y se les agrega el nombre del algoritmo al final. Los atributos que contienen cada una de estas tablas con los siguientes:

- rib_in_*algoritmo*:
  - *versionBGP*: Indica el tipo de mensaje recibido (BGP4MP, TABLE_DUMP, OSPFv2, etc).
  - *date*: Indica la fecha y hora en que se recibió el mensaje.
  - *mesgType*: Indica el tipo de mensaje BGP recibido (update o withdraw).
  - *ipSource*: La dirección IP del router que envía el mensaje.
  - *peerAS*: Número de Sistema Autónomo del vecino que enseñó la ruta.
  - *prefix*: El prefijo enseñado.
  - *aspath*: El camino de Sistemas Autónomos por el cual ha pasado el anuncio del prefijo.

- *origin*: Indica la dirección IP del router que origina el anuncio.
- *router*: Indica el nombre del router que recibió el mensaje.

- table_dump_*algoritmo*:
  - *tableDump*: Indica el tipo de mensaje recibido (BGP4MP, TABLE_DUMP, OSPFv2, etc).
  - El resto de los atributos son iguales a los de la tabla rib_in.
- loc_rib_ipv4_*algoritmo* y loc_rib_ipv6_*algoritmo*:
  - *prefix*: Prefijo aprendido.
  - *next-hop*: Siguiente salto hacia el prefijo aprendido.
  - *router*: Nombre del router que recibió el prefijo.

Una de las primeras comparaciones que hicimos fue con la topología "Abilene", para cual creamos una base con el nombre "abilene". Con esta topología probamos una implementación con *full-mesh* y una con RR para las cuales creamos las tablas: "loc_rib_ipv4_FM", "loc_rib_ipv6_FM", "rib_in_FM", "table_dump_FM", "loc_rib_ipv4_RR_Sep", "loc_rib_ipv6_RR_Sep", "rib_in_RR_Sep" y "table_dump_RR_Sep". Los resultados que obtuvimos (inyectando trazas BGP a través de tres routers externos) fueron los siguientes:

Implementación	Mensajes BGP enviados	Prefijos IPv4 aprendidos	Prefijos IPv6 aprendidos
full-mesh	10.146.273	16.562	11.044
RR_Sep	23.817.261	16.562	11.044

Tabla 3.1: Tabla comparativa de topología Abilene

Como se aprecia, la cantidad de mensajes para el caso de una implementación con reflectores de rutas es más del doble que la cantidad de mensajes para el caso con *full-mesh*. Esto tiene sentido ya que en el caso del *full-mesh* por cada mensaje que le llega al router de borde, se enviará un mensaje a cada uno del resto de los routers (como máximo). Para el caso de los RRs, por cada mensaje que le llegue al router de borde, éste le enviará un mensaje a cada Reflector de Rutas, y a su vez, cada uno de éstos RRs le enviarán un mensaje a cada uno de sus clientes. Debido a que todos los routers son clientes de más de un RR, cada router (no RR) recibe la misma ruta, por intermedio de distintos RRs.

Con este resultado, se podría pensar que la cantidad de mensajes que se envían en redes donde se implementan los RRs siempre va a ser más que en una caso donde se utiliza el clásico modelo BGP con *full-mesh*, pero al sacar esta conclusión nos estaríamos apresurando ya que hay varios factores que aún no estamos considerando.
Otros aspectos importantes que tenemos que evaluar son el tamaño de las tablas de ruteo (*Routing Information Base* o RIB), y la cantidad de rutas alternativas que tenemos para alcanzar un mismo prefijo.

Capítulo 4

Pruebas y Resultados

En este capítulo hablaremos de las diferentes emulaciones que se realizaron y cuáles fueron los resultados obtenidos. Además, analizaremos el comportamiento de las distintas configuraciones de BGP, compararemos la cantidad de sesiones, la cantidad de mensajes enviados, los tiempos de convergencia y otros aspectos importantes para cada uno de los algoritmos de localización de Reflectores de Rutas.
El estudio se realizará emulando cada una de las topologías de red en el software MiniNExT durante una hora.

4.1. Pruebas

Cada emulación consistió en la inyección de trazas BGP reales, obtenidas de la página del Centro de Coordinación de redes IP europeas [42], durante una hora. Cada traza contiene cientos de miles de *updates* y *withdraws*. Uno se puede preguntar cuántas trazas BGP se deberían inyectar para emular la realidad de un AS o en cuáles routers del AS se deben establecer las sesiones eBGP. Afortunadamente para nosotros, la gran mayoría de las topologías en "The Topology Zoo"[46] tienen esta información. Por ejemplo, para el caso de Cesnet, en la imagen 4.1, se puede ver que mantiene siete conexiones con diferentes ASes por intermedio de distintos medios físicos, con distintas velocidades de transferencia.

Es importante aclarar, que en la emulación de los algoritmos de una misma topología, las trazas a inyectar son las mismas.

Las topologías utilizadas para desarrollar las emulaciones son:

- Abilene
- Forthnet
- Istar
- UsCarrier
- Cesnet
- CWIX
- Garr
- COX
- Colt
- DialTelecom
- Carnet

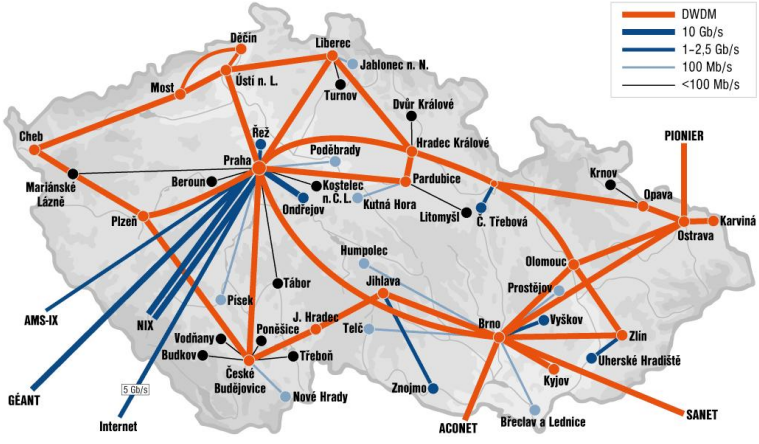


Figura 4.1: Cesnet Network

Se llevo a cabo una emulación por cada algoritmo de localización. Los algoritmos utilizados son:

- *full-mesh*
- BGP Sep
- BGP SepD
- BGP SepS
- BGP Bates
- BGP BatesY
- BGP BatesZ
- BGP Zhang

Los datos de las topolgías como cantidad de nodos, cantidad de routers inyectores y prefijos IPv4 e IPv6 aprendidos por nodo se pueden ver en la tabla 4.1

Topología	Cantidad de routers	Cantidad de routers inyectores	Prefijos IPv4 aprendidos por nodo	Prefijos IPv6 aprendidos por nodo
Abilene	12	3	16.562	11.044
Forthnet	60	3	16.562	11.044
Istar	19	4	20.461	11.064
UsCarrier	158	3	5.047	408
Cesnet	45	7	20.554	18.263
CWIX	24	12	185.896	22.142
Garr	48	13	187.516	22.282
COX	32	3	16.562	11.044
Colt	153	5	17.246	607
DialTelecom	201	2	2.115	57
Carnet	41	3	16.562	11.044

Tabla 4.1: Información sobre las topologías

4.1.1. Verificación de los resultados obtenidos

Una vez finalizada cada emulación realizamos algunas verificaciones de manera de saber si los resultados que obtuvimos son coherentes con los datos de entrada

que se suministraron. Por ejemplo, dadas dos topologías A y B, a las cuales se les inyectan las mismas trazas BGP, se debe cumplir que las tablas de ruteo de cada router deben contener la misma cantidad de prefijos. Lo mismo ocurre entre distintas implementaciones de BGP en una misma topología. Si realizamos una emulación con una topología A donde se implementó un *full-mesh* de sesiones iBGP y otra emulación con la misma topología pero distinta implementación de BGP, los prefijos aprendidos deben ser los mismos. De no ser así, se debe estudiar dónde ocurre la falla, solucionarla y volver a correr la emulación. Cabe aclarar que no necesariamente las tablas de ruteo son la mismas debido a que el *next-hop* puede variar entre distintas topologías o distintas implementaciones de BGP.

En la etapa de verificación, entra en juego un punto que nos parece importante a destacar que es la utilización de direcciones IP privadas y el uso de números de Sistemas Autónomos (ASN) privados en las topologías que hemos utilizado para las pruebas. ¿Por qué nos parece importante? Porque de antemano no sabemos cuáles son las direcciones que se aprenden por intermedio de BGP, ni tampoco cuáles son los ASN que forman los AS-PATH de cada *update* BGP. Si llegáramos a utilizar alguna IP o ASN repetido esto alteraría el resultado. Veamos un ejemplo: Si utilizamos un ASN que ya es utilizado por otro Sistema Autónomo y una de las actualizaciones BGP que nos llega contiene este ASN, el router al que le llegue descartará la ruta anunciada, y si esta ruta es la única ruta que se enseña para alcanzar un determinado prefijo “p”, la tabla de ruteo de cada router del AS no contendrá dicho prefijo “p”.

4.2. Cantidad de Sesiones iBGP

Comenzaremos nuestro análisis comparando la cantidad de sesiones iBGP. Como se puede ver en las gráficas 4.2, 4.3 y 4.4, y en las tablas 4.2 y 4.3, la cantidad de sesiones iBGP disminuye considerablemente cuando se utilizan reflectores de rutas, excepto en el caso del algoritmo Bates.

Como se explico anteriormente, la implementación del algoritmos “Bates” considera la topología IGP entera como un solo PoP. Esto implica que igualmente exista un *full-mesh* de sesiones iBGP y además se definan reflectores de rutas. Claramente este tipo de configuración no aporta ningún beneficio. No obstante, se realizaron algunas pruebas con este algoritmo de manera de verificar la correctitud del mismo.

La brecha entre la cantidad de sesiones en una configuración de BGP con *full-mesh* y una con reflectores de rutas se agranda a medida que el tamaño de la topología aumenta. En particular, las heurísticas “BatesY”, “BatesZ” y “Zhang”

	full-mesh	BGP Sep	BGP SepD	BGP SepS
Abilene	66	34	31	41
Forthnet	1770	155	59	59
Istar	171	54	19	19
UsCarrier	12403	3289	2480	3009
Cesnet	990	190	105	123
CWIX	276	102	85	209
Garr	1128	192	116	123
COX	496	168	134	124
Colt	11628	2367	995	2553
DialTelecom	20100	5941	5240	14425
Carnet	820	68	40	40

Tabla 4.2: Cantidad de sesiones iBGP por topología, por algoritmo - Parte 1

	BGP Bates	BGP BatesY	BGP BatesZ	BGP Zhang
Abilene	66	31	22	17
Forthnet	1770	582	123	123
Istar	171	84	36	31
UsCarrier	12403	1585	404	484
Cesnet	990	322	98	143
CWIX	276	134	46	41
Garr	1128	367	99	99
COX	496	210	67	67
Colt	11628	1526	394	534
DialTelecom	20100	1208	1102	2052
Carnet	820	288	85	85

Tabla 4.3: Cantidad de sesiones iBGP por topología, por algoritmo - Parte 2

son las que logran mejores resultados considerando sólo la cantidad de sesiones iBGP.

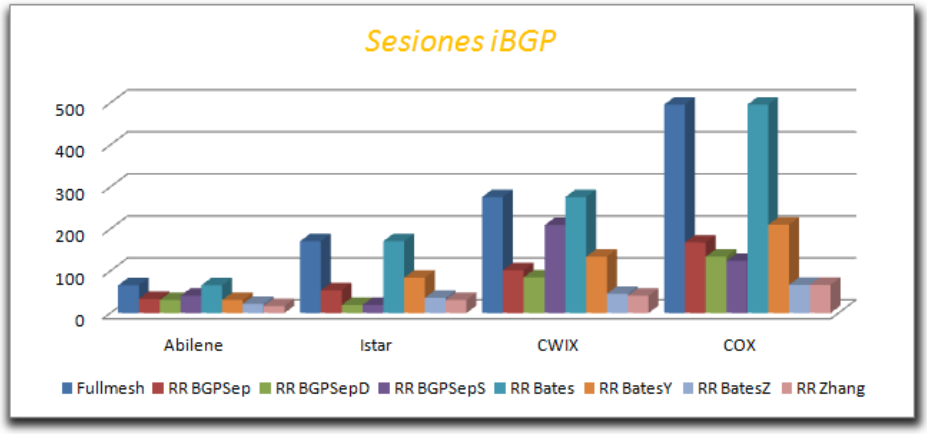


Figura 4.2: Cantidad de Sesiones iBGP (1)

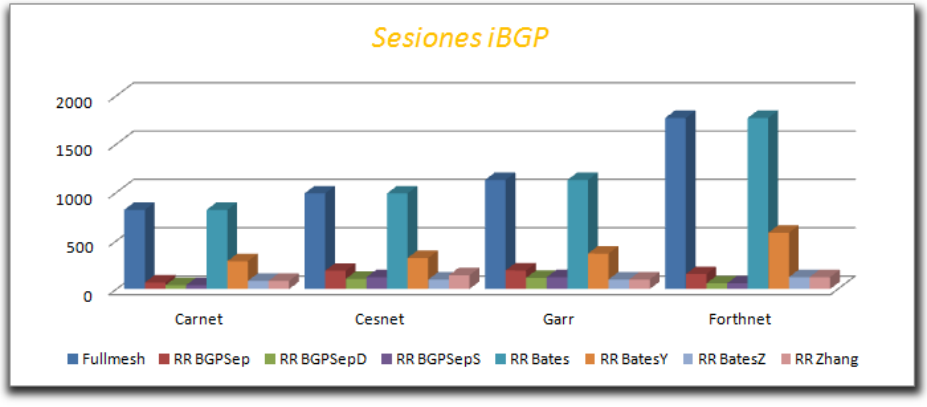


Figura 4.3: Cantidad de Sesiones iBGP (2)

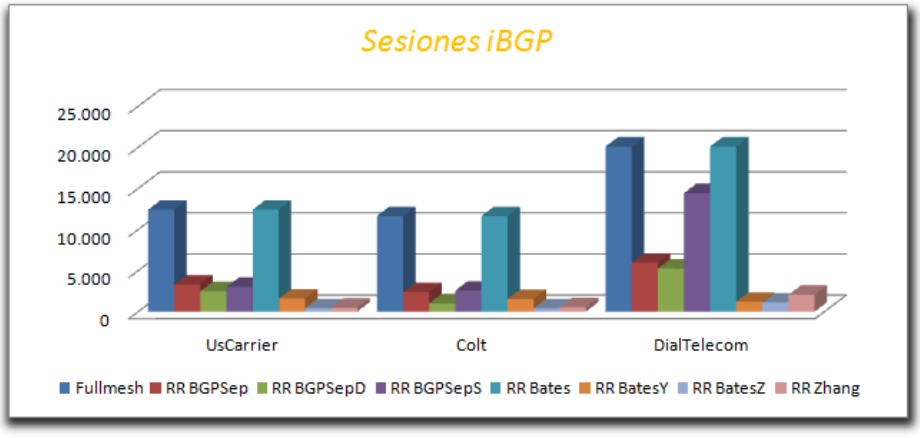


Figura 4.4: Cantidad de Sesiones iBGP (3)

4.3. Cantidad de Reflectores de Rutas

Un punto importante que hay que tener presente a la hora de analizar los datos es la cantidad de RRs que se definieron en cada prueba. Esto nos ayuda a entender cómo es el comportamiento de los algoritmos con las distintas topologías y nos da un indicio de si existe una relación entre la cantidad de reflectores de rutas y la cantidad de mensajes que son transmitidos. Si observamos la tabla 4.4 vemos que los algoritmos BGP SepD y BGP SepS son los que definen mayor cantidad de RRs. Sin embargo, al contrario de lo que uno puede pensar, éstos generan mayores cantidad de sesiones que las heurísticas estudiadas (Bates, BatesY, BatesZ y Zhang). Además, hay que agregar que en muchos casos los algoritmos BGP SepD y BGP SepS no definen sesiones redundantes para todos los routers del AS. Teniendo sólo esto en cuenta, podemos afirmar que las heurísticas definen una mejor configuración BGP con respecto al resto de los algoritmos.

4.4. Cantidad de mensajes BGP enviados

Uno de los focos principales de este estudio es la comparación de la cantidad de mensajes BGP que son enviados en las diferentes configuraciones. Si bien existen varias tesis y artículos publicados (como por ejemplo [17]) que aseguran que las to-

	BGP Sep	BGP SepD	BGP SepS	BGP Bates	BGP BatesY	BGP BatesZ	BGP Zhang
Abilene	5	6	5	3	2	4	6
Forthnet	9	10	10	3	3	6	13
Istar	5	7	7	3	2	4	6
UsCarrier	62	74	62	-	8	16	60
Cesnet	12	15	17	3	3	6	16
CWIX	7	11	8	3	2	4	6
Garr	12	18	19	3	3	6	18
COX	12	14	13	3	3	6	14
Colt	42	53	46	-	8	16	68
DialTelecom	81	87	37	-	20	40	77
Carnet	4	9	9	3	3	6	15

Tabla 4.4: Cantidad de Reflectores de Rutas por topología y algoritmo.

pologías iBGP con reflectores de rutas disminuyen significativamente la cantidad de mensajes, las argumentaciones que dan no son suficientes para probar dicha afirmación.

