Localización de Reflectores de Rutas en un Sistema Autónomo de Internet

Proyecto de grado

2012

Emiliano Gutiérrez (emilianogfcm@gmail.com)

Diego Agriel (agrieldiego@gmail.com)

Emiliano Saenz (saenz_emi@hotmail.com)

Tutor: Eduardo Grampin (grampin@fing.edu.uy)

Instituto de Computación

Facultad de Ingeniería

Universidad de la República

Montevideo, Uruguay

Resumen

Internet es una red levemente jerárquica compuesta por más de 40.000 dominios interconectados entre sí llamados Sistemas Autónomos (AS). Dentro de cada dominio, un protocolo de routing interno (IGP) es utilizado para implementar el routing entre los routers del dominio. Entre distintos AS, la información de routing se intercambia mediante el protocolo BGP. Sesiones BGP externas (eBGP) son establecidas entre diferentes ASs, mientras que sesiones BGP internas (iBGP) son establecidas entre los routers internos a cada AS para difundir la información de routing obtenida por eBGP. En ASs con una gran cantidad de routers, utilizar una configuración de full mesh (colocar una sesión iBGP por cada par de routers) no escala (para n routers genera $\frac{n*(n-1)}{2}$ sesiones), por lo que suelen usarse reflectores de rutas [1]. Los reflectores de rutas permiten reducir el número de sesiones que genera una configuración de full mesh. Esta reducción en la cantidad de sesiones trae como efecto secundario algunos problemas como robustez, visibilidad incompleta, bucles y desviaciones en el envío de paquetes.

En este trabajo se realizó el diseño y la implementación integrada al framework TOTEM [2] de una herramienta de investigación compuesta por un conjunto de herramientas que facilitan la implementación de una solución al problema de localización de reflectores de rutas dentro de un AS. A su vez, se implementaron varios algoritmos existentes que intentan resolver el problema integrados a la herramienta y se ejecutaron sobre un conjunto de topologías reales. Finalmente, se presentó una prueba de concepto con el fin de incluir un ejemplo de ejecución de la herramienta.

Palabras claves: routing, Internet, reflector de ruta, BGP, full mesh, sistema autónomo, topología iBGP.

Contenido

[1]

Introducción

1.1 Introducción

Internet está compuesta por un conjunto de varios dominios llamados sistemas autónomos (ASs). Cada AS se compone de múltiples redes operadas por la misma autoridad. Un AS puede ser la red de una empresa, un proveedor de servicios de internet (ISP), o la red de una universidad. Dentro de cada AS, un protocolo de routing interno (IGP) como IS-IS [RFC 1142], OSPF [Version 2, RFC 2328] o RIP [Version 2, RFC 2435] se utiliza para propagar la información de routing. Entre ASs, un protocolo de routing externo (EGP) se utiliza para el intercambio de información de routeo. Hoy en día, Border Gateway Protocol (BGP) [BGP-4, RFC 4271] es el protocolo estándar de facto de routing entre dominios utilizado en Internet.

Un AS se compone de un conjunto de routers que están interconectados. Un AS posee dos tipos de enlaces, los enlaces centrales o "core links" que interconectan los routers dentro de un AS y los enlaces de borde o "edge links" que cruzan los límites del AS. Dado que los enlaces de borde se conectan a routers que se encuentran fuera de su red, un AS sólo controla un lado de estos enlaces. Los routers donde se establecen los enlaces de borde reciben el nombre de routers de borde del AS o "border routers".

A través de los enlaces de borde, un AS se conecta con diferentes tipos de AS vecinos. Mediante enlaces de acceso o "access links" se conecta a redes de clientes. Estas redes de clientes compran conectividad a Internet al AS. Mediante enlaces de interconexión o "peeering links" el AS se conecta a proveedores de tránsito y nodos privados. La mayoría de los ASs compran conectividad a Internet a uno o varios proveedores de tránsito.

La topología física de un AS define posibles caminos que se pueden utilizar para cruzar la red. Cómo cruza el tráfico por la red realmente depende de las decisiones tomadas por los protocolos de routing. Estas decisiones dependen de dos factores principales: las rutas disponibles y las configuraciones de los routers. Las rutas disponibles conocidas por un AS dependen de la información de routing recibida de ASs vecinos. Entre estas rutas disponibles, los protocolos de routing eligen cuál de ellas será utilizada para llegar a cada destino. Esta elección depende de los objetivos de los operadores de redes por medio de la configuración de los routers.