Uno se puede preguntar, ¿cómo puede ser posible que la cantidad de mensajes BGP que se envían disminuya utilizando reflectores de rutas si a cada nodo de la red le puede llegar un mismo *update* a través de distintas sesiones?, algo que no pasa en una configuración *full-mesh*. La argumentación de los autores que han estudiado el tema es que la cantidad de mensajes que recibe un router cliente de un reflector de rutas es significativamente menor. Esto es debido a que los reflectores de rutas reciben los *updates* por sus vecinos pero sólo anuncian las mejores rutas (según el proceso de decisión de BGP) a sus clientes. Por lo tanto, un reflector de rutas sólo anunciará aquellos *updates* que cambien la selección del mejor camino. Algo similar pasaría con los mensajes *withdraw*, dado que no es necesario enviar un *withdraw* de una ruta que no fue seleccionada como la mejor.

Pero entonces, ¿disminuye o no la cantidad de mensajes BGP que se envían?. ¿Es significativamente menor la cantidad de mensajes que se envían?. Observando las gráficas 4.5 y 4.6 vemos que varía significativamente entre un algoritmo y otro, pero en casi todos los casos la configuración *full-mesh* es la que mantiene mejores resultados.

Los algoritmos BGP Sep, BGP SepD y BGP SepS arrojan resultados para nada alentadores. Estos algoritmos generan un aumento en la cantidad de mensajes

significativo. Este aumento, como se detallará más adelante, tiene un impacto negativo en la *performance* de la red. Por otro lado, la heurística BatesY mantuvo valores cercanos a los obtenidos en las configuraciones *full-mesh*. Los valores exactos obtenidos en las pruebas se pueden ver en las tablas 4.5 y 4.6

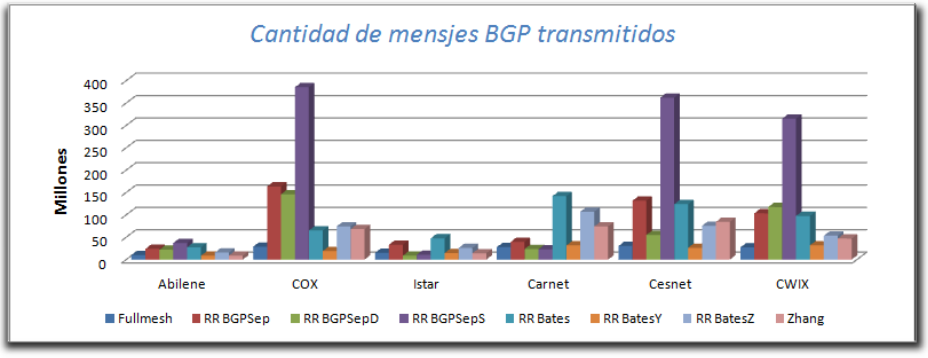


Figura 4.5: Cantidad de mensajes transmitidos (1)

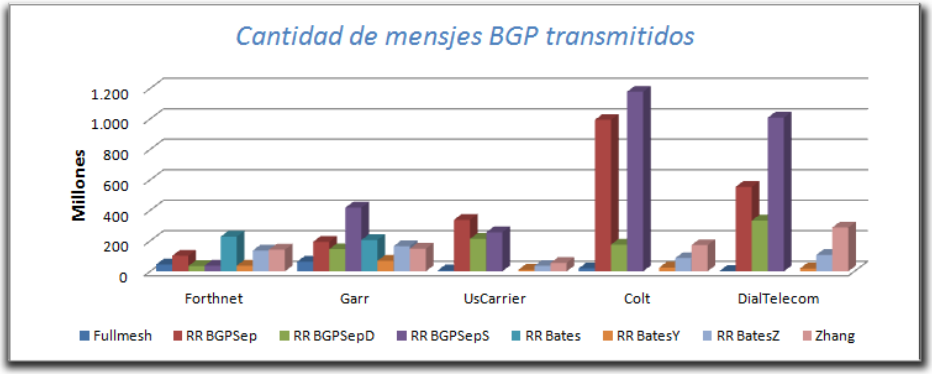


Figura 4.6: Cantidad de mensajes transmitidos (2)

	full-mesh	BGP Sep	BGP SepD	BGP SepS
Abilene	10.146.273	23.817.261	21.890.710	36.753.873
Forthnet	45.414.523	102.497.741	34.464.441	37.095.347
Istar	14.887.624	33.217.029	8.663.796	10.789.536
UsCarrier	6.495.531	337.706.786	213.055.581	255.392.084
Cesnet	30.412.924	131.550.170	55.109.674	361.943.837
CWIX	27.116.481	103.266.060	117.347.179	315.065.143
Garr	62.462.301	193.726.946	145.226.043	417.523.018
COX	28.850.181	163.462.722	146.101.673	385.294.349
Colt	19.471.091	993.520.984	174.556.542	1.178.093.489
DialTelecom	2.052.378	554.306.286	333.055.973	1.008.767.123
Carnet	27.720.063	39.492.893	23.747.228	22.957.619

Tabla 4.5: Cantidad de mensajes BGP enviados por topología, por algoritmo - Parte 1

	BGP Bates	BGP BatesY	BGP BatesZ	BGP Zhang
Abilene	27.557.717	8.619.573	15.681.048	8.832.542
Forthnet	227.013.040	36.833.218	136.636.820	141.782.175
Istar	47.484.237	14.559.674	25.799.427	14.140.755
UsCarrier	-	10.061.800	35.776.797	55.426.686
Cesnet	123.820.735	26.299.996	75.527.345	84.589.933
CWIX	97.462.629	31.787.270	53.461.382	46.867.051
Garr	205.783.378	68.862.853	165.387.448	148.362.288
COX	64.967.494	19.079.871	74.265.174	68.054.019
Colt	-	24.720.164	86.565.346	171.720.978
DialTelecom	-	18.919.238	107.162.570	287.271.180
Carnet	141.737.928	31.565.732	107.034.879	73.876.190

Tabla 4.6: Cantidad de mensajes BGP enviados por topología, por algoritmo - Parte 2

4.4.1. Tamaño promedio de la tabla rib_in

Si bien hemos visto que la gran mayoría de las configuraciones con reflectores de rutas generan un aumento en la cantidad de mensajes trasmitidos con respecto a las topologías *full-mesh*, nos interesa comparar las tablas rib_in de los routers. En particular, nos interesa comparar las tablas rib_in de los RRs con las tablas de los que no lo son. Con lo que hemos visto hasta el momento del comportamiento de los RRs todo indica que la cantidad de mensajes que recibe un RR debe ser mucho mayor. También nos parece importante comparar la tabla rib_in de los routers en una topología *full-mesh* con los routers de que no son RRs en un topología con RRs.

A continuación haremos las comparaciones con los resultados obtenidos tomando como medida el promedio del tamaño de las tablas rib_in de los routers del AS. Las gráficas 4.7 y 4.8 muestran el tamaño promedio de la tabla rib_in de los routers que no actúan como reflectores de rutas, mientras que las gráficas 4.9 y 4.10 muestran el tamaño promedio para el caso de los reflectores. Estos resultados muestran que los algoritmos de localización RR BGPSep, RR BGPSepD y RR BGPSepS no logran el efecto esperado en una topología con RRs. El escenario es totalmente distinto con las heurísticas que si muestran que el promedio de mensajes recibidos en los nodos que son reflectores de rutas es mayor que en el caso de los nodos clientes. Como puede ver en las tablas 4.7, 4.8, 4.9 y 4.10, los RRs son los que reciben la mayor cantidad de mensajes y luego éstos filtran los anuncios que no provocan un cambio en la loc-rib. Como consecuencia, los clientes de estos RRs reciben menor cantidad de mensajes. En particular, en el algoritmo BatesY los routers clientes reciben en promedio menos mensajes que en una configuración *full-mesh*.

4.5. Tiempos de Propagación y promedio de anuncios repetidos

Cuando hablamos de tiempos debemos tener en cuenta que el ambiente de pruebas no es una ambiente real donde cada dispositivo de red cuenta con recursos propios y donde no existen *links* físicos como fibra óptica, cables UTP o antenas. En nuestro ambiente de pruebas los routers son dispositivos virtuales que comparten todos los recursos como memoria y procesador, y además, se comunican entre ellos mediante *links* virtuales. Por lo tanto, nos enfocaremos en la comparación de los resultados y no en los valores obtenidos ya que éstos pueden diferir de la realidad.

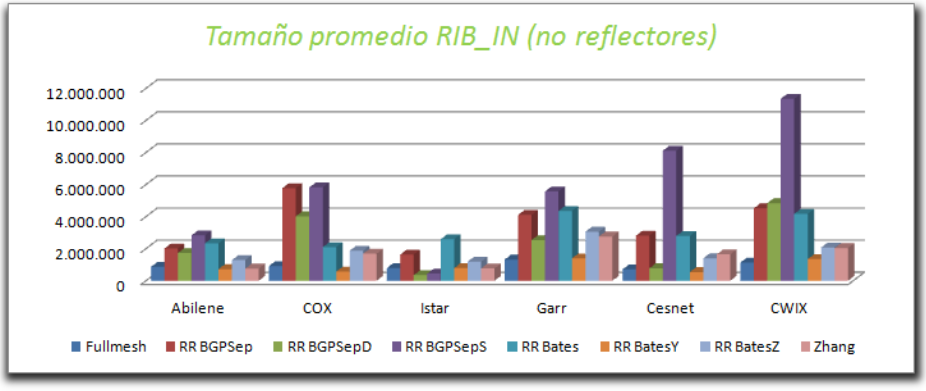


Figura 4.7: Tamaño promedio RIB_IN de nodos no reflectores (1)

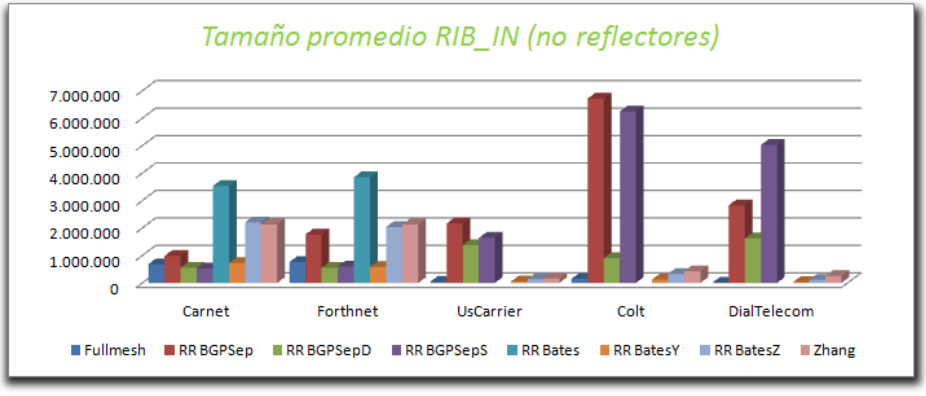


Figura 4.8: Tamaño promedio RIB_IN de nodos no reflectores (2)

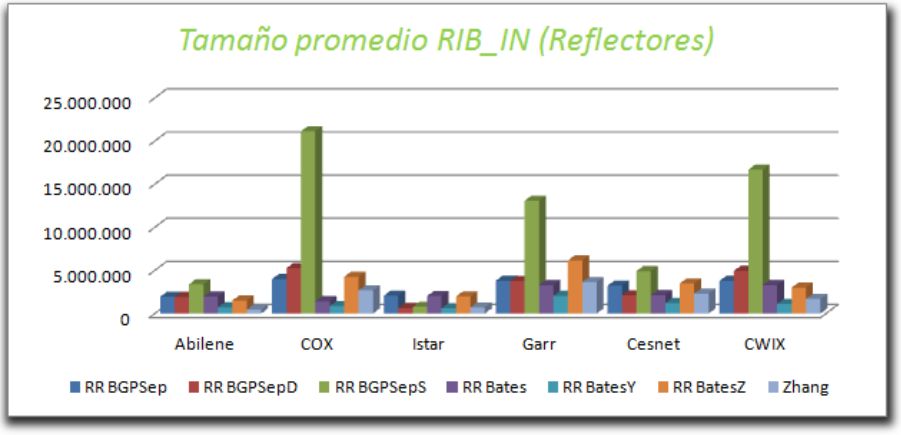


Figura 4.9: Tamaño promedio RIB_IN dereflcetores de rutas (1)

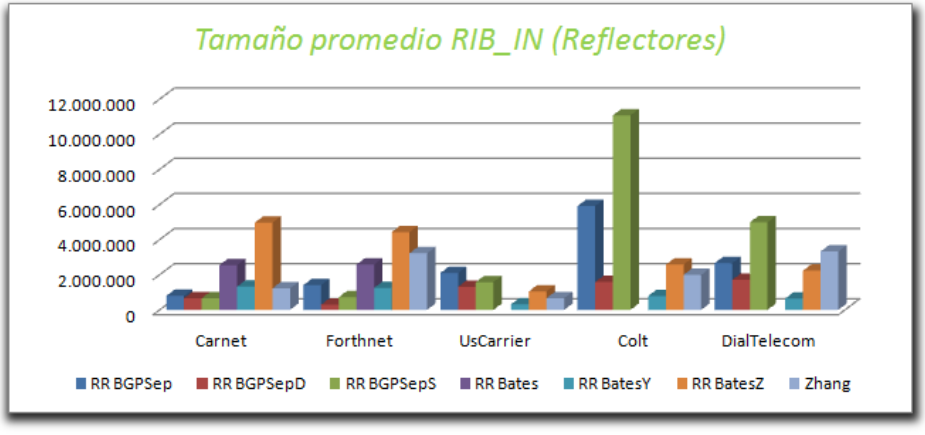


Figura 4.10: Tamaño promedio RIB_IN de reflectores de rutas (2)

	BGP Sep	BGP SepD	BGP SepS
Abilene	1.946.751	1.914.652	3.390.703
Forthnet	1.405.404	289.271	722.602
Istar	2.074.498	643.082	780.637
UsCarrier	2.100.522	1.314.183	1.577.666
Cesnet	3.228.505	2.117.919	4.919.339
CWIX	3.782.073	4.950.133	16.682.873
Garr	3.801.069	3.770.928	13.076.028
COX	3.996.275	5.268.891	21.128.566
Colt	5.908.520	1.580.940	11.077.502
DialTelecom	2.651.814	1.703.042	4.984.615
Carnet	792.625	665.549	662.065

Tabla 4.7: Promedio de la rib_in para el conjunto de Reflectores de Rutas - Parte 1

	BGP Bates	BGP BatesY	BGP BatesZ	BGP Zhang
Abilene	1.957.731	719.193	1.518.133	465.564
Forthnet	2.574.445	1.249.180	4.426.066	3.222.097
Istar	2.035.946	610.122	1.985.883	700.704
UsCarrier	-	319.015	1.051.937	668.680
Cesnet	2.137.149	1.212.720	3.483.674	2.308.988
CWIX	3.300.965	1.122.783	3.008.170	1.690.197
Garr	3.304.841	2.052.560	6.156.319	3.646.334
COX	1.416.986	868.589	4.255.233	2.694.877
Colt	-	790.236	2.573.846	1.992.704
DialTelecom	-	646.769	2.231.158	3.323.708
Carnet	2.544.716	1.320.951	4.964.939	1.224.550

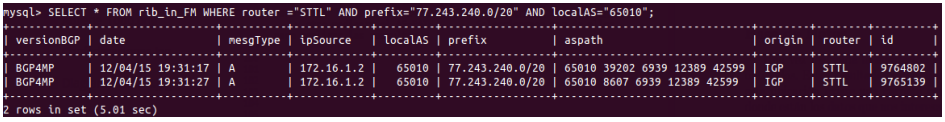
Tabla 4.8: Promedio de la rib_in para el conjunto de Reflectores de Rutas - Parte 2

	full-mesh	BGP Sep	BGP SepD	BGP SepS
Abilene	864.342	2.011.928	1.733.799	2.828.622
Forthnet	762.124	1.761.747	551.434	597.386
Istar	785.942	1.631.752	346.851	443.756
UsCarrier	47.366	2.161.191	1.378.642	1.641.424
Cesnet	703.598	2.812.366	778.029	8.114.193
CWIX	1.133.519	4.517.149	4.838.131	11.350.134
Garr	1.320.232	4.103.250	2.537.186	5.571.515
COX	912.736	5.775.370	4.018.732	5.822.262
Colt	146.480	6.714.983	907.667	6.247.928
DialTelecom	12.300	2.829.244	1.621.853	5.026.441
Carnet	683.605	981.686	554.915	531.219

Tabla 4.9: Promedio de la rib_in para el conjunto de NO Reflectores de Rutas - Parte 1

	BGP Bates	BGP BatesY	BGP BatesZ	BGP Zhang
Abilene	2.327.271	718.216	1.287.537	760.634
Forthnet	3.847.187	580.450	2.038.526	2.125.423
Istar	2.586.024	784.672	1.190.392	764.348
UsCarrier	-	50.064	133.421	156.182
Cesnet	2.795.459	539.567	1.400.648	1.642.969
CWIX	4.169.511	1.342.804	2.071.435	2.040.326
Garr	4.352.641	1.393.448	3.058.322	2.757.609
COX	2.093.673	568.072	1.874.376	1.684.763
Colt	-	126.884	331.268	426.083
DialTelecom	-	33.059	111.280	252.787
Carnet	3.529.046	726.391	2.207.006	2.134.920

Tabla 4.10: Promedio de la rib_in para el conjunto de NO Reflectores de Rutas - Parte 2



```
mysql> SELECT * FROM rib_in_FM WHERE router ="STTL" AND prefix="77.243.240.0/20" AND localAS="65010";
+------------+---------------------+----------+-----------+---------+-----------------+-----------------------------+--------+--------+---------+
| versionBGP | date                | mesgType | ipSource  | localAS | prefix          | aspath                      | origin | router | id      |
+------------+---------------------+----------+-----------+---------+-----------------+-----------------------------+--------+--------+---------+
| BGP4MP     | 12/04/15 19:31:17 | A        | 172.16.1.2 |   65010 | 77.243.240.0/20 | 65010 39202 6939 12389 42599 | IGP    | STTL   | 9764802 |
| BGP4MP     | 12/04/15 19:31:27 | A        | 172.16.1.2 |   65010 | 77.243.240.0/20 | 65010 8607 6939 12389 42599  | IGP    | STTL   | 9765139 |
+------------+---------------------+----------+-----------+---------+-----------------+-----------------------------+--------+--------+---------+
2 rows in set (5.01 sec)
```

Figura 4.11: Consulta BD por prefijo 77.243.240.0/20

En esta sección queremos mostrar los tiempos que demora un mensaje BGP determinado en propagarse a través de todo el Sistema Autónomo para los diferentes algoritmos implementados. Para esto, seleccionamos algunos prefijos que hayan sido aprendidos a través de sesiones eBGP y haremos consultas a la base de datos para determinar el tiempo desde que se aprendió dicho prefijo por uno de los routers de borde hasta que se propagó a todos los routers del AS.