El objetivo del routing intradominio es encontrar el camino más corto de acuerdo con una métrica seleccionada. Los ISPs suelen utilizar una métrica proporcional al retardo de propagación a lo largo de la ruta o al ancho de banda. Algunos ASs grandes utilizan un IGP jerárquico, donde se divide al AS

en diferentes áreas. Dentro de un área se transmite toda la información de adyacencia, mientras que entre áreas se transmite sólo información sumarizada.

Además de IGP, un AS a veces utiliza el routing estático. Las rutas estáticas se utilizan con frecuencia en los enlaces de borde ya que los routers de ambos lados de estos enlaces no son operados por la misma autoridad. Las rutas estáticas también se utilizan para configurar el acceso a pequeños clientes que no utilizan BGP.

Finalmente, un AS ejecuta un protocolo de intercambio de información de routing como BGP. BGP es responsable de la selección de rutas hacia redes fuera del AS y garantizar que dicha información sea propagada a todos los routers del AS. Los objetivos del diseño de BGP son proporcionar accesibilidad entre dominios y la capacidad de que cualquier dominio pueda utilizar sus propias políticas de routing (es decir, controlar el tráfico que entra y sale del dominio). BGP no optimiza una métrica global, sino que se basa en una secuencia de decisiones (más adelante se detalla este proceso).

A un router que ejecuta BGP lo llamaremos router BGP. Los routers BGP intercambian información de routing mediante sesiones BGP. Cada sesión BGP se establece entre un par de routers a través de una conexión TCP. Sesiones BGP externas (eBGP) se establecen a través de enlaces de borde, mientras que sesiones BGP internas (iBGP) se establecen entre los routers internos del AS. En ASs grandes, utilizar una configuración de full mesh (colocar una sesión iBGP por cada par de routers del AS, lo que para n routers genera $\frac{n*(n-1)}{2}$ sesiones) no escala, por lo que suelen usarse confederaciones BGP [RFC 5065] o reflectores de rutas [RFC 4456], este último núcleo central de este proyecto. Los reflectores de rutas permiten reducir el número de sesiones que genera una configuración de full mesh. Esta reducción en la cantidad de sesiones trae como efecto secundario algunos problemas como robustez, visibilidad incompleta, bucles y desviaciones en el envío de paquetes.

Con los problemas ya mencionados que poseen las configuraciones con reflectores de rutas, la elección de que routers del AS serán reflectores no es trivial. Este proyecto se centra en el estudio del problema de localización de reflectores de rutas dentro de un AS.

1.2 Objetivos del proyecto

El objetivo principal del proyecto es el de desarrollar una herramienta que facilite la implementación de nuevos algoritmos que tengan como objetivo solucionar el problema de localización de reflectores de rutas dentro de un sistema autónomo. Esta herramienta debe permitir la evaluación de dichos algoritmos y la simulación con herramientas ya existentes. A su vez, es de interés estudiar, implementar y evaluar varios algoritmos ya existentes integrados a la herramienta.

Durante el desarrollo del proyecto se estudiará y revisará el estado del arte compuesto por una serie de documentos relacionados al tema para lograr una mejor comprensión del problema.

[2]

Estado del arte

2.1 Border Gateway Protocol (BGP)

El Border Gateway Protocol (BGP) es un protocolo de routing entre sistemas autónomos. Un Sistema Autónomo (AS) es un conjunto de routers bajo una única administración. Un AS utiliza un protocolo de routing interno (IGP) y varias métricas para determinar cómo enrutar paquetes dentro del AS, y otro protocolo de routing entre ASs para determinar la forma de enrutar los paquetes a otros ASs.

La motivación principal de BGP es el intercambio de información de routing con otros sistemas. Resumidamente esta información está compuesta por una lista de prefijos, una lista de sistemas autónomos a atravesar para alcanzar esos prefijos y finalmente el next-hop (último router en anunciar la información). Esta información es suficiente para determinar las rutas a tomar para cada prefijo externo.