De manera de ilustrar cómo se obtienen estos resultados, veamos el ejemplo de la red Abilene. Las pruebas en esta red pequeña que tiene tan solo doce routers consistieron en la inyección de tres trazas BGP en los routers "STTL", "NYCM" y "HSTN" para cada uno de los algoritmos estudiados.

Para empezar seleccionamos algunos anuncios que le llegan a los routers de borde a través de las sesiones eBGP, preferentemente alguno que no se repita. Por ejemplo, podemos seleccionar el anuncio de la red "77.243.240.0/20" que se recibe por la sesión eBGP entre el router "STTL" y el router de borde del AS 65010.

Como se puede ver en la imagen 4.11, se anuncian dos rutas distintas para llegar a la red "77.243.240.0/20": una a través de el AS-PATH [65010 39202 6939 12389 42599] y otra mediante el AS-PATH [65010 8607 6939 12389 42599]. Seleccionamos una de ellas para realizar las consultas.

Primero veamos el caso del *full-mesh*. Como se puede ver en la imagen 4.12 todos los routers del AS reciben este anuncio una única vez. El primero en recibir el mensaje es el router "STTL" por intermedio de la sesión eBGP que mantiene con el router de borde del AS 65010. Luego, éste trasmite dicho anuncio al resto de los routers mediante las sesiones iBGP. El tiempo que tardó en propagarse por todo el AS es igual $19:32:32 - 19:32:27$ o sea 5 segundos.

Ahora veamos el caso con Reflectores de Rutas localizados con el algoritmo BGPSep. A diferencia del caso *full-mesh*, como se puede ver en las imágenes 4.13 y 4.14, a cada uno de los routers les llega el anuncio repetido. Para entender porque pasa esto veamos la topología iBGP (imagen 4.15). Cuando el router "STTL" recibe el *update* con el prefijo ' "77.243.240.0/20" por intermedio de su *peer* eBGP, éste lo reenvía por intermedio de las sesiones iBGP a los reflectores de rutas "SNVA", "DNVR", "HSTN" y "IPLS". Luego, cada uno de éstos reenvía el prefijo a

```
mysql> SELECT * FROM rib_in_FM WHERE prefix="77.243.240.0/20" AND aspath="65010 8607 6939 12389 42599" ORDER BY date;
+-----------+-----------------+---------+------------+---------+---------------+-------------------------+--------+--------+---------+
| versionBGP | date            | mesgType | ipSource   | localAS | prefix        | aspath                  | origin | router | id      |
+-----------+-----------------+---------+------------+---------+---------------+-------------------------+--------+--------+---------+
| BGP4MP    | 12/04/15 19:31:27 | A       | 172.16.1.2 |   65010 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | STTL   | 9765139 |
| BGP4MP    | 12/04/15 19:31:30 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | HSTN   | 5505489 |
| BGP4MP    | 12/04/15 19:31:30 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | NYCM   | 4326030 |
| BGP4MP    | 12/04/15 19:31:31 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | KSCY   | 1704527 |
| BGP4MP    | 12/04/15 19:31:31 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | SNVA   | 7209207 |
| BGP4MP    | 12/04/15 19:31:31 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | WASH   | 8061163 |
| BGP4MP    | 12/04/15 19:31:31 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | LOSA   |     370 |
| BGP4MP    | 12/04/15 19:31:32 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLA   | 2556316 |
| BGP4MP    | 12/04/15 19:31:32 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | CHIN   | 3473929 |
| BGP4MP    | 12/04/15 19:31:32 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLAM5 |  852491 |
| BGP4MP    | 12/04/15 19:31:32 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | DNVR   | 6357445 |
| BGP4MP    | 12/04/15 19:31:32 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | IPLS   | 8913361 |
+-----------+-----------------+---------+------------+---------+---------------+-------------------------+--------+--------+---------+
12 rows in set (0.00 sec)
```

Figura 4.12: Resultado consulta a base por prefijo 77.243.240.0/20 y con AS-PATH=[65010 8607 6939 12389 42599]

```
mysql> SELECT * FROM rib_in_RR_Sep WHERE prefix="77.243.240.0/20" AND aspath="65010 8607 6939 12389 42599" ORDER BY date;
+-----------+-----------------+---------+------------+---------+---------------+-------------------------+--------+--------+----------+
| versionBGP | date            | mesgType | ipSource   | localAS | prefix        | aspath                  | origin | router | id       |
+-----------+-----------------+---------+------------+---------+---------------+-------------------------+--------+--------+----------+
| BGP4MP    | 12/04/15 21:09:54 | A       | 172.16.1.2 |   65010 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | STTL   | 22217187 |
| BGP4MP    | 12/04/15 21:09:56 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | IPLS   | 19989236 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.7 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | CHIN   |  8258672 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | DNVR   | 14287887 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | HSTN   | 12125087 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | DNVR   | 14287715 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | WASH   | 18023174 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.12 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | DNVR   | 14288104 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | SNVA   | 16122697 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | SNVA   | 16122754 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.7 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLA   |  6423369 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.7 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLAM5 |  2294664 |
| BGP4MP    | 12/04/15 21:09:57 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | LOSA   |     1101 |
| BGP4MP    | 12/04/15 21:09:58 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | KSCY   |  4195363 |
| BGP4MP    | 12/04/15 21:09:58 | A       | 192.168.0.1 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | DNVR   | 14288185 |
| BGP4MP    | 12/04/15 21:09:58 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | NYCM   | 10748792 |
| BGP4MP    | 12/04/15 21:10:00 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | CHIN   |  8258986 |
| BGP4MP    | 12/04/15 21:10:00 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLA   |  6423634 |
| BGP4MP    | 12/04/15 21:10:00 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLAM5 |  2294929 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.12 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | LOSA   |     1436 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.2 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | LOSA   |     1692 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | WASH   | 18023260 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | IPLS   | 19989628 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | SNVA   | 16123150 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | CHIN   |  8259057 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLAM5 |  2294992 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.12 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | KSCY   |  4195614 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.12 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | HSTN   | 12126035 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | HSTN   | 12125724 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.4 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | NYCM   | 10748951 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.7 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | NYCM   | 10749018 |
| BGP4MP    | 12/04/15 21:10:01 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLA   |  6423701 |
| BGP4MP    | 12/04/15 21:10:02 | A       | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | KSCY   |  4195730 |
```

Figura 4.13: Resultado de consulta a base por prefijo 77.243.240.0/20 [1/2] - Algoritmo BGPPSep

versionBGP	date	mesgType	ipSource	localAS	prefix	aspath	origin	router	id
BGP4MP	12/04/15 21:10:02	A	192.168.0.2	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	KSCY	4196190
BGP4MP	12/04/15 21:10:02	A	192.168.0.2	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	IPLS	19990040
BGP4MP	12/04/15 21:10:02	A	192.168.0.12	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	IPLS	19990226
BGP4MP	12/04/15 21:10:02	A	192.168.0.2	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	HSTN	12126262
BGP4MP	12/04/15 21:10:02	A	192.168.0.2	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	SNVA	16123675
BGP4MP	12/04/15 21:10:02	A	192.168.0.10	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	LOSA	2014
BGP4MP	12/04/15 22:23:48	A	192.168.0.4	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	SNVA	17969017
BGP4MP	12/04/15 22:23:48	A	192.168.0.10	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	DNVR	16087565
BGP4MP	12/04/15 22:23:48	A	192.168.0.4	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	DNVR	16081943
BGP4MP	12/04/15 22:23:48	A	192.168.0.10	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	NYCM	12040935
BGP4MP	12/04/15 22:23:48	A	192.168.0.10	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	IPLS	22158233
BGP4MP	12/04/15 22:23:48	A	192.168.0.4	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	KSCY	6391826

45 rows in set (56.49 sec)

Figura 4.14: Resultado de consulta a base por prefijo 77.243.240.0/20 [2/2] - Algoritmo BGPPSep

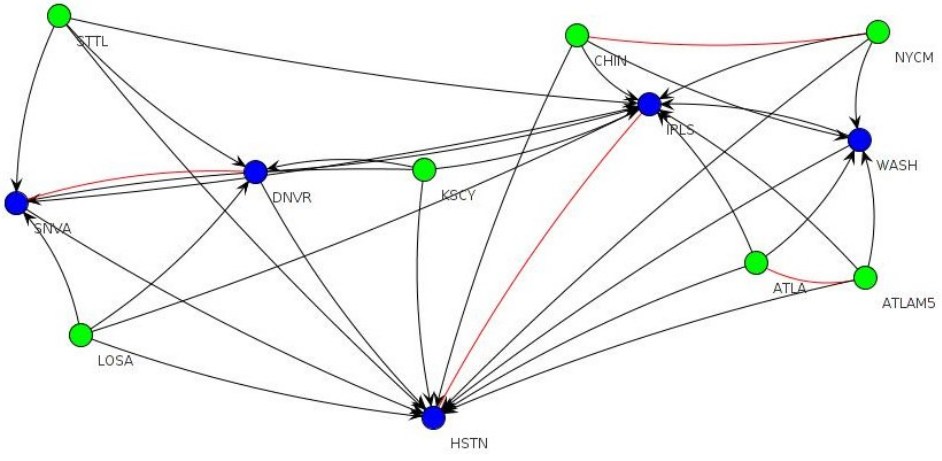


Figura 4.15: Sesiones iBGP topología Abilene - Algoritmo BGPSep

```
mysql> SELECT * FROM rib_in_RR_Sep WHERE prefix="77.243.240.0/20" AND aspath="65010 8607 6939 12389 42599"  AND router="LOSA";
```

versionBGP	date	mesgType	ipSource	localAS	prefix	aspath	origin	router	id
BGP4MP	12/04/15 21:09:57	A	192.168.0.4	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	LOSA	1101
BGP4MP	12/04/15 21:10:01	A	192.168.0.12	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	LOSA	1436
BGP4MP	12/04/15 21:10:01	A	192.168.0.2	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	LOSA	1692
BGP4MP	12/04/15 21:10:02	A	192.168.0.10	65000	77.243.240.0/20	65010 8607 6939 12389 42599	IGP	LOSA	2014

Figura 4.16: Resultado de consulta a base por prefijo 77.243.240.0/20 router LOSA - Algoritmo BGPPSep

```
mysql> SELECT * FROM rib_in_RR_Sep WHERE prefix="77.243.240.0/20" AND aspath="65010 8607 6939 12389 42599" GROUP BY router ORDER BY date ASC;
+------------+------------------+----------+--------------+---------+----------------+-------------------------------+--------+--------+----------+
| versionBGP | date             | mesgType | ipSource     | localAS | prefix         | aspath                        | origin | router | id       |
+------------+------------------+----------+--------------+---------+----------------+-------------------------------+--------+--------+----------+
| BGP4MP     | 12/04/15 21:09:54 | A        | 172.16.1.2   |   65010 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | STTL   | 22217187 |
| BGP4MP     | 12/04/15 21:09:56 | A        | 192.168.0.1  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | IPLS   | 19989236 |
| BGP4MP     | 12/04/15 21:09:57 | A        | 192.168.0.7  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLA   |  6423369 |
| BGP4MP     | 12/04/15 21:09:57 | A        | 192.168.0.4  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | SNVA   | 16122697 |
| BGP4MP     | 12/04/15 21:09:57 | A        | 192.168.0.7  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | ATLAM5 |  2294664 |
| BGP4MP     | 12/04/15 21:09:57 | A        | 192.168.0.1  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | HSTN   | 12125087 |
| BGP4MP     | 12/04/15 21:09:57 | A        | 192.168.0.7  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | CHIN   |  8258672 |
| BGP4MP     | 12/04/15 21:09:57 | A        | 192.168.0.4  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | WASH   | 18023174 |
| BGP4MP     | 12/04/15 21:09:57 | A        | 192.168.0.4  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | DNVR   | 14287715 |
| BGP4MP     | 12/04/15 21:09:57 | A        | 192.168.0.4  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | LOSA   |     1101 |
| BGP4MP     | 12/04/15 21:09:58 | A        | 192.168.0.4  |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | KSCY   |  4195363 |
| BGP4MP     | 12/04/15 21:09:58 | A        | 192.168.0.10 |   65000 | 77.243.240.0/20 | 65010 8607 6939 12389 42599 | IGP    | NYCM   | 10748792 |
+------------+------------------+----------+--------------+---------+----------------+-------------------------------+--------+--------+----------+
```

Figura 4.17: Consulta para el cálculo del tiempo de propagación - Algoritmo BGPPSep

```
mysql> SELECT count(*) / count(DISTINCT router) FROM rib_in_RR_Sep WHERE prefix="77.243.240.0/20" AND aspath="65010 8607 6939 12389 42599";
+-----------------------------------+
| count(*) / count(DISTINCT router) |
+-----------------------------------+
|                            3.7500 |
+-----------------------------------+
```

Figura 4.18: Promedio de anuncios recibidos con el prefijo P por router - Algoritmo BGPPSep

sus clientes. En promedio, los routers recibieron 3.75 (imagen 4.18) veces el *update* con el prefijo "77.243.240.0/20". Más adelante entraremos en detalle de este tema Si observamos el router "LOSA' podemos ver que recibe cuatro veces el mismo *update* (imagen 4.16) debido a que es cliente de los RRs antes mencionados. Esta situación, en topologías más grandes, puede tener consecuencias negativas ya que son más mensajes los que debe procesar cada router, lo que podría llegar a generar que el tiempo de propagación aumente.
En este caso, los últimos en recibir el *update* con el prefijo "77.243.240.0/20" son "NYCM" y "KSCY", que lo hacen 4 segundos después de ser recibido por el router de borde ("STTL"). Esta información la obtenemos haciendo la consulta que se muestra en la imagen 4.17. Para este caso particular, la diferencia con la topología con configuración *full-mesh* es despreciable ya que estamos hablando de una red con tan solo doce nodos.

Este mismo estudio se realizó con prefijos IPv4 e IPv6 para los distintos algoritmos de localización obteniendo los resultados que se muestran en las gráficas 4.19 y 4.20. En las tablas 4.11 y 4.12 se pueden ver los resultados cuantitativos obtenidos para cada topología y algoritmo utilizado.

Utilizando los mismos prefijos, estudiamos también el promedio de anuncios BGP repetidos que se recibían. Claramente el modelo *full-mesh* asegura que no se reciban anuncios repetidos, pero cuando hablamos de topologías con RRs el

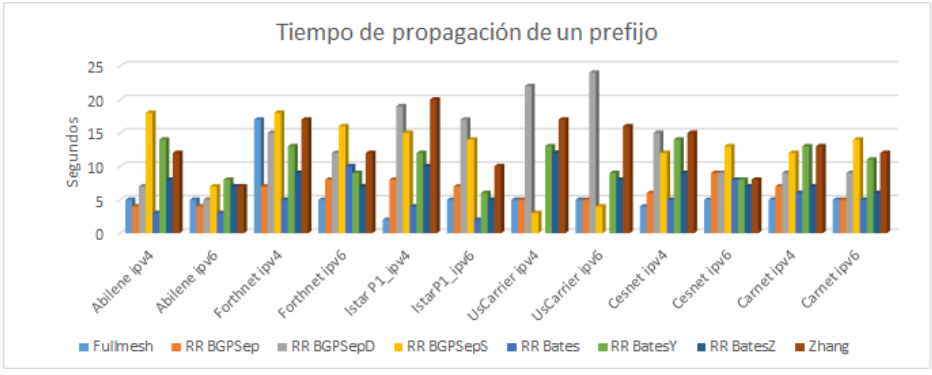


Figura 4.19: Tiempo de propagación de un prefijo p (1)

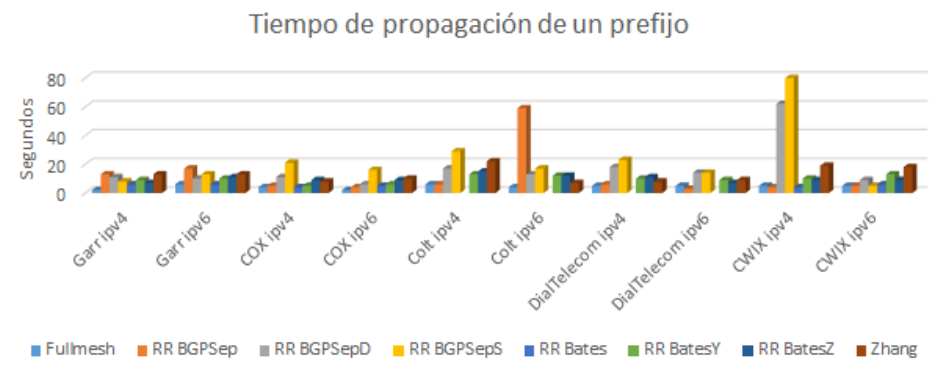


Figura 4.20: Tiempo de propagación de un prefijo p (2)

Topología	*full-mesh*		BGP Sep		BGP SepD		BGP SepS	
	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6
Abilene	5	5	4	4	7	5	18	7
Forthnet	17	5	7	8	15	12	18	16
Istar	2	5	8	7	19	17	15	14
UsCarrier	5	5	5	5	22	24	3	4
Cesnet	4	5	6	9	15	9	12	13
CWIX	5	5	4	5	62	9	80	5
Garr	2	6	13	17	11	10	8	13
COX	4	2	5	4	11	6	21	16
Colt	6	4	6	59	17	13	29	17
DialTelecom	5	5	6	3	18	14	23	14
Carnet	5	5	7	5	9	9	12	14

Tabla 4.11: Tiempo de propagación (en segundos) para un prefijo - Parte 1

Topología	BGP Bates		BGP BatesY		BGP BatesZ		BGP Zhang	
	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6
Abilene	3	3	14	8	8	7	12	7
Forthnet	5	10	13	9	9	7	17	12
Istar	4	2	12	6	10	5	20	10
UsCarrier	-	-	13	9	12	8	17	16
Cesnet	5	8	14	8	9	7	15	8
CWIX	4	6	10	13	9	9	19	18
Garr	6	6	9	10	7	11	13	13
COX	4	5	5	6	9	9	8	10
Colt	-	-	13	12	15	12	22	7
DialTelecom	-	-	10	9	11	7	8	9
Carnet	6	5	13	11	7	6	13	12

Tabla 4.12: Tiempo de propagación (en segundos) para un prefijo - Parte 2

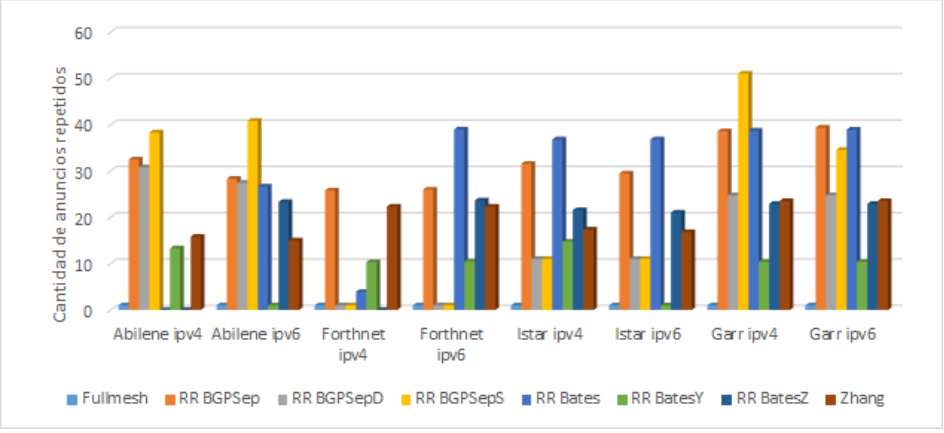


Figura 4.21: Cantidad de anuncios repetidos que recibe un router en promedio (1)

promedio varía según el algoritmo de localización de reflectores que se haya implementado. Las gráficas 4.21, 4.22 y 4.23 muestran la cantidad promedio de anuncios repetidos que le llegan a cada router. Las tablas 4.13 y 4.14 muestra los valores obtenidos de la base de datos y utilizados para realizar las gráficas antes mencionadas.
Cabe destacar que para tomar estos valores elegimos prefijos IPv4 e IPv6 que fueran propagados por todos los routers del AS. Como ya sabemos, en las topologías con RRs existen anuncios que son descartados por los reflectores de rutas.
Estas gráficas, como era de esperar, se comportan de manera muy similar a las gráficas que muestran la cantidad de mensajes trasmitidos dentro del AS. Es decir, nos muestra que la cantidad de anuncios descartados no son suficientes como para disminuir la cantidad de mensajes con respecto a una topología *full-mesh*.

4.6. Tiempos de Convergencia

Se define estado de convergencia del protocolo BGP cuando el conjunto de routers que participa de dicho protocolo tiene la misma información topológica de la red. El tiempo de convergencia se denomina al tiempo que demoran el conjunto de routers en alcanzar dicho estado.

Cuando hablamos de BGP y vemos la cantidad de anuncios que se reciben en los ASes del *core* de Internet nos parece difícil ver que exista un punto de

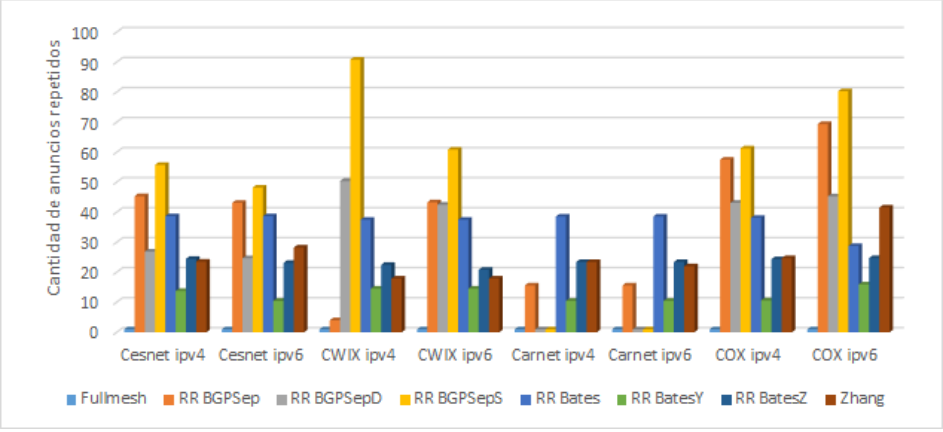


Figura 4.22: Cantidad de anuncios repetidos que recibe un router en promedio (2)

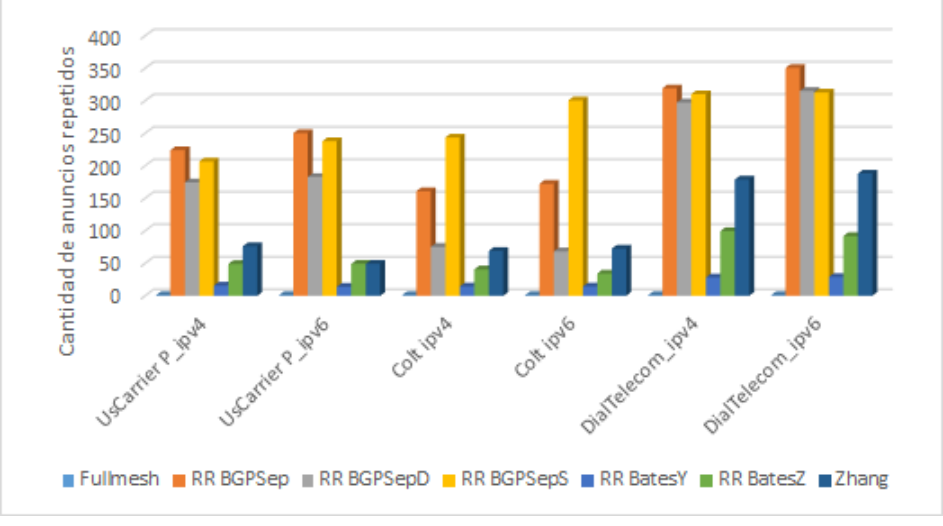


Figura 4.23: Cantidad de anuncios repetidos que recibe un router en promedio (3)

Topología	*full-mesh*		BGP Sep		BGP SepD		BGP SepS	
	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6
Abilene	1,00	1,00	3,25	2,83	3,08	2,75	3,83	4,08
Forthnet	1,00	1,00	2,58	2,60	1,00	1,00	1,00	1,00
Istar	1,00	1,00	3,15	2,95	1,10	1,10	1,10	1,10
UsCarrier	1,00	1,00	22,41	25,03	17,44	18,29	20,67	23,77
Cesnet	1,00	1,00	4,53	4,31	2,68	2,47	5,57	4,82
CWIX	1,00	1,00	4,00	4,33	5,04	4,25	9,08	6,08
Garr	1,00	1,00	3,86	3,93	2,47	2,47	5,10	3,45
COX	1,00	1,00	5,75	6,93	4,31	4,53	6,12	8,03
Colt	1,00	1,00	16,09	17,24	7,54	6,84	24,35	30,05
DialTelecom	1,00	1,00	31,92	35,06	29,74	31,52	31,01	31,28
Carnet	1,00	1,00	1,56	1,56	1,00	1,00	1,00	1,00

Tabla 4.13: Promedio de updates por nodo para un prefijo - Parte 1

Topología	BGP Bates		BGP BatesY		BGP BatesZ		BGP Zhang	
	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6
Abilene	3,50	2,66	1,33	1,00	2,25	2,33	1,58	1,50
Forthnet	3,90	3,90	1,03	1,05	2,33	2,36	2,23	2,23
Istar	3,68	3,68	1,47	1,00	2,15	2,10	1,73	1,68
UsCarrier	-	-	1,56	1,38	4,88	4,92	7,63	4,91
Cesnet	3,86	3,86	1,37	1,04	2,44	2,31	2,35	2,82
CWIX	3,75	3,75	1,45	1,45	2,25	2,08	1,79	1,79
Garr	3,87	3,89	1,04	1,04	2,29	2,29	2,35	2,35
COX	3,81	2,87	1,06	1,59	2,43	2,46	2,46	4,15
Colt	-	-	1,44	1,44	4,06	3,42	6,89	7,26
DialTelecom	-	-	2,83	2,90	9,95	9,18	17,88	18,81
Carnet	3,85	3,85	1,04	1,04	2,34	2,34	2,34	2,20

Tabla 4.14: Promedio de updates por nodo para un prefijo - Parte 2

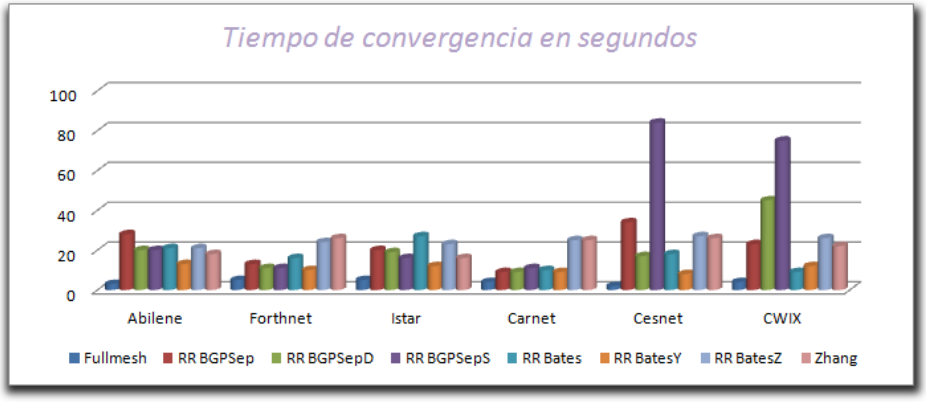


Figura 4.24: Tiempo de convergencia en segundos (1)

convergencia, incluso, según Randy Bush "*global bgp never converges*"[47]. Pero dado que en nuestras pruebas la cantidad de anuncios están acotados, podemos estar seguros de que se va alcanzar un estado de convergencia.

Las pruebas consistieron en una hora de inyección de trazas BGP, por lo que la información topológica de la red que mantiene cada router fue cambiado constantemente durante esa hora. Luego de que se envía el ultimo anuncio BGP, los routers empiezan a converger. En este estudio interesa ver cómo es la variación de este tiempo según los distintos algoritmos de localización de reflectores de rutas que se hayan utilizado, tomando como base los valores obtenidos con el modelo *full-mesh*. El tiempo que vamos a considerar es el tiempo desde que se inyectó el último anuncio BGP hasta que se alcanzó el estado de convergencia. Nuevamente recurrimos a la base de datos para obtener estos valores. Los resultados obtenidos se pueden apreciar en la gráficas 4.24 , 4.25 y 4.26, y en las tablas 4.15 y 4.16.
Lo interesante de estos valores es que muestran cómo afecta la cantidad de mensajes transmitidos a la *preformance* de la red. Es más notorio aún cuando los recursos no son suficientes para procesar todos los mensajes a tiempo. En estos casos, el tiempo de convergencia es mucho más alto.

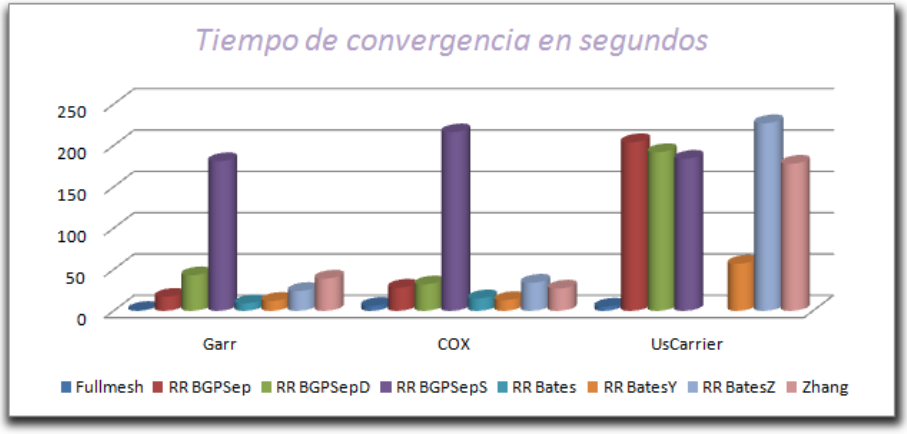


Figura 4.25: Tiempo de convergencia en segundos (2)

	full-mesh	BGP Sep	BGP SepD	BGP SepS
Abilene	3	28	20	20
Forthnet	5	13	11	11
Istar	5	20	19	16
UsCarrier	5	204	192	184
Cesnet	2	34	17	84
CWIX	4	23	45	75
Garr	1	17	43	181
COX	6	28	32	216
Colt	8	371	133	6.889
DialTelecom	5	1.740	391	7351
Carnet	4	9	9	11

Tabla 4.15: Tiempos de convergencia (en segundos) - Parte 1

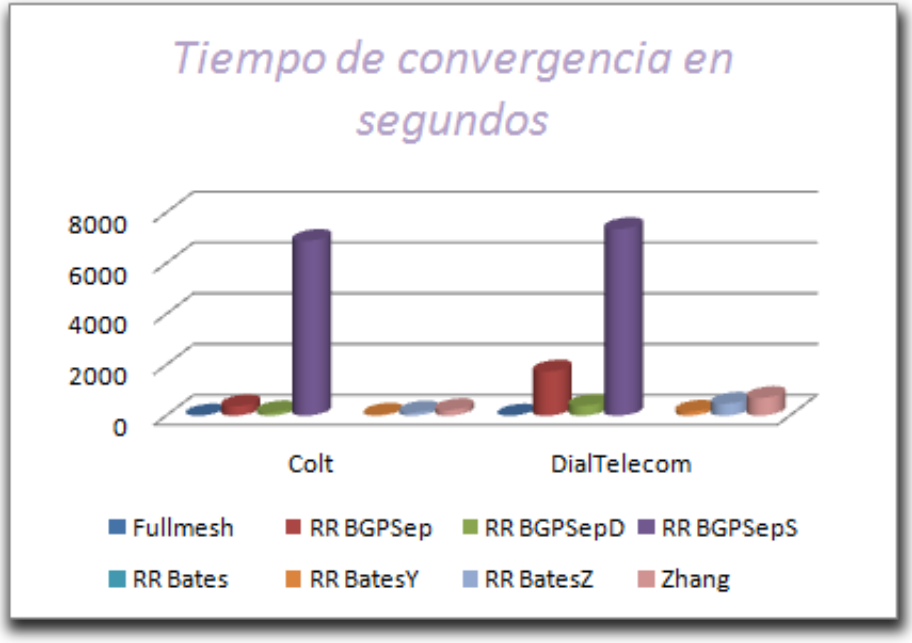


Figura 4.26: Tiempo de convergencia en segundos (3)

	BGP Bates	BGP BatesY	BGP BatesZ	BGP Zhang
Abilene	21	13	21	18
Forthnet	16	10	24	26
Istar	27	12	23	16
UsCarrier	-	57	227	178
Cesnet	18	8	27	26
CWIX	9	12	26	22
Garr	9	12	24	39
COX	15	13	34	27
Colt	-	41	129	220
DialTelecom	-	159	471	697
Carnet	10	9	25	25

Tabla 4.16: Tiempos de convergencia (en segundos) - Parte 2

4.7. Emulación en servidor de la facultad

Las primeras topologías utilizadas para realizar este estudio eran de pequeño tamaño y con pocas sesiones eBGP desde donde se les inyectaban las trazas BGP. El motivo de esto era principalmente para estudiar la correctitud de las herramientas y asegurarnos que podíamos obtener la información necesaria para nuestro estudio. A medida que las topologías fueron creciendo y también la cantidad trazas que se le inyectaba, fue necesario utilizar el servidor de la Facultad de Ingeniería con mayores prestaciones (64 GB de RAM, 100 GB de disco, 8 CPUs) para poder realizar las emulaciones ya que con los 16 GB de memoria RAM que poseía la computadora donde realizábamos las emulaciones no era suficiente.
El motivo por el cual el uso de memoria es significativo es debido a que durante la emulación las tablas de ruteo tanto de IPv4 como de IPv6 se almacenan en la memoria. Cuando hablamos que hay casos donde sólo la tabla de ruteo de IPv4 posee más 180.000 entradas y esto lo tenemos que multiplicar por la cantidad de routers de la topología, este uso de memoria empieza a tener sentido. Además, hay que considerar los procesos OSPF, BGP y zebra que se están corriendo en paralelo, más la cantidad de información extra que se guarda en cada router. Claro está, que el procesador también juega un papel importante en este tipo de emulación, pero a diferencia de la memoria, no es un factor del todo crítico ya que de no poseer un procesador cuyas prestaciones sean suficientes para realizar la emulación, se puede realizar igual la emulación pero la ejecución demorará un tiempo mayor al estimado. En cambio, el uso de memoria debe mantenerse en valores no muy cercanos al máximo disponible porque de no ser así, no se podrá terminar la emulación debido a que no hay memoria suficiente para guardar más rutas en los routers.