La información de routing intercambiada a través de BGP sólo es compatible con el paradigma de forwarding basado en el destino, que supone que un router reenvía un paquete basándose únicamente en la dirección IP de destino contenida en la cabecera del paquete. Esto, a su vez, determina el conjunto de políticas y decisiones que pueden ser aplicadas utilizando BGP.

BGP utiliza TCP [RFC 793] como su protocolo de transporte. Esto elimina la necesidad de implementar explícitamente la fragmentación, retransmisión, confirmación de recepción y la secuenciación en los mensajes enviados. BGP escucha en el puerto TCP 179. El mecanismo de notificación de errores utilizado en BGP asume que TCP soporta un cierre "correcto" (es decir, que todos los datos pendientes se entregarán antes de que la conexión sea cerrada). Una conexión TCP se forma entre dos routers. Éstos intercambian mensajes para abrir y confirmar los parámetros de conexión.

Después de que se establece la conexión TCP, el primer mensaje enviado por cada par es un mensaje OPEN. Si el mensaje OPEN es correcto, se confirma enviando un mensaje KEEPALIVE. Éstos últimos deben ser enviados periódicamente para asegurar que la conexión está activa. Por otra parte, mensajes NOTIFICATION se envían en respuesta a errores o condiciones especiales. Finalmente, rutas son anunciadas entre pares BGP mediante mensajes UPDATE. Si una conexión encuentra un error, un mensaje NOTIFICATION se envía y la conexión es cerrada.

BGP posee dos modos diferentes de funcionamiento. External BGP (eBGP) intercambia información de routing entre sistemas autónomos, mientras Internal BGP (iBGP) intercambia información de routing hacia prefijos externos dentro del sistema autónomo. iBGP no fue diseñado para mantener alcanzables destinos dentro del sistema autónomo. Esta tarea es normalmente realizada por un protocolo de routing interno (IGP) como OSPF, IS-IS y RIP. La principal tarea de iBGP es propagar rutas externas dentro del AS. El routing BGP de las rutas externas interactúa con el routing interno (el proceso de selección de

rutas de iBGP puede preferir una ruta sobre otra porque ésta tiene asociada una "distancia" IGP menor con el punto de egreso).

La selección de la mejor ruta es realizada en cada router iBGP de acuerdo con el proceso de decisión de BGP que se encuentra resumido a continuación:

1. Preferir las rutas con el mayor LOCAL_PREF
2. Preferir las rutas con el AS_PATH más corto
3. Preferir las rutas con el menor ORIGIN
4. Entre las rutas recibidas desde el mismo AS vecino, preferir aquellas que tengan el menor MED
5. Preferir las rutas aprendidas vía eBGP
6. Preferir las rutas con menor costo IGP
7. Preferir las rutas que tengan el menor identificador BGP
8. Preferir la ruta que proviene del vecino con la dirección IP menor

Un emisor BGP utiliza el atributo LOCAL_PREF para informar a los demás pares internos el grado de preferencia para una ruta anunciada.

El atributo AS_PATH identifica los ASs que ha atravesado la información de routing contenida en un mensaje UPDATE.

El atributo ORIGIN es generado por el par que genera la información de routing asociada al mensaje. No debería ser cambiado por los demás routers.

El atributo opcional MED es utilizado en enlaces externos para discriminar entre múltiples puntos de entrada o egreso al mismo AS vecino.

Cada router BGP asigna al identificador BGP una dirección IP de las que le han sido asignadas. Este identificador es el mismo para todas las interfaces locales del router.

El proceso de decisión de BGP consiste en un conjunto de reglas, cuando hay empates para una regla, la siguiente es aplicada. La evaluación de los pasos de 1-4 es la misma para cada router iBGP ya que estos pasos consideran atributos globales, que usualmente no son modificados por iBGP.

2.2 Reflectores de rutas

Típicamente, todos los routers BGP dentro de un mismo AS son configurados en full mesh y cualquier nueva ruta externa debe ser redistribuida a todos los otros routers del AS. Para n routers BGP dentro de un AS se deben mantener $\frac{n*(n-1)}{2}$ sesiones BGP internas (iBGP) distintas.