Capítulo 5

Ingeniería de Software

En este capitulo hablaremos de la planificación que se hizo al inicio del proyecto y cómo ésta fue variando durante el transcurso del proyecto. Hablaremos de cuáles fueron los puntos que nos llevaron más tiempo del estimado, qué tareas planificamos hacer y finalmente no se realizaron, y explicaremos los motivos. Además, hablaremos de cuáles fueron los problemas encontrados que llevaron a que éstos tiempos no se cumplieran.

5.1. Planificación

La planificación de un proyecto cuyo tema no es el *expertise* de ninguno de los integrantes del grupo hace que no sea una tarea sencilla. El margen de error es

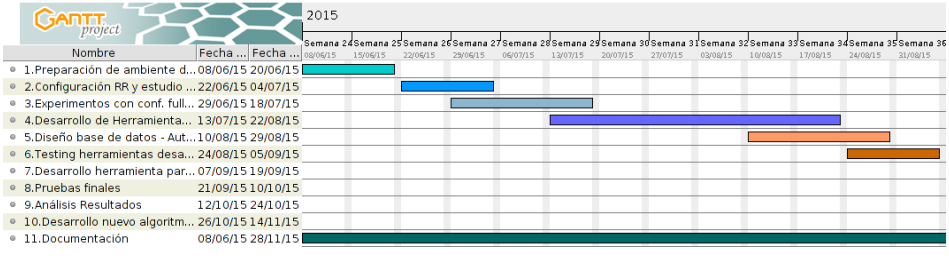


Figura 5.1: Planificiación inicial del proyecto (1)

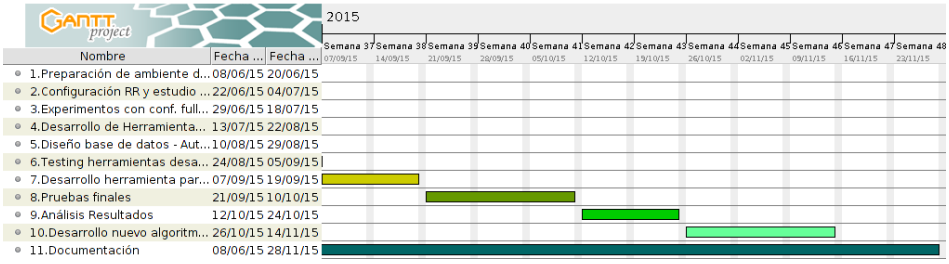


Figura 5.2: Planificiación inicial del proyecto (2)

mucho mayor a la hora de estimar los tiempos en las distintas tareas a realizar. Por eso, antes de realizar cualquier planificación se dedicaron los primeros meses al estudio de los temas involucrados y a definir el alcance. Una vez adquiridos los conocimientos básicos y con los objetivos claros se realizó la planificación inicial. La misma se muestra en las figuras 5.1 y 5.2. A continuación detallaremos cada una de las tareas planificadas:

1. Preparación de ambiente de trabajo:
   - Estudio de las herramientas disponibles para la emulación de Sistemas Autónomos y elección de las que cumplen con los requisitos para nuestro estudio.
   - Definir parámetros de configuración para un desempeño óptimo de las herramientas.
   - Experimentar con redes de gran tamaño (decenas o cientos de routers) y ver uso de recursos.
2. Configuración de los Reflectores de Rutas y estudio de la correctitud de las herramientas elegidas:
   - Estudiar y entender la configuración de RR en Sistemas Autónomos en general.
   - Configurar los RRs en una topología de prueba utilizando el emulador seleccionado.
   - Configurar los clientes de los RRs en la topología de prueba.
   - Realizar pruebas que muestren la correctitud de la configuración con RRs.

- Diseñar *templates* de configuración para los routers.

3. Experimentos con configuraciones *full-mesh* y con RRs. Inyección de trazas:
   - Estudiar el formato de las salidas y los distintos parámetros de la herramienta RRLoc.
   - Analizar los datos y decidir qué información es relevante y cuál se puede descartar.
   - Diseñar un procedimiento para obtener los datos que nos interesan.
   - Estudiar alternativas para analizar grandes volúmenes de datos.
4. Desarrollo de herramienta que permite convertir la salida de la herramienta RRLoc en la entrada del emulador:
   - Estudiar la salida de xTotem.
   - Estudiar cómo debe ser la entrada para la herramienta de emulación tanto para una configuración *full-mesh* como para una con RR.
   - Desarrollo de la herramienta.
5. Pruebas sobre la herramienta desarrollada:
   - Generar distintas topologías para pruebas.
   - Obtener distintas configuraciones iBGP con la ayuda de la herramienta RRLoc sobre las topologías.
   - Probar herramienta con las salidas obtenidas y verificar correctitud.
   - Inyectar pequeñas trazas y obtener datos en texto plano.
6. Base de Datos:
   - Definir tablas.
   - Desarrollar herramienta que permita automatizar el proceso de cargar los datos a la base.
   - Estudiar que tipo de consultas realizar a la base para comparar resultados.
7. Obtener datos para las pruebas finales:
   - Obtener topologías reales de Sistemas Autónomos de Internet.

- Desarrollar herramienta que permita convertir las topologías al formato soportado por la herramienta RRLoc.
- Obtener trazas BGP reales de una hora de duración.

8. Pruebas finales:
   - Correr todos los algoritmos de localización de Reflectores de Rutas para las topologías obtenidas.
   - Emular las redes inyectando trazas durante una hora.
   - Comparar el uso de recursos entre las diferentes configuraciones BGP.
   - Almacenar los resultados obtenidos en la base de datos.
   - Cargar los datos a las bases.
9. Análisis de los resultados:
   - Realizar consultas a la base de datos.
   - Graficar resultados.
   - Comparar topologías con *full-mesh* de sesiones iBGP contra topologías con RR.
   - Escribir conclusiones.
10. Buscar y/o proponer otro algoritmo de localización de RR:
    - Buscar y estudiar algoritmos de localización de Reflectores de Rutas.
    - Diseñar algoritmo de localización.
    - Implementar algoritmo y realizar pruebas.
11. Documentación.

Como se dijo antes, por falta de experiencia algunas de las tareas requirieron más tiempo que otras. Además, se enfrentaron algunos problemas durante el desarrollo del proyecto que hicieron que las tareas se atrasaran más. De estos problemas hablaremos en la siguiente sección. Sin embargo, a pesar de los problemas enfrentados y los tiempos mal calculados se realizaron todas las tareas propuestas menos una: la búsqueda o proposición de otro algoritmo. Esta tarea se había propuesto, en el inicio del proyecto, para el caso que los tiempos planificados fueran los adecuados y se tuviera tiempo de realizar este estudio. Además, no está dentro de los objetivos del proyecto por lo que quedrá para un trabajo a futuro.

5.2. Problemas Encontrados

A lo largo de este proyecto nos hemos enfrentado a problemas de distinta índole. Algunos de los cuales nos han consumido gran parte del tiempo que le hemos dedicado al proyecto y otros que hemos sabido superarlos fácilmente. En esta sección hablaremos de todos ellos haciendo un especial énfasis en aquellos que no hemos podido encontrar solución y que quedarán para trabajos futuros.
Podemos clasificar los problemas en dos tipos: por un lado aquellos problemas que surgen del desarrollo de las aplicaciones y la integración de los componentes, y por otro lado aquellos problemas que hemos heredado.

5.2.1. Problemas enfrentados durante el desarrollo

Al inicio de nuestro estudio nos habíamos planteado cómo debíamos encarar un proyecto de estas características. Estaba la opción de utilizar herramientas ya conocidas o desarrollar nuestro propio software de emulación, con todo lo requerido para para realizar el proyecto. Pero, ¿para qué reinventar la rueda?. Es mucho más sencillo utilizar software de terceros que ya ha sido probado, que empezar desde cero el desarrollo de una aplicación de gran porte como lo es un emulador de red. Por otra parte, los problemas que surgen de herramientas desarrolladas por terceros son mucho más complejas y difíciles de solucionar, más aun si no existe ningún tipo de soporte por parte de los desarrolladores. Éste es el caso de Totem [30]. No queremos profundizar en los problemas que tuvimos con esta herramienta ya que mucho de éstos problemas también han sido experimentados y documentados por el grupo de estudiantes que desarrolló RRLoc [31]. Simplemente queremos acotar que se tuvieron varios problemas de compatibilidad con las distintas librerías que precisa la herramienta para poder funcionar. Más adelante, veremos un ejemplo cuando hablemos de los problemas relacionados con el algoritmo Optimal "Buob".

Otra de las herramientas que nos ha causado algunos inconvenientes fue *exabgp*. Como ya hemos mencionado *exabgp*, denominada *"The BGP swiss army knife of networking"* por su desarrollador Thomas Mangin, comenzó como un simple inyector de mensajes BGP hasta convertirse en mucho más que eso. A diferencia de Totem, esta herramienta sigue creciendo agregando nuevas funcionalidades y además cuenta con soporte (que a nuestro criterio es muy bueno). Los problemas que surgieron con esta herramienta fueron reportados como *bugs* y solucionados por la comunidad que brinda soporte rápidamente. Esto hizo que bastara con algunas actualizaciones de éste software para solucionar los problemas. No obstante, los errores detectados en la herramienta causaron que varios experimentos fallaran en mitad de la emulación o, en algunos casos, que no pasaran el proceso de verifi-

cación al final de la corrida. Esto generaba que los datos no fueran válidos y se perdieran varias horas de trabajo.

Al final del proyecto, en la etapa de análisis de los datos, nos llamó la atención que algunas pruebas registraban una cantidad excesiva de mensajes de tipo *withdraw* tiempo después de haber terminado la inyección de mensajes BGP. También vimos que el tiempo de convergencia en esos casos era mucho mayor. Analizado los *withdraws* notamos que se estaban descartando todos los prefijos aprendidos en algunos routers, lo que nos dio una pista de que era lo que estaba sucediendo. De la misma forma que cuando se inicia la emulación se deben "levantar" cada uno de los routers, cuando se finaliza se deben "bajar" los mismos. Esto implica, por ejemplo, que se corten las sesiones eBGP y por lo tanto, que se envíen mensajes de tipo *withdraw* con todos los prefijos aprendidos por esa sesión. Esto pasa cada vez que se termina la emulación, pero sólo queda registrado en aquellas topologías que demoran más de un minuto en "bajar" todos los routers. Esto sucede debido a que los mensajes recibidos por cada router, por defecto, son guardados cada un minuto.
Cabe aclarar, como se mencionó antes, que al final de cada experimento se realizaba un procedimiento de verificación, como el control de de las tablas "Loc-RIB" o el tamaño de las tablas "RIB-in" de los routers de borde, que tenía como objetivo obtener indicios de si los datos obtenidos eran correctos. Estos procedimientos no detectaban el comportamiento descrito anteriormente ya que es un comportamiento totalmente válido el cual no nos interesa registrar.
Este problema implicó, por un lado, modificar los archivos de configuración generados por la herramienta "TTools" de forma que los mensajes sean almacenados cada cinco minutos. Por otro lado, lo que más complicaciones nos generó, es la eliminación de estos mensajes en las bases de datos. La consulta para determinar cuáles eran estos mensajes y su posterior eliminación requería de varias horas en las topologías de gran tamaño.

5.2.2. Problemas heredados

Una de las etapas del proyecto consistió en la elección de herramientas. Para esto se hicieron varios experimentos de forma de determinar si las mismas funcionaban correctamente antes de empezar las pruebas finales. Varias herramientas que se descartaron por no cumplir con los requisitos mínimos. Los problemas relacionados con las implementaciones de los diferentes algoritmos de localización de RR fueron problemas inevitables y que no podíamos desechar de la misma forma que lo hicimos con las herramientas porque formaban parte de nuestro estudio. Son éstos problemas los que consideramos heredados y se comentan a continuación.

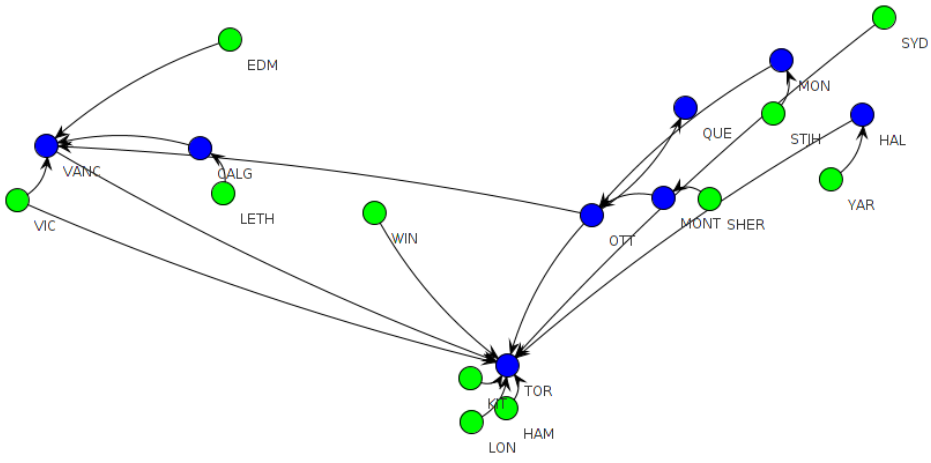


Figura 5.3: Topología ISTAR luego de la aplicación del algoritmo Optimal. Los RRs están representados en azul.

5.2.2.1. Problemas con el algoritmo Optimal

El algoritmo Optimal nos trajo muchos dolores de cabeza. Para comenzar, la implementación del algoritmo utiliza un paquete de software de optimización llamado *CPLEX*. Este paquete es de licencia propietaria de IBM. Existe una versión de prueba gratuita, pero sólo permite correr el algoritmo en topologías con 19 routers máximo. Investigando, se encontró que el grupo que implementó el algoritmo consiguió la licencia de *CPLEX* a través de la Facultad. Nos contactamos con Carlos Testuri quien nos facilitó el acceso al software. El problema surgió al probar la versión que se tenía en Facultad. La herramienta RRLoc es un software de 32 bits y la versión del paquete que se tiene en la actualidad en Facultad es de 64 bits (en su momento se tuvo la versión de 32 bits), por lo que no son compatibles. Se estuvieron varios meses tratando de que IBM nos brinde el software de manera gratuita sin éxito. Sólo a través de nuestro tutor (que amablemente llenó varios formularios y contestó muchas preguntas del personal de IBM) pudimos conseguir el software.

Luego de tener el software necesario para correr el algoritmo en RRLoc, se comenzaron las emulaciones con diferentes topologías después de haberles ejecutado el algoritmo Optimal. Se tenían muchos datos guardados en la base de datos, cuan-

do se hizo la corrida de una hora con la topología "ISTAR" (resultado de aplicarle el algoritmo Optimal, ver figura 5.3) y se vio que no todos los routers recibían todos los anuncios. En particular, el router "QUE" no recibe ningún anuncio. Las pruebas con la topología "ISTAR" consistieron en la inyección de trazas en los routers "VANC" y "TOR" (routers de borde). Como se puede ver en la figura 5.3 el router denominado "OTT" es cliente tanto de "VANC" como de "TOR" por lo tanto va a recibir todos los anuncios. La razón por la cual el router "QUE" no recibe ningún anunció se debe a que el router "OTT" recibe anuncios de routers que no son clientes de él. Como vimos en la sección 2.1.7.1, los anuncios provenientes de *"Non-Client peers"* únicamente se propagan a los *"Client peers"*, por lo tanto, el router "OTT" no propagará los anuncios al router "QUE".
Se vio, además, que en algunos casos los routers reciben sólo algunas rutas. Este es el caso de la topología "Forthnet" luego de la aplicación del algoritmo (ver figura 5.4). Los routers "CHAN" y "HERA" reciben sólo las rutas aprendidas por "ATH" a través de eBGP, y no las que aprende "THESS". Como se puede ver en la figura los router "CHAN" y "HERA" no son clientes de "ATH" pero dado que los anuncios que provienen de una sesión eBGP son propagados a los *"Clients and Non-Clients peers"* (ver sección 2.1.7.1), los anuncios también llegan a estos routers. No sucede los mismo con los anuncios que provienen de una sesión iBGP, en este caso de la sesión iBGP que mantiene el router "ATH" con "THESS". Por lo tanto, los routers "CHAN" y "HERA" no reciben los anuncios que envía el router "THESS".
Estos errores tuvieron como consecuencia la posterior eliminación de los resultados obtenidos con otras topologías utilizando el algoritmo Optimal para la localización de RRs.