Esta configuración de full mesh claramente no escala cuando hay muchos routers iBGP. Cada router BGP debe mantener una gran cantidad de sesiones TCP con el resto de los routers BGP en el AS. Esto además implica que cada router BGP debe mantener una copia de la tabla de routing de cada vecino.

Este problema de escala es bien conocido y se han presentado varias propuestas para resolverlo. Una de estas alternativa para evitar la necesidad de una configuración full mesh es conocida como "route reflectors" o reflectores de rutas.

En el enfoque tradicional de BGP las rutas aprendidas por iBGP no son propagadas a los demás routers. Utilizando reflectores de rutas se permite que un router BGP (conocido como un "reflector de rutas") pueda publicar rutas iBGP aprendidas a otros pares iBGP. Esto representa un cambio en el concepto de iBGP, e implica el agregado de dos nuevos atributos BGP no transitivos opcionales para evitar bucles en las actualizaciones de información de routing.

Los reflectores de rutas han sido diseñados para satisfacer los siguientes criterios:

- Sencillez

  La solución debe ser fácil de configurar y fácil de entender.

- Transición fácil desde una configuración de full mesh

  Debe ser posible hacer la transición desde una configuración de full mesh sin la necesidad de cambiar la topología del AS.

- Compatibilidad

  Debe ser posible que coexistan con los nuevos nodos, nodos iBGP clásicos, sin pérdida de información de routing BGP.

La idea básica de funcionamiento de una configuración con reflectores de rutas es disminuir la cantidad de sesiones con respecto a una configuración de full mesh, asignando de manera estratégica una cierta cantidad de routers de la topología como reflectores. Estos reflectores se encargarán de anunciar las rutas aprendidas a otros routers. De esta manera, algunas sesiones existentes en una configuración de full mesh pasan a ser innecesarias y pueden ser eliminadas.

Terminología y conceptos

Se define una ruta como una secuencia de nodos, pudiendo ser éstos hosts o routers. Con esto en cuenta, usamos el término "reflector de rutas" para describir el funcionamiento de un router encargado de anunciar una ruta iBGP aprendida a otros routers. Un router como el descripto se dice que es un "reflector de rutas" (RR), y a una ruta de ese tipo se le llama ruta reflejada.

Los pares de un RR se dividen en dos grupos:

1) Pares Clientes

2) Pares No Clientes

Un RR refleja rutas entre estos grupos. Un RR junto con sus clientes forman un clúster.

Los no clientes deben estar configurados en full mesh, aunque los clientes no tienen por qué estarlo.

Modo de operación

Cuando un RR recibe una ruta desde un par iBGP, selecciona la mejor ruta en función de sus reglas de selección de ruta. Después de que la mejor ruta es seleccionada, se debe hacer lo siguiente, dependiendo del tipo de par del que se recibió la ruta:

1) Si la ruta proviene de un No Cliente:

Reflejar la ruta a todos los clientes.

2) Si la ruta proviene de un Cliente:

Reflejar la ruta a todos los pares, clientes y también no clientes. (De ahí que los clientes no están obligados a estar configurados en full mesh.)

Un Sistema Autónomo puede tener muchos RR. Un RR trata a otros RR al igual que a cualquier otro par BGP clásico. Un RR puede ser configurado para tener otros RR como clientes o no clientes.

En una configuración sencilla con reflectores de rutas, el backbone de la red se puede dividir en varios clústers. Cada RR se configura como no cliente con los otros RR (por lo tanto todos los RR estarán en una configuración de full mesh). Los clientes serán configurados para mantener una sesión iBGP solo con el RR en su clúster. Debido a los reflectores de rutas, todos los pares iBGP recibirán la información de routing.

Cambios en el proceso de decisión de BGP

El proceso de decisión de BGP es modificado de la siguiente manera:

- Si una ruta lleva el atributo ORIGIN, en el paso 7 del proceso de decisión el ORIGIN debe ser tomado como el identificador BGP del par BGP que anunció la ruta.
- Además, la siguiente regla debe insertarse entre los pasos 7 y 8 del proceso de decisión: un par BGP debe preferir una ruta con el CLUSTER_LIST de menor longitud. El CLUSTER_LIST es un atributo compuesto por una secuencia de CLUSTER_ID que representan el camino de clústers por los que la ruta ha pasado. Se toma longitud cero si una ruta no tiene asignado dicho atributo.

2.3 Características deseables que cumple una configuración full mesh

Visibilidad completa: La transmisión de la información entre los routers debe ser "completa" en el sentido de que, por cada ruta externa, cada router elige la misma ruta que hubiera elegido si hubiera visto las mejores rutas desde todos los demás routers eBGP en el AS.

Routing libre de bucles: Después de la transmisión de las rutas eBGP aprendidas, las rutas resultantes (y los caminos de los paquetes enviados a través de esas rutas) elegidas por todos los routers deberán estar libres de desviaciones y bucles de routing.

Robustez a fallas IGP: El mecanismo de transmisión de rutas debe ser robusto frente a fallas de nodos o enlaces y además cambios en el costo IGP de una ruta no deben dar lugar a una violación de la propiedad anterior.

Es deseable que cualquier configuración iBGP elegida, ya sea con reflectores de rutas o no, cumpla estas propiedades. A continuación veremos que las configuraciones con reflectores de rutas no necesariamente cumplen con estas tres propiedades y presentan varios problemas si no se elige cuidadosamente la topología iBGP.

2.4 Problemas con reflectores de rutas

En la práctica, configuraciones iBGP que utilizan reflectores de rutas no satisfacen necesariamente las propiedades de correctitud mencionadas anteriormente. Ofrecemos a continuación algunos ejemplos de cómo estas propiedades pueden ser violadas en algunas de éstas configuraciones.

- Visibilidad completa: En una configuración iBGP utilizando reflectores de rutas, un reflector de rutas refleja a sus clientes solo su mejor ruta (y no todas las rutas que aprende) y por lo tanto todos los routers no eligen las mismas rutas que habría elegido si tuviera visibilidad sobre todas las rutas aprendidas por eBGP. Notar que esta propiedad nunca podría ser violada en una configuración de full mesh.

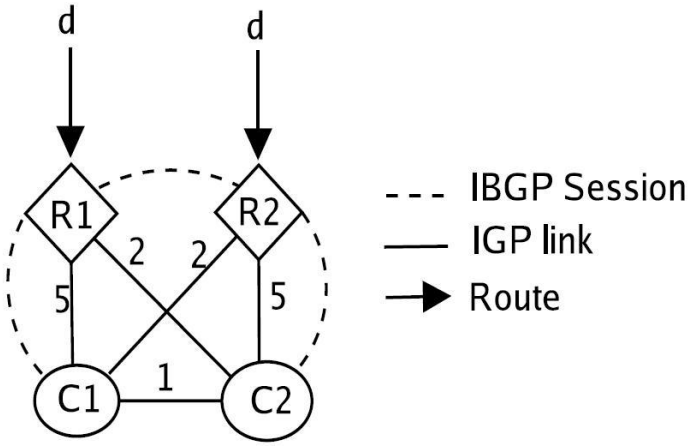


Figura 2.1: Una configuración iBGP incorrecta

Por ejemplo, considere la configuración iBGP que se muestra en la Figura 2.1. El reflector de rutas R1 y su cliente C1 constituyen un clúster, al igual que R2 y C2. Los grafos IGP e iBGP son como se muestran en la figura. Dos rutas, que empatan en el proceso de decisión de BGP hasta la etapa de comparación de costos IGP al next hop, llegan a R1 y R2. R1 y R2 eligen las rutas a través de sí mismos como las mejores rutas y las anuncian a sus clientes. C1 elige la ruta a través de R1 y C2 la ruta a través de R2. Sin embargo, si C1 hubiera aprendido todas las rutas eBGP, habría elegido la ruta a través de R2 porque C1 tiene un menor costo IGP a R2.

La propiedad de visibilidad completa es importante para un routing eficiente y predecible. Si esta propiedad es violada, el sistema fallará al implementar un routing de "hot potato" (intentar pasar el tráfico a otros ASs lo más rápido posible), causando que sean desperdiciados recursos de la red. Además, la predicción del resultado del complejo proceso de decisión de BGP es mucho más fácil cuando hay visibilidad completa, ya que se garantiza que cada router escoge la ruta que habría elegido si hubiera visto todas las rutas aprendidas por eBGP. Tales predicciones han probado ser de utilidad para el modelado BGP y para la ingeniería de tráfico.