Se realizaron más pruebas y se comprobó que el algoritmo estaba mal implementado, por lo que se inició la ardua tarea de buscar los errores y corregirlos. Como se vio en el capítulo de Estado del Arte, la descripción de este algoritmo es muy vaga. Es un problema de optimización, pero sólo se plantean algunas de las restricciones como funciones, el resto se describen aunque no en detalle. Por lo tanto, algunos puntos del algoritmo se dejan libres a lo que entienda la persona que lo está leyendo. Además, se tuvo la difícil tarea de entender un código que no es de nuestra autoría y que no tiene muchos comentarios, sumado al estudio de la integración del paquete *CPLEX* y sus funciones. Se encontró un error y se corrigió. Sin embargo, ese error no arreglaba el algoritmo. Durante un par de meses se fue estudiando el algoritmo y el código de implementación del mismo. Se buscó más información sobre el mismo sin éxito. Un sólo artículo existe sobre el mismo y era con el que trabajábamos [28]. Finalmente nos contactamos con el docente Claudio Risso (experto en la temática de Programación Entera) para ver

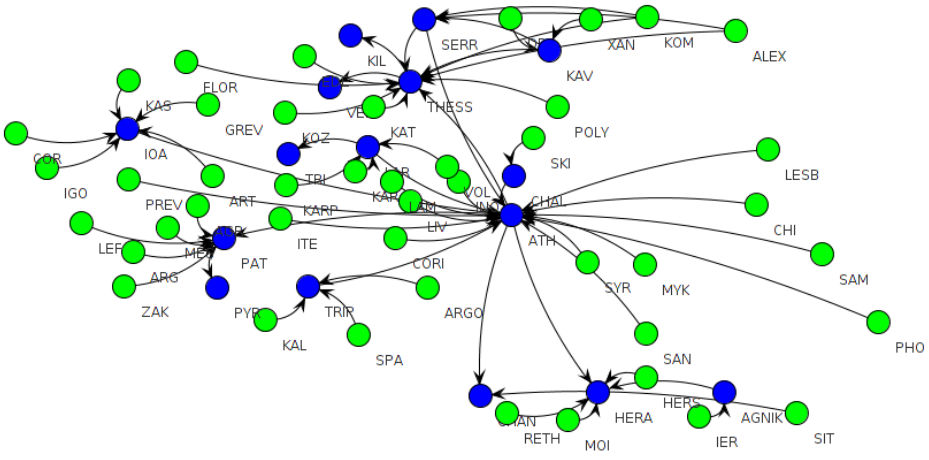


Figura 5.4: Topología Forthnet luego de la aplicación del algoritmo Optimal. Los RRs están representados en azul.

si nos podía ayudar a entender el artículo del algoritmo, y de esta forma ver dónde radicaba el/los error/es en la implementación. Luego de que dicho docente pudo estudiar el artículo que le mandamos nos reunimos para hablar sobre el tema. La conclusión que sacamos fue que no se podía realizar una implementación del algoritmo únicamente con lo que dice el artículo ya que existen restricciones de las cuales no se puede sacar fácilmente una función. Por lo tanto, luego de más de 3 meses de lucha con el algoritmo se descartó su uso.
Cabe aclarar que se le dedicó mucho tiempo ya que, según el autor del mismo, es el algoritmo que da mejores resultados para las implementaciones de topologías con reflectores de rutas.

5.2.2.2. Problema con la heurística BatesY

Como ya se explicó en el capítulo de Estado del Arte, la heurística BatesY toma el router más conectado de cada PoP y lo selecciona como Reflector de Rutas de ese PoP. El resto de los routers de ese PoP serán clientes del RR, establecerán un *full-mesh* de sesiones entre los routers clientes del PoP, y todos los RRs de los diferentes PoPs establecerán un *full-mesh* de sesiones entre ellos. Sin embargo, en la implementación de este algoritmo se realiza una selección de los routers más

conectados de cada PoP (según un cierto criterio), y si se tienen varios routers como los más conectados (a pesar de tener diferente cantidad de enlaces con el resto de los routers del PoP) se toman todos esos routers como Reflectores de Rutas, cuando la heurística debería escoger uno.
Afortunadamente, estudiando el código de implementación de la heurística fácilmente se pudo arreglar este problema. Sin embargo, el estudio de la implementación, arreglo y posteriores pruebas de correctitud consumieron tiempo planificado para otras tareas (aproximadamente una semana).

5.2.2.3. Problema con la heurística BatesZ

Como también se vio antes, la heurística BatesZ es similar a BatesY, con la diferencia que por cada PoP se tienen dos Reflectores de Rutas en lugar de uno. Estos RRs son los dos routers más conectados del PoP. Luego, cada router del PoP es cliente de ambos RRs, y los RRs de todos los PoPs establecen un *full-mesh* de sesiones entre ellos. En este caso, también encontramos un error en la implementación similar al anterior. En vez de elegir los dos routers más conectados del PoP, si existían más de dos, se elegían todos como Reflectores de Rutas del PoP, no cumpliendo con lo que dice ésta heurística.
Estudiando el código se encontró el error y se pudo corregir. Pero al igual que en caso del BatesY, éstas tareas insumieron un tiempo no planificado. Cabe aclarar que el tiempo insumido en el arreglo de este problema fue menor que en caso anterior ya que, al ser un problema similar y el algoritmo muy parecido, se pudo entender más rápidamente el código.

Capítulo 6

Conclusiones y Trabajo a Futuro

En este capitulo trataremos de responder las interrogantes que nos hemos planteado al comienzo. Hablaremos de las enseñanzas y de las conclusiones que arrojó este Proyecto de Grado. Finalizaremos el capítulo proponiendo trabajos futuros en la misma temática del proyecto impulsados por nuevas interrogantes que surgen de las conclusiones y de aquellos objetivos que no hemos podido alcanzar por diversos motivos.

6.1. Conclusiones

Uno de los objetivos más importantes que nos hemos planteado es estudiar cómo afecta a la *performance* de la red las distintas configuraciones iBGP. En particular, uno de los factores que más nos interesa es la cantidad de mensajes que se envían en las sesiones. Como hemos mencionado en el "Estado del Arte", existen algunos autores que afirman que las configuraciones iBGP con RRs disminuyen el tamaño de la rib_in total de los routers, algo que hemos puesto en duda. Los resultados obtenidos en los experimentos nos indican que esto no es así. En los únicos casos que hemos tenido una reducción en la cantidad de mensajes totales enviados fueron en topologías ideales para utilizar RRs, como lo son "Istar" y "Forthnet" (imagen 6.1). Los grandes AS de internet no tienen este tipo de topologías ya que tienen enlaces redundantes.
De todas formas, no podemos afirmar que no existe algún algoritmo que permita

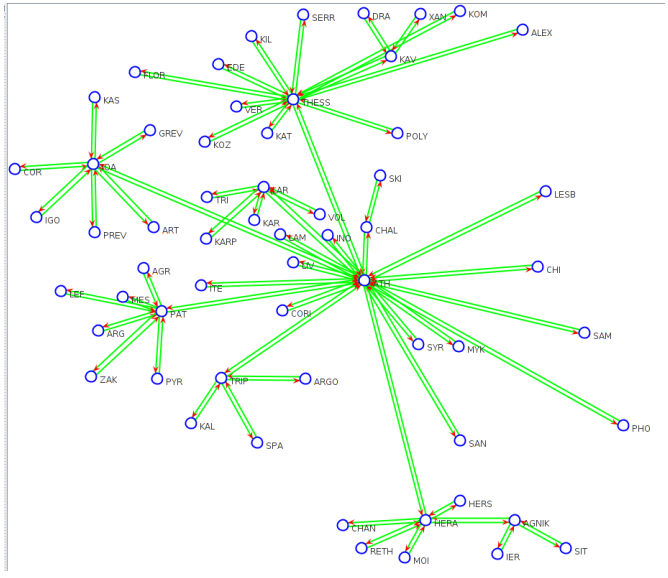


Figura 6.1: Topología Forthnet

disminuir la cantidad de mensajes sea cual sea la topología de la red. Simplemente podemos afirmar que la implementación de RRs no implica una disminución de los mensajes transmitidos por el protocolo BGP.
Hemos comprobado que los RRs muchas veces evitan que anuncios sean propagados por todo el AS, pero no la cantidad suficiente como para reducir la cantidad de mensajes totales trasmitidos. Analizando el tiempo de propagación y la cantidad de anuncios repetidos pudimos comprender el porqué de semejante aumento. Los reflectores de rutas provocaban que cada router reciba reiteradas veces el mismo anuncio, mientras que en un configuración *full-mesh* esto no pasa.

De los distintos algoritmos estudiados la heurística "BatesY" ha sido la que obtuvo mejores resultados en todo los aspectos analizados. Si bien logró, en algunas topologías, reducir el total de mensajes, dicha disminución no fue significativa. Nos parece importante destacar que este algoritmo fue el único que logró disminuir en mayor medida la cantidad de mensajes recibidos por los clientes, algo que creíamos que lo lograba cualquier configuración con RRs.

Como hemos visto al comienzo, el hecho de que se envíen más mensajes BGP para transmitir la misma cantidad de información implica que cada router debe

procesar más mensajes y, por lo tanto, que utilice mayores recursos. Esto claramente puede tener un impacto negativo en la *performance* de BGP, pero ¿qué tan negativo puede ser procesar más mensajes?. Uno de los estudios se basó en comparar los tiempos de convergencia para las distintas configuraciones. Este estudio muestra el impacto que tiene procesar más mensajes. Teniendo en cuenta que los recursos disponibles en cada prueba fueron los mismos, podemos afirmar que el el aumento en la cantidad de mensajes BGP procesados tiene un impacto notorio en la *performance* de BGP.

Algunas de las preguntas que nos hacemos luego de ver los resultados son: ¿Se pueden mejorar éstos algoritmos?, ¿Qué deben tener en cuenta algoritmos futuros de localización de Reflectores de Rutas para tener un desempeño óptimo?. Claramente existen factores que no son tenidos en cuenta por los algoritmos estudiados a la hora de definir la configuración iBGP de la red, y que sin duda alguna ayudarían a mejorar los mismos. La implementación de *Clusters* para la redundancia de RRs es una buena practica para tratar de disminuir la cantidad de mensajes transmitidos, como hemos visto en la sección 2.1.7.3 de este documento. Lamentablemente, ninguno de los algoritmos estudiados definen *Clusters* con RRs redundantes.
Otro aspecto importante a tener en cuenta son los routers de borde del AS. Definir un router de borde como RR no es muy útil ya que todo trafico recibido por una sesión eBGP va a ser transmitido tanto a clientes como a no clientes. El único algoritmo que utiliza estos datos para obtener una buena localización de RR es el algoritmo óptimo, el mismo que no pudimos corregir por falta de información.

En el pasado la cantidad de sesiones BGP que podía soportar un router era relativamente pequeña. Gracias a los avances tecnológicos hoy en día los routers soportan miles de sesiones BGP eliminando la razón primordial para la implementación de Reflectores de Rutas. Sin embargo, el costo operacional de mantener un *full-mesh* de sesiones iBGP mantiene una fuerte motivación para implementar configuraciones iBGP con Reflectores de Rutas en redes de gran tamaño.

Cuando hablamos de costo operacional también debemos tener en cuenta que tan dinámica es la infraestructura de red del AS a la hora de implementar RRs. Los ASes de Internet están en constante crecimiento y, como hemos visto, agregar un nuevo nodo en una red con RRs es más sencillo. El problema surge cuando la localización de los Reflectores están dados por algún algoritmo como los que hemos estudiado. Estos algoritmos pueden variar de manera significativa cuando se agrega un nuevo nodo a la red. Sin embargo, volver a correr el algoritmo de localización de RR cada vez que se extiende la red es poco aconsejable ya que

puede llevar a reconfigurar toda la red. Esto arroja, a nuestro entender, una de las conclusiones más importantes en lo que respecta a los algoritmos estudiados. Ante pequeños cambios en la red los algoritmos pueden llegar a generar una configuración BGP con grandes cambios, lo que implica un costo operacional muy alto.

La planificación inicial, explicada en el capítulo anterior, fue bastante acertada en cuanto a las tareas que debíamos realizar a los largo del proyecto, pero no así en cuanto a los plazos definidos para las mismas. Si pudríamos volver atrás en el tiempo hubiéramos agrado la tarea "Estudiar correctitud de las implementaciones de los algoritmos de Reflectores de Rutas". Esta tarea nos hubiera ahorrado de mucho trabajo y tiempo que al fin de cuentas solo sirvió para poder afirmar que algunos algoritmos no funcionaban correctamente.
Algo que aprendimos es que es muy difícil afirmar que una herramienta funciona correctamente en cualquier caso, lo fácil es probar que falla. Pero lo que si se puede hacer es realizar pruebas para tener mayor certeza de que funciona bien. Esto sin duda fue algo que no consideramos.

Finalmente queremos concluir esta sección haciendo un pequeño resumen con alguno de los puntos más interesantes sobre los RRs y distintos algoritmos estudiados.

- El costo operacional de mantener un *full-mesh* de sesiones iBGP es una de las grandes razones para implementar configuraciones iBGP con Reflectores de Rutas en AS de Internet.
- Implementar una configuración de sesiones iBGP con RRs sin un algoritmo de localización en redes de gran tamaño es una tarea muy difícil. Una mala elección de un Reflector de Rutas puede introducir una falla que deje algún nodo sin recibir todos los anuncios BGP o provoque algunos de los problemas que acarrean los RRs (visto en Estado del Arte).
- Por otra parte los algoritmos de localización estudiados tienen varios puntos débiles. Entre ellos:
  - Generan una cantidad de mensajes significativamente mayor que la configuración *full-mesh*.
  - Ante pequeños cambios en la red, el resultado de los algoritmos genera varios cambios en la configuración.
  - No implementan *Clusters* para la redundancia de RRs, lo que hace que se envíen más mensajes.

- No tienen en cuenta cuáles son los routers de borde del AS. Excepto el algoritmo Optimal, pero se vió que está mal implementado.

6.2. Trabajo Futuro

Cuando uno pasa tanto tiempo estudiando un mismo tema se le ocurren un sin fin de trabajos para realizar en la misma temática. En esta sección mencionaremos aquellos que nos hubiera gustado trabajar o aquellos que surgen de los nuevos datos obtenidos y de las conclusiones.

Una de las tareas propuestas en la planificación inicial era la de crear un algoritmo de localización de Reflectores de Rutas a partir de los datos obtenidos en las pruebas o implementar alguno diferente a los que ya se encontraban en la herramienta RRLoc. Por los problemas que ya hemos descrito, nos ha quedado esta tarea como pendiente. Creemos que nuestro estudio ha arrojado varios datos interesantes que pueden ser tenidos en cuenta a la hora de desarrollar un algoritmo eficiente de localización de Reflectores de Rutas.

Otra tarea pendiente, pero que no estaba dentro de la planificación es la corregir la implementación del algoritmo óptimo "Boub" (Optimal). Sin duda alguna, esta no es una tarea para nada sencilla y requiere de mucho tiempo y dedicación. No queremos volver a mencionar los problemas que nos causó este algoritmo y el esfuerzo que hicimos para tratar de solucionarlo, pero es importante tener esto en cuenta.
Cabe aclarar que si bien se han encontrado errores en la implementación del algoritmo no descartamos que el problema sea el algoritmo en sí.

En la herramienta RRLoc no se pueden correr los algoritmos por línea de comandos. Los autores de dicho software aseguran que sí se puede, pero hemos probado y el único que se puede correr es el Optimal. El resto de los algoritmos requieren algún dato adicional. Sin embargo, al ejecutarlo por línea de comandos no se puede modificar estos parámetros adicionales, por lo que siempre se ejecutan con los valores por defecto. Por lo tanto, un posible trabajo a futuro es poder ejecutar los algoritmos por línea de comandos y no teniendo que entrar a la interfaz gráfica. Esto se pensó para poder tener un *framework* que tome un archivo xml o gml con la topología, el algoritmo a correr y sus parámetros, y genere directamente el ambiente para emular, facilitando así la tarea de crear el ambiente de emulación.

De forma de mejorar el *framework* desarrollado se podría configurar NetFlow en los routers del AS y un colector donde se pueda almacenar toda la información que exporten los routers.
La ventajas de configurar Netflow es que a medida que se realiza la emulación

todos los datos son almacenados en el colector y no es necesario ni traducir los archivos binarios en formato MRT, ni luego cargarlos a una base ya que el colector recibe la información en texto plano y cuenta con su propia base. Además, existen distintos programas *open source* que permiten analizar este trafico y que brindan distintas herramientas de analítica.

Cuando empezamos investigar sobre los Reflectores de Rutas nos encontramos con varios documentos que hablaban de configuración de VPNs de Capa 3 MPLS con Reflectores de Rutas. Si bien el tema escapaba de nuestro conocimientos, nos llamó la atención que hubiera tanta información al respecto. Esto, para nosotros, fue un indicio de que habían otros escenarios interesantes para estudiar el comportamiento de los RRs.
MPLS (*MultiProtocol Label Switching*) es un protocolo de conmutación por etiquetas definido por la IETF y definido en el RFC 3031 [48], que trata de proporcionar algunas de las características de las redes orientadas a conexión a las redes no orientadas a conexión. En el encaminamiento IP sin conexión tradicional, la dirección de destino junto a otros parámetros de la cabecera es examinada cada vez que el paquete atraviesa un router. La ruta del paquete se adapta en función del estado de las tablas de encaminamiento de cada nodo, pero como la ruta no puede predecirse, es difícil reservar recursos que garanticen la QoS (*Quality of Service* - Calidad de Servicio); además, las búsquedas en tablas de encaminamiento hacen que cada nodo pierda cierto tiempo, que se incrementa en función de la longitud de la tabla.
MPLS permite a cada nodo asignar una etiqueta a cada uno de los elementos de la tabla y comunicarla a sus nodos vecinos. Esta etiqueta es un valor corto y de tamaño fijo transportado en la cabecera del paquete para identificar un FEC (*Forward Equivalence Class*), que es un conjunto de paquetes que son reenviados sobre el mismo camino a través de la red, incluso si sus destinos finales son diferentes. La etiqueta es un identificador de conexión que sólo tiene significado local y que establece una correspondencia entre el tráfico y un FEC específico. Esto permite reservar recursos para garantizar calidad de servicio lo que hace que sea una muy buena solución para transportar voz sobre IP (voIP).
Para entender un poco mejor la motivación veamos la siguiente realidad. La Intendencia de Montevideo (IM) cuenta con cientos de sitios externos como Policlínicas, Centro Cívicos Centros Comunales, Municipios, Usinas, Museos, Laboratorios, entre otros, dispersos por todo Montevideo. Para conectar todos ellos, la IM contrata un servicio de VPN capa 3 sobre la red MPLS a su proveedor de servicios. Para poder anunciar las redes de cada sitio los routers denominados CE (*Customer Edge*) utilizan el protocolo BGP. Si bien del lado del cliente solo significa agregar

una sesión eBGP, del lado del ISP se debe configurar un *full-mesh* de sesiones con el nuevo PE (*Provider Edge*) que conecta el nuevo sitio de la IM. Utilizado Route Reflector esto no es necesario, en principio alcanzaría con agregar una simple sesión iBGP con un RR. Esto hace atractivo el estudio de algoritmos de localización de reflectores de rutas sobre redes MPLS.

Apéndice A

Instalación del Ambiente de Trabajo

En este anexo se describen las instalaciones necesarias para la creación del ambiente de trabajo utilizado en la pruebas del proyecto.

Estas instalaciones se realizaron tanto en computadoras personales como en el servidor de Facultad. No requiere grandes cantidades de almacenamiento, pero dependiendo de la cantidad de routers de la topología es la cantidad de memoria RAM que se necesita. Como ejemplo, un PC con 16 GB de RAM llegó a soportar la emulación de un Sistema Autónomo de 60 routers.
Todas las instalaciones se realizaron en un sistema operativo Linux, en particular, en Ubuntu 14.04 de 64 bits.

A.1. Instalación RRLoc

El *framework* RRLoc es un software de 32 bits, probado en Linux y requiere Java JDK en su versión de 1.6 o 1.7. En nuestro caso, al tener PCs de 64 bits realizamos la instalación del Java JDK 1.7 de 32 bits y antes de utilizar el *framework* exportamos el Java 32 bits de la siguiente manera:

```
$ export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-i386
$ export PATH=/usr/lib/jvm/java-1.7.0-openjdk-i386/bin:$PATH
```

Luego de tener instalado Java, se compila el código fuente del RRLoc mediante el comando:

```
$ ant
```

Y se ejecuta el software de la siguiente manera:

```
$ sh totem.sh
```

Por más información ver el archivo Readme.pdf que se encuentra en la carpeta del *framework*.

A.2. Instalación ambiente de emulación

En esta sección se detallarán las instalaciones necesarias obtener el ambiente de emulación.

A.2.1. Instalación MiniNExT

Para la instalación de MiniNExT, primero se tiene que realizar la instalación de Mininet. Es importante aclarar que a la fecha 01/09/16 MiniNExT no soporta la versión 4.x.x del kernel de Linux. Las pruebas se realizaron con la versión 3.x.x del mismo.

Instalación Mininet

La última versión de Mininet no es soportada por MiniNExT hasta la fecha (15/10/2016), por lo que es necesario instalar la versión 2.1.0. Para instalar esta versión en Ubuntu utilizar el siguiente comando:

```
$ sudo apt-get install mininet=2.1.0-0ubuntu1
```

Instalación MiniNExT

Para la instalación de MiniNExT es necesario descargar los paquetes de instalación. Los mismos se pueden encontrar en: `http://mininext.uscnsl.net/`. Una vez descargados, desde la carpeta donde se descomprimieron los archivos ejecutar los siguientes comandos:

```
$ make deps
```

Instalar estas dependencias mediante el comando:

```
$ sudo apt-get install `make deps`
```

Finalmente, se instala MiniNExT:

```
$ sudo make install
```

A.2.2. Instalación Quagga

Para poder ejecutar MiniNExT es necesario instalar Quagga mediante el siguiente comando:

```
$ sudo apt-get install quagga
```

A.2.3. Instalación exabg

Para la instalación del exabgp se deberá ejecutar el siguiente comando:

```
$ sudo apt-get install exabgp
```

Las versiones menores a la 3.4.13 tienen algunos *bugs* que causan problemas en la emulación. La versión 3.4.13 es la que utilizamos para realizar las emulaciones, por lo que recomendamos verificar que se tenga instalada esta versión o una más nueva. Para verificar la versión de exabgp instalada utilizar el comando:

```
$ exabgp -v
```

Si es necesario actualizar, utilizar el comando:

```
$ sudo pip install -U exabgp
```

A.2.4. Instalación bgpdump

Para instalar el bgpdump primero se debe descargar el mismo desde `https://bitbucket.org/ripencc/bgpdump/downloads`. Luego se deben

instalar algunas librerías necesarias para la instalación mediante el comando:

```
$ sudo apt-get install build-essencial zlib1g-dev
libbz2-dev automake
```

Luego, desde la carpeta donde se encuentra el software ejecutar los siguientes comandos:

```
$ sh ./bootstrap.sh
$ make
$ ./bgpdump -T
$ sudo cp bgpdump /usr/local/bin
```

Apéndice B

Manual de Usuario

A largo de este proyecto hemos desarrollado un conjunto de herramientas en *python* que automatizan el proceso de emulación y que agiliza el análisis de los datos recabados. Para facilitar el uso de las mismas hemos desarrollado también un interfaz gráfica ("TTools.py") muy simple que permite invocar cada una de las herramientas.

Figura B.1: TTools

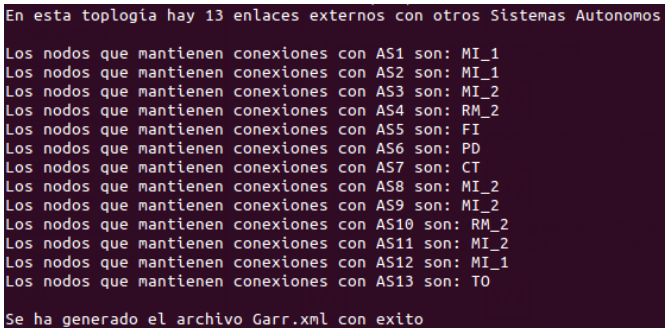


Figura B.2: Salida del script "GMLtoXMLconverter.py" - Conexiónes eBGP

Para abrir la interfaz simplemente debemos ejecutar `python TTools.py` en la linea de comandos. Como se puede ver en la imagen B.1 la interfaz consiste en una serie de botones donde cada uno tiene una breve descripción de lo que hace. A continuación veremos las opciones que nos brinda la interfaz.

- Convertir archivo GML a XML: EL lenguaje GML es el más frecuente utilizado para describir información geográfica. La mayoría de las topologías de red que podemos encontrar en la web las encontramos con la extensión ".gml". Como ya hemos visto para poder correr los algoritmos de localización es necesario otro formato. Esta opción nos permite convertir el archivo a un formato valido para luego poder correr el algoritmo de localización.
  Si queremos convertir un archivo debemos simplemente seleccionar el archivo con extensión "gml" (ver B.3) e ingresar el número de Sistema Autónomo (ASN) que va a tener la red (ver B.4). El resultado se guardará en la misma ubicación donde se encuentra el archivo original. También podemos ejecutar esta opción desde la linea de comando. Para esto debemos ejecutar `python GMLtoXMLconverter.py [topo.gml]`. En el caso de que el archivo gml contenga información de los enlaces inter-AS que tiene la red, esta será desplegada en la linea de comandos al finalizar la conversión del formato. En la imagen B.2 podemos ver el ejemplo de la conversión del archivo "Garr.xml".

- Preparar emulación de red: Cuando hablamos de preparar la emulación de red nos referimos a crear todos los archivos de configuración de cada uno de los dispositivos más la configuración física de la red. Además, se crea el

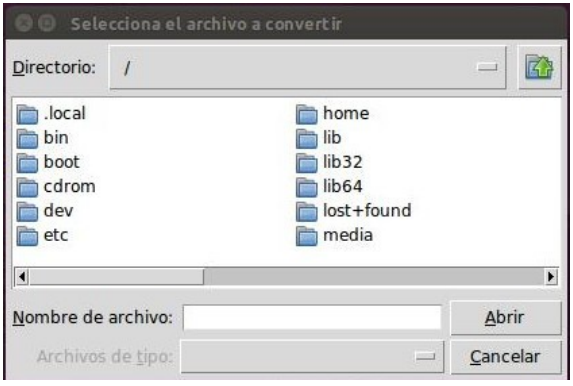


Figura B.3: Selección de archivo para convertir a formato adecuado

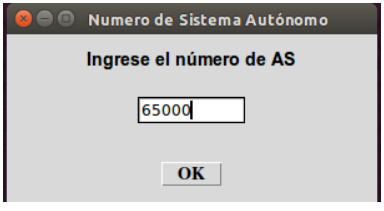


Figura B.4: Ingresar número de Sistema Autónomo

Figura B.5: Seleccionar routers de borde

Figura B.6: Seleccionar ASN vecino eBGP

script “start.py” que contiene la programación de los distintos eventos que participan en la emulación, como por ejemplo la inyección de trazas o el almacenamiento de la tabla “loc_rib” en disco al final de la ejecución.
Para poder preparar la emulación debemos previamente haber utilizado la herramienta RRLoc para correr el algoritmo de localización de RR.
Habiendo realizado lo anterior, para empezar, debemos seleccionar el archivo xml con la información de red. En la siguiente ventana podemos seleccionar la cantidad de sesiones eBGP, los routers de borde de nuestro AS (ver B.5) y los ASN de los vecinos (ver B.6). Esto generará una estructura de documentos, los cuales contendrán la configuración de los dispositivos participan en nuestro AS.

- Ejecutar MiniNExT: Una vez que hayamos preparado la emulación de red, podemos correr MiniNExT. Para esto debemos seleccionar la carpeta donde se encuentra los datos de la red que se ha generado previamente. Esto también se puede realizar desde la consola, simplemente ejecutamos el comando `python start.py` estando parado dentro de la carpeta generada y empezará la emulación. Dado que es necesario el ejecutar el comando con privilegios de superusuario, se debe escribir la contraseña del mismo para poder correr MiniNExT.

Figura B.7: Cargar datos en la base

- Cargar Bases de Datos Mysql: Esta opción nos permite cargar todos los datos en una base mysql. Los datos de las tablas rib_in de cada router son almacenados en binario con el formato MRT [49]. Antes de cargar los datos de las tablas rib_in utilizamos la herramienta bgpdump para traducir esta información.
  Los datos son almacenados en cuatro tablas: "loc_rib_ipv4", "loc_rib_ipv6", "rib_in" y "table_dump". Para esto debemos seleccionar la ubicación donde se almacenaron los resultados de la emulación, por defecto se encuentran en la carpeta "/var/log/mininext". Luego de esto se nos abre otra ventana que nos pide el nombre que se la va a dar a la base y el algoritmo utilizado en la emulación (ver B.7). Una vez completado estos campos comenzamos la carga de la base. Mientras se produce la carga, queda inhabilitado el resto de las opciones de la herramienta TTools.

Figura B.8: Selecciar trazas para descargar

Nota: Se deben configurar los parámetros de la base antes de utilizar esta herramienta. Para esto se debe editar el archivo "loadDB.py" y modificar los parámetros "HOST_NAME", "USER_NAME" y "PASSWORD".

- Borrar archivos generados por bgpdump: Para cargar las bases de datos es necesario hacer la traducción de los archivos rib_in. Una vez que se realiza la traducción estos archivos se mantienen junto a los archivos originales. Esto produce que se utilice más espacio en el disco de forma innecesaria.
  También se puede ejecutar por fuera del *framewok* a través de la consola escribiendo `python deleteTemporaryFiles` y seleccionado la carpeta donde esta ubicados los resultados de la ejecución.

- Descargar trazas BGP: Esta opción permite descargar la trazas BGP de la pagina "www.ripe.net" y convertirlos a texto plano para después ser utilizado en la emulación. Esta página brinda la posibilidad de descargar trazas BGP de diecisiete colectores [50] localizados en distintas partes del globo. Para descargar la traza debemos elegir la ubicación donde queremos que se descargue la traza. Luego debemos seleccionar el colector (estos son nombrados de la forma rrcXY), la fecha y el intervalo de tiempo (ver B.8). Las trazas descargadas deben llamarse PruebaRRCXY, donde XY indica el número de traza. Por ejemplo, "PruebaRRC03" sera la traza que se inyecte en la tercer sesión eBGP.

B.1. Posible caso de uso

La idea de esta sección es poder mostrar el proceso más común que llevamos a cabo para realizar una emulación.
Como primera instancia buscamos una topología de red que se encuentre en formato gml. En nuestro caso las obtuvimos de la pagina "The topology Zoo" [46]. Luego de descargar la topología es necesario convertir el archivo. Como vimos antes, para esto debemos seleccionar la opción "Convertir archivo GML a XML" y seguir los pasos que ya hemos mencionado. Las topologías obtenidas de la pagina "The topology Zoo" pueden, o no, contar con la información de los enlaces que mantiene la red con otros Sistemas Autónomos. Si posee esta información se desplegará en la linea de comandos cuáles son los routers de borde del AS que mantienen sesiones eBGP como se vió en la imagen B.2. Una vez con la topología en formato xml y con la ayuda de la herramienta RRLoc podemos correr los algoritmos de localización de reflectores de rutas. En esta etapa podremos seleccionar entre un *full-mesh* de sesiones iBGP o entre los algoritmos "Sep", "SepD", "SepS", "Bates", "BatesY", "BatesZ" y "Zhang". El resultado de esta ejecución será un archivo xml que contenga toda la información BGP de la red que queremos emular. Una vez que llegamos a este punto debemos configurar toda la red. Como ya hemos mencionado, esto involucra la configuración de enlaces, interfaces, protocolos entre otros. Para esto simplemente debemos seleccionar la opción "Preparar emulación de red". En este punto es donde será necesario la información correspondiente a las sesiones eBGP del AS.
Lo único que resta para poder llevar a cabo la emulación es conseguir las trazas BGP que queremos inyectar. Estas trazas pueden ser creadas por nosotros mismos, para estudiar el comportamiento ante ciertos mensajes, o podemos emular tráfico BGP real. Si queremos esto último, podemos obtener la trazas reales seleccionando

la opción "Descargar trazas BGP". Ahora si estamos en condiciones de emular la red. Como ya hemos visto lo podemos hacer con el botón "Ejecutar MiniNExT". Una vez que la red se esta emulando, sugerimos esperar a que el protocolo OSPF converja para comenzar a inyectar las trazas. Esto se puede chequear de una manera sencilla escribiendo el comando `sh ip route` dentro de alguno de los routers. Si se aprenden todas la direcciones de *loopback* de los routers quiere decir que OSPF ya convergió. Para poder inyectar la trazas debemos escribir `exit`, de esta forma salimos de la consola del MiniNExT para poder ejecutar lo que se haya programado en el archivo "start.py" y luego vuelve a la consola. Algunos segundos después veremos que se abren nuevas terminales, donde cada una de ellas se corresponde con una traza. En las mismas se puede ver todos los mensajes que son inyectados, tanto *updates* como *withraws*, y en caso de que se produzca algún error también es mostrado en la terminal.
Una vez terminada la inyección de trazas (el tiempo varía según el intervalo de tiempo que hemos seleccionado en la descarga de la traza), debemos escribir nuevamente `exit` en la consola. Esta acción copia los datos que nos interesan al disco y finaliza la emulación. Los datos serán almacenados en la carpeta "/var/log/mininext" por defecto.
Como última paso nos queda cargar los datos obtenidos en la base de datos. Como ya hemos mencionado anteriormente, algunos de estos datos es necesario traducirlos a texto plano antes de cargarlos a la base. La opción "Cargar base de datos Mysql" realiza la traducción de estos datos y luego carga la base.
Recomendemos, luego de cargar la base, borrar los archivos traducidos (los generados por bgpdump) ya que ocupan espacio con información duplicada. Esto se puede hacer de manera sencilla con el botón "Borrar archivos generados por bgpdump" que nos ahorra el tener que entrar a cada uno de los routers para borrar estos archivos. Cuando hablamos de topologías con cientos de routers significa un ahorro importante de trabajo.

Apéndice C

Consultas realizadas a la Base

En este apéndice se muestran las distintas consultas realizadas a la base cuyos resultados se han mostrado en el capitulo "Pruebas y Resultados".

- Cada vez que termina una emulación, luego de cargar la base hacemos algunos chequeos de manera de tener mayor certeza si la prueba fue realizada satisfactoriamente o si hubo algún problema. Como explicamos anteriormente, uno de los chequeos es comparar la cantidad de prefijos aprendidos, tanto prefijos IPv4 como IPv6. Para esto realizamos las siguientes consultas:

  ```
  SELECT count(*), router FROM loc_rib_ipv6_[ALGORITMO]
  GROUP BY BINARY router;

  SELECT count(*), router FROM loc_rib_ipv4_[ALGORITMO]
  GROUP BY BINARY router;
  ```

  El resultados de cada una de estas consultas es una lista con la cantidad de prefijos aprendidos por cada router.