- Routing libre de bucles: Las configuraciones iBGP con reflectores de rutas son susceptibles a anomalías de routing tales como las desviaciones y los bucles, que hacen que las redes sean más difíciles de mantener y depurar. En cada router, BGP selecciona solo el router de salida para un destino, mientras que la trayectoria de envío real desde un router hacia la salida es determinada por IGP. Algunos router en el camino más corto hacia la salida pueden elegir una salida diferente para el mismo destino externo, haciendo que los paquetes sean desviados a lo largo del camino. Desviaciones múltiples pueden producir bucles de routing persistentes.

  Por ejemplo, véase de nuevo la Figura 2.1. Recordemos que C1 elige la ruta a través de R1 y C2 la ruta a través de R2. El camino más corto de C1 a R1 pasa a través de C2 y el camino más corto de C2 a R2 pasa a través de C1. Entonces, cuando C1 envía los paquetes destinados a *d* a C2 (con la intención de que deben llegar a R1), C2 los envía de vuelta a C1, porque su camino elegido hacia *d* es C2-C1-R2. Cualquier paquete destinado a *d* que llegue a C1 o C2 se quedará atascado en un bucle.

- Robustez a fallas IGP: En configuraciones con reflectores de rutas arbitrarias, las propiedades de correctitud como ser libre de bucles de routing pueden ser violados si ocurren cambios en los costos de los enlaces IGP o si ocurre un fallo en un router o enlace.

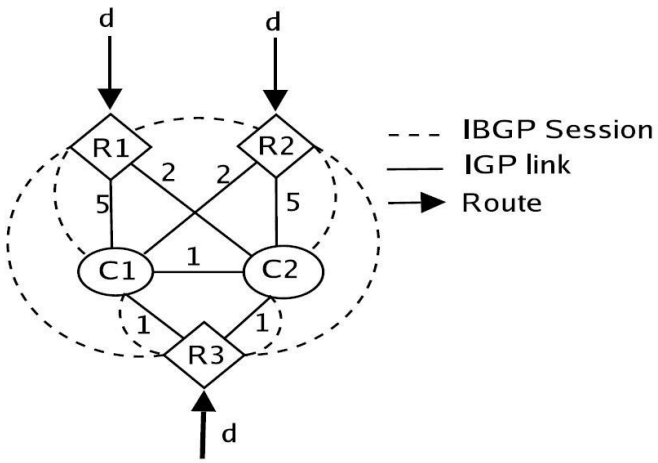


Figura 2.2: Una configuración iBGP utilizando reflectores de rutas que tiene baja tolerancia a fallos.

Por ejemplo, considere la configuración iBGP que se muestra en la Figura 2.2. Esta topología IGP es similar a la topología de la Figura 2.1, con la adición de un nuevo reflector de rutas R3, del que tanto C1 y C2 son clientes. Tanto C1 y C2 eligen a R3 como su next hop para el destino *d* y no hay desviaciones en el camino a R3. Cuando R3 falla, la topología, ahora equivalente a la de la Figura 2.1, tiene un bucle de routing. Por lo tanto, es difícil garantizar que una topología iBGP arbitraria es libre de bucles de routing y que posee visibilidad completa si ocurre el fallo de un nodo o enlace. Por la misma razón, es inherentemente difícil construir topologías

con reflectores de rutas con redundancia, porque si un reflector de ruta falla, el "backup" del reflector de rutas puede llegar a causar bucles de routing.

2.5 Correctitud de una configuración iBGP

En esta sección se estudiarán los problemas que pueden ocurrir en una configuración iBGP y se presentarán dos propiedades que tiene que cumplir cualquier configuración iBGP para ser correcta. Esta subsección se basa en el trabajo presentado en [18].

Como ya hemos mencionado, BGP posee dos modos diferentes de funcionamiento. External BGP (eBGP) intercambia información de accesibilidad entre sistemas autónomos, mientras Internal BGP (iBGP) intercambia información de accesibilidad hacia prefijos externos dentro del sistema autónomo. iBGP no fue diseñado para mantener alcanzables destinos dentro del sistema autónomo. Esta tarea es normalmente realizada por un Interior Gateway Protocol (IGP) como OSPF, IS-IS y RIP. La principal tarea de BGP es permitir que fuentes internas alcancen destinos externos y que fuentes externas alcancen destinos internos. El routing BGP de las rutas externas interactúa con el routing interno (el proceso de selección de rutas de iBGP puede preferir una ruta sobre otra porque esta tiene asociada una "distancia" IGP menor con el punto de egreso). Estudiaremos dos tipos de anomalías en el comportamiento que pueden surgir de esta interacción.

Ambos, eBGP e iBGP difieren en el tratamiento de ciertos atributos y en la manera en que los mensajes BGP son intercambiados. Para nuestro estudio, el modelo de intercambio de mensajes iBGP será de suma importancia. eBGP normalmente asume que una sesión BGP se establece entre dos routers que comparten una red física. Por esto, los mensajes eBGP normalmente no son enrutados. Por otra parte iBGP es enrutado dentro del sistema autónomo utilizando la conectividad provista por el IGP. Con eBGP puro, los mensajes de señalización y el flujo de tráfico son intercambiados a través del mismo camino pero en direcciones opuestas. Para describir este escenario asumiremos que los caminos son simétricos. Ya que iBGP es enrutado por el IGP, no está garantizado que el intercambio del flujo de datos y los caminos señalizados sean simétricos.

Nos centraremos en las anomalías de routing causadas por la asimetría de estos caminos. Se sabe que la oscilación de las rutas puede ocurrir en iBGP debido al uso del atributo Multi Exit Discriminator (MED). Sin embargo, ha sido observado que las oscilaciones inducidas por el uso del atributo MED tienen una causa diferente. En lo que resta de nuestro análisis ignoraremos la existencia del atributo MED.

La primera anomalía de iBGP que estudiaremos ha sido cuidadosamente estudiada para eBGP (el hecho que el protocolo puede fallar y no converger). Es decir, en contraposición a una red estable, los routers BGP pueden intercambiar mensajes por siempre sin lograr establecer un conjunto estable de rutas. Veremos que, aún en ausencia del MED, iBGP puede diverger como resultado de caminos asimétricos.

El segundo tipo de anomalías iBGP que estudiaremos implican inconsistencias en los caminos dentro del sistema autónomo. Cuando un router BGP selecciona la mejor ruta a un destino externo, esta ruta tiene un valor para el próximo salto BGP que identifica un punto de egreso particular desde el sistema autónomo. El camino desde el router hacia este punto de egreso es proporcionado por el routing IGP. Sin embargo, es posible que suceda que a lo largo del camino otro router BGP haya elegido un punto de egreso diferente para el mismo destino externo. En este caso podemos decir que el camino ha sido desviado. Nuevamente, este problema surge debido a asimetrías. Las desviaciones pueden tornar muy complicado la detección de problemas de routing, y en el peor caso en que múltiples desvíos se combinen pueden formar forwarding loops (bucles en el intercambio de los paquetes a través de la red).

Consideraremos que la configuración de la red de un sistema autónomo está compuesta por la topología de la red, las métricas IGP, las sesiones iBGP, y las políticas de routing de BGP. Dada una configuración de red, ¿qué puede conducirnos a decir que ésta es libre de las anomalías mencionadas? Los administradores de la red tienen completo control sobre la configuración de su red, pero tienen poco control sobre los mensajes enviados por sus vecinos externos. Por esta razón, puede ayudar la visualización de la configuración de nuestra red como un procedimiento que toma como entrada mensajes de routing externos y produce como salida un routing estable para el sistema autónomo. La correctitud de la configuración entonces es tomada como la correctitud de este procedimiento (necesitamos que, para todo posible conjunto de entradas, el procedimiento termine con un routing estable que es libre de desviaciones). Llamaremos a estas configuraciones correctas. Nótese que esto es esencialmente una forma de definir una configuración para un sistema autónomo aislado sin la necesidad de asumir nada en cuanto a lo que routing global se refiere.

Adicionalmente para proveer un marco de discusión para la correctitud de una configuración iBGP, mencionaremos algunos resultados complejos concernientes a la correctitud de una configuración. En particular veremos que determinar si una configuración converge para toda posible entrada es un problema NP-hard. Veremos que decidir si, para todas