- Otro aspecto que nos interesa verificar de la emulación es que efectivamente se inyecta trafico durante una hora. Para esto obtenemos el tiempo que se inyectó el primer mensaje, luego el tiempo que se inyectó el último mensaje a través de una sesión eBGP.

  ```
  SELECT date FROM rib_in_[ALGORITMO] ORDER BY date ASC
  LIMIT 1;
  ```

```
SELECT date FROM rib_in_[ALGORITMO] WHERE ipSource LIKE
"172.16.%" ORDER BY date DESC LIMIT 1;
```

También, puede ser muy útil obtener el último mensaje trasmitido en el AS, para esto debemos realizar la consulta:

```
SELECT date FROM rib_in_[ALGORITMO] ORDER BY date DESC
LIMIT 1;
```

- Para obtener uno de los datos que más nos interesa, la cantidad de mensajes trasmitidos dentro del Sistema Autónomo, realizamos la siguiente consulta para cada una de las topologías y cada uno de los algoritmos.

```
SELECT count(*) FROM rib_in_[ALGORITMO];
```

- Otro de los datos que obtenemos de las pruebas son los tiempos de propagación de prefijos IPv4 e IPv6, para esto utilizamos prefijos que son enseñados una única vez con un determinado *as_ path*. La manera de obtener este tipo de anuncios es con la siguiente consulta:

```
SELECT prefix, aspath , count(*) FROM rib_in_FM WHERE
prefix LIKE "%.%" GROUP BY prefix,  aspath  HAVING
count(*)=[CANT_ROUTERS] LIMIT  10;
```

La consulta se realiza a la tabla que tiene los datos de la configuración *full-mesh*. Esto es así porque podemos preguntar por un anuncio que ha sido recibido una única vez por cada uno de los routers.

Para el caso de IPv6, hacemos:

```
SELECT prefix, aspath , count(*) FROM rib_in_FM WHERE
prefix LIKE "%::%" GROUP BY prefix,  aspath  HAVING
count(*)=[CANT_ROUTERS] LIMIT  10;
```

A éstas consultas también se les aplicaba un *LIMIT 10* de manera de no obtener un único anuncio y tampoco obtener todos.

Como primera parte obtuvimos el tiempo de propagación del anuncio con la consulta:

```
SELECT MIN(minimum) AS min_date, MAX(minimum) AS max_date
FROM (SELECT min(date) AS minimum FROM rib_in_[ALGORTIMO]
WHERE prefix=[PREFIX] AND aspath=[ASPATH] GROUP BY BINARY
router ORDER BY date ASC) AS lowes_dates
```

Esta consulta nos devuelve el momento en que se inyectó el anuncio por una sesión eBGP y el momento en que el último router recibe el anuncio por primera vez. La resta es el tiempo que demora en propagarse el anuncio por todo el AS.

Posteriormente, consultamos la cantidad de veces que un anuncio llega repetido en promedio a cada router

```
SELECT count(*) , count(DISTINCT BINARY router),
count(*) / count(DISTINCT BINARY router) FROM rib_in_
[ALGORITMO] WHERE prefix=[PREFIX] AND aspath=[AS_PATH];
```

- Como comentamos, en algunas pruebas cometimos el error de registrar mensajes de tipo *withdraw* que no tienen que ser tenidos en cuenta. Como consecuencia, habían mensajes que aparecían luego de haber finalizado la prueba. Para detectar este tipo de comportamientos utilizamos la siguiente consulta:

```
SELECT t2.id, t2.router FROM rib_in_[ALGORITMO] AS t1,
rib_in_[ALGORITMO] AS t2 WHERE (t2.id=t1.id+1) AND
(BINARY t1.router = BINARY t2.router) AND
(t1.date<>t2.date) AND STR_TO_DATE(t2.date, '%m/%d/%y
%k:%i:%s') > DATE_ADD( STR_TO_DATE(t1.date, '%m/%d/%y
%k:%i:%s'), INTERVAL 2 MINUTE);
```

Esta consulta devuelve algún valor únicamente si hubo un intervalo de tiempo (en este caso dos minutos) donde no se transmitió ningún mensaje.

Glosario

Adj-RIB-out La tabla RIB-out contiene las rutas enviadas a cada uno de sus vecinos luego de aplicar las políticas BGP a las rutas almacenadas en la Local-RIB. Cada router tiene tantas RIB-out como vecinos BGP tiene..

Adj-RIB-in La tabla Adj-RIB-in (o RIB-in) contienen información de ruteo no procesada que le ha sido enseñada por cada unos sus pares BGP. Cada router tiene una RIB-in por cada vecino BGP .

API - Interfaz de programación de aplicaciones (*Application Programming Interface*) Es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ASN - Número de Sistema Autónomo (*Autonomous System Number*) En BGP, un sistema autónomo se identifica mediante su número de sistema autónomo globalmente único.

AS - Sistema Autónomo (*Autonomous System*) Un Sistema Autónomo (AS) es un grupo de redes de direcciones IP que son gestionadas por uno o más operadores de red que poseen una clara y única política de ruteo.La expresión Sistema Autónomo es con frecuencia interpretada incorrectamente como apenas una forma conveniente de agrupar redes que están bajo de una misma gestión. Sin embargo, en el caso en que hay más de una política de ruteo en el grupo, más de un AS es necesario.

BGP - *Border Gateway Protocol* Es actualmente el protocolo de enrutamiento entre sistemas autónomos estándar de facto en Internet.

BIRD - *BIRD Internet Routing Daemon* Es un protocolo de ruteo de Internet dirigido a sistemas Linux y Unix.

CIDR - Enrutamiento entre dominios sin clase (*Classless Inter-Domain Routing*) CIDR es un estándar de red para la interpretación de direcciones IP .

EGP - *Exterior Gateway Protocol* Hace referencia a los protocolos usados para intercambiar información de encaminamiento entre sistemas autónomos.

FIB - Tabla de reenvío (*Forwarding information base*) Es la tabla que contiene la información necesaria para reenviar datagramas IP. Como mínimo, contiene el identificador de interfaz y la información del siguiente salto para cada prefijo de red de destino accesible.

FTP - (*File Transfer Protocol*) Es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP.

GML - *Geography Markup Language* GML es un sublenguaje de XML descrito como una gramática en XML Schema. Es el formato más común para describir una red que se extiende por diferentes zonas geográficas.

IGP - *Interior Gateway Protocol* Hace referencia a los protocolos de enrutamiento interno de un AS y se utilizan para determinar cómo se lleva a cabo el enrutamiento dentro de un Sistema Autónomo.

IP - Protocolo de Internet (*Internet Protocol*) El protocolo de la capa de red de Internet es el protocolo IP. Este protocolo proporciona una comunicación lógica entre dos hosts.

IR - Registro de Internet (*Internet Registry*) Un IR es una organización responsable de la destribución de espaciones de direcciones IP a sus miembros o clientes, y del registro de esa distribución.

ISP - Proveedor de Servicios de Internet (*Internet Service Provider*) Es una organización que provee servicios para acceder y utilizar Internet.

Local-RIB La Local-RIB contiene la información resultante de aplicar el proceso de decisión BGP a partir de las tablas RIB-in. A diferencia de la RIB-in y la RIB-out la Local-RIB es única en cada router.

OSPF - *Open Shortest Path First* Es un protocolo de enrutamiento interno de los Sistemas Autónomos de Internet.

PoP - Punto de Presencia (*Point of Presence*) Dentro de la red de un ISP, los puntos en los que el ISP se conecta a otros ISP se conocen como Puntos de Presencia. Un PoP es simplemente un grupo de uno o más routers de la red del ISP en los que los routers de otros ISP o de las redes que pertenecen a los clientes del ISP pueden conectarse.

RFC - (*Request For Comments*) Son una serie de publicaciones que describen diversos aspectos del funcionamiento de Internet y otras redes de computadoras, como protocolos, procedimientos, etc. y comentarios e ideas sobre estos.

RIB (*Routing Information Base*) Es la base donde se almacena la información de ruteo del protocolo BGP.

RIP - *Routing Information Protocol* Es un protocolo de enrutamiento interno de Internet para los AS. Su algoritmo de enrutamiento está basado en el vector de distancias.

RIR - Registro Regional de Internet (*Regional Internet Registry*) Un RIR es una organización que supervisa la asignación y el registro de recursos de números de Internet dentro de una región particular del mundo. Los recursos incluyen direcciones IP (tanto IPv4 como IPv6) y números de Sistemas Autónomos.

RR - Reflector de Rutas (*Route Reflector*) Un reflector de rutas reenvía información de rutep que recibe a través de BGP a sus clientes.

TCP/IP - *Transport Control Protocol/Internet Protocol* Se denomina TCP/IP al la familia de protocolos de Internet, en referencia a los dos protocolos más importantes que la componen, que fueron de los primeros en definirse, y que son los dos más utilizados de la familia.

XML - *eXtensible Markup Language* XML es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el *World Wide Web Consortium* (W3C) y es utilizado para almacenar datos en forma legible.

XORP - *eXtensible Open Router Platform* Es un protocolo de ruteo de Internet *open source*.

Bibliografía

[1] Internet Society. The History of Internet. http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet. Último acceso: 21/10/16.

[2] LACNIC. Manual de Políticas (v2.6 01/08/2016). http://www.lacnic.net/web/lacnic/manual-3 Último acceso: 21/10/16.

[3] Geoff Huston. The 32-bit AS Number Report. http://www.potaroo.net/tools/asn32/. Último acceso: 21/10/16.

[4] K Lougheed and Y Rekhter. Border Gateway Protocol (BGP). Request for Comment RFC-1105. *Network Information Center*, 1989.

[5] Jon Mitchell. Autonomous System (AS) Reservation for Private Use. 2013.

[6] IANA. Autonomous System (AS) Numbers. http://www.iana.org/assignments/as-numbers/as-numbers.xhtml. Último acceso: 21/10/16.

[7] IANA. The Internet Assigned Numbers Authority (IANA). http://www.iana.org/. Último acceso: 22/09/16.

[8] Geert Jan de Groot, Yakov Rekhter, Daniel Karrenberg, and Eliot Lear. Address Allocation for Private Internets. 1996.

[9] S Deering. Host extensions for IP multicasting. *RFC-1112*, 1997.

[10] James F Kurose and Keith W Ross. *Computer networking: a top-down approach*. Addison Wesley, 2007.

[11] Vince Fuller and Tony Li. Classless inter-domain routing (cidr): The internet address assignment and aggregation plan. 2006.

[12] RIPE APNIC, ARIN. IPv6 Address Allocation and Assignment Policy. `https://archive.icann.org/en/aso/ipv6-statement-11jul02.htm`. Último acceso: 21/10/16.

[13] IANA. Number Resources. `https://www.iana.org/numbers`. Último acceso: 15/08/16.

[14] Danny McPherson and John G Scudder. Autonomous system confederations for BGP. 2007.

[15] T Bates, E Chen, and R Chandra. RFC 4456: BGP route reflection: an alternative to full mesh internal BGP (IBGP), 2006.

[16] Mark Winther. Tier 1 ISPs: What they are and why they are important. *IDC White Paper*, 2006.

[17] Jong Han Park. *Understanding the Impact of Internal BGP Route Reflection.* PhD thesis, University of California, Los Angeles, 2011.

[18] Randy Zhang and Micah Bartell. *BGP design and implementation.* Cisco Press, 2003.

[19] Jong Han Park, Ricardo Oliveira, Shane Amante, Danny McPherson, and Lixia Zhang. BGP route reflection revisited. *IEEE Communications Magazine*, 50(7):70–75, 2012.

[20] Ravi Musunuri and J Cobb. Stable iBGP through selective path dissemination. In *Fifteenth IASTED International Conference on Parallel and Distributed Computing and Systems*, volume 1, pages 99–104, 2003.

[21] Anindya Basu, Chih-Hao Luke Ong, April Rasala, F Bruce Shepherd, and Gordon Wilfong. Route oscillations in I-BGP with route reflection. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 235–247. ACM, 2002.

[22] Li Xiao, Jun Wang, and Klara Nahrstedt. Reliability-aware ibgp route reflection topology design. In *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, pages 180–189. IEEE, 2003.

[23] Tomas Klockar and Lenka Carr-Motycková. Preventing oscillations in route reflector-based I-BGP. In *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, pages 53–58. IEEE, 2004.

[24] Mythili Vutukuru, Paul Valiant, Swastik Kopparty, and Hari Balakrishnan. How to construct a correct and scalable iBGP configuration. 2005.

[25] Feng Zhao, Xicheng Lu, Peidong Zhu, and Jinjing Zhao. BGPSep_D: an improved algorithm for constructing correct and scalable IBGP configurations based on vertexes degree. In *International Conference on High Performance Computing and Communications*, pages 406–415. Springer, 2006.

[26] Feng Zhao, Xicheng Lu, Peidong Zhu, and Jinjing Zhao. BGPSep_S: an algorithm for constructing IBGP configurations with complete visibility. In *International Conference on Distributed Computing and Networking*, pages 379–384. Springer, 2006.

[27] Cristel Pelsser, Steve Uhlig, Tomonori Takeda, Bruno Quoitin, and Kohei Shiomoto. Providing scalable NH-diverse iBGP route re-distribution to achieve sub-second switch-over time. *Computer Networks*, 54(14):2492–2505, 2010.

[28] Marc-Olivier Buob, Steve Uhlig, and Mickael Meulle. Designing optimal iBGP route-reflection topologies. In *International Conference on Research in Networking*, pages 542–553. Springer, 2008.

[29] Marc-Olivier Buob, Mickael Meulle, and Steve Uhlig. Checking for optimal egress points in iBGP routing. In *Design and Reliable Communication Networks, 2007. DRCN 2007. 6th International Workshop on*, pages 1–8. IEEE, 2007.

[30] Totem project. TOolbox for Traffic Engineering Methods. `http://totem.run.montefiore.ulg.ac.be/`. Último acceso: 22/07/2015.

[31] Eladio Gutierrez, Diego Agriel, Elena Saenz, and Eduardo Grampin. RRLOC: A tool for iBGP Route Reflector topology planning and experimentation. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–4. IEEE, 2014.

[32] B Quotin. Official site of c-bgp network resolver, 2010. `http://c-bgp.sourceforge.net/`. Último acceso: 21/10/16.

[33] Bruno Quoitin and Steve Uhlig. Modeling the routing of an autonomous system with C-BGP. *IEEE network*, 19(6):12–19, 2005.

[34] B Lantz, Nikhil Handigol, Brandon Heller, and Vimal Jeyakumar. Introduction to Mininet, 2015.

[35] Brandon Schlinker. MiniNExT. `https://github.com/USC-NSL/miniNExT`. Último acceso: 22/07/2015.

[36] Andras Varga. The Omnet++ discrete event simulation system. Version 4.3. User Manual. 2013. `http://www.omnetpp.org`. Último acceso: 21/10/16.

[37] US Naval Research Laboratory. Core documentation. `http://downloads.pf.itd.nrl.navy.mil/docs/core/core_manual.pdf`. Último acceso: 22/07/2015.

[38] Brian Linkletter. CORE Network Emulator test drive. `http://www.brianlinkletter.com/core-network-emulator-test-drive/`, Último acceso: 15/08/16.

[39] Thomas Mangin. ExaBGP - A new Tool to Interact with BGP. `https://labs.ripe.net/Members/thomas_mangin/content-exabgp-new-tool-interact-bgp`. Último acceso: 05/09/16.

[40] Andrey Korolyov. BGPsimple: simple BGP peering and route injection script. `https://github.com/xdel/bgpsimple`. Último acceso: 10/09/2016.

[41] mininet.org/api. Mininet Python API Reference Manual. `http://mininet.org/api/classmininet_1_1net_1_1Mininet.html`. Último acceso: 03/07/15.

[42] RIPE NCC. RIPE Network Coordination Centre. `https://www.ripe.net/`. Último acceso: 21/10/16.

[43] nongnu.org. Quagga Routing Suite. `http://www.nongnu.org/quagga/`. Último acceso: 20/04/16.

[44] xorp Team. XORP. `http://www.xorp.org/`. Último acceso: 21/10/16.

[45] CZ.NIC Labs. The BIRD Internet Routing Daemon Project. `http://bird.network.cz/`. Último acceso: 21/10/16.

[46] University of Adelaide. The Internet Topology Zoo. `http://www.topology-zoo.org/dataset.html`. Último acceso: 10/03/16.

[47] Internet hall of fame. Randy Bush. http://www.internethalloffame.org/inductees/randy-bush. Último acceso: 21/10/16.

[48] E Rosen, A Viswanathan, and R Callon. RFC-3031. *Multiprotocol label switching architecture*, 2001.

[49] L Blunk, M Karir, and C Labovitz. Multi-threaded routing toolkit (MRT) routing information export format. Technical report, 2011.

[50] RIPE NCC. RIS Raw Data. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data. Último acceso: 21/10/16.

[51] Steve Uhlig and Sébastien Tandel. Quantifying the BGP routes diversity inside a tier-1 network. In *International Conference on Research in Networking*, pages 1002–1013. Springer, 2006.

[52] Pierre Francois, Bruno Decraene, Cristel Pelsser, Keyur Patel, and Clarence Filsfils. Graceful BGP session shutdown. 2014.

[53] Ashley Flavel and Matthew Roughan. Stable and flexible iBGP. *ACM SIGCOMM Computer Communication Review*, 39(4):183–194, 2009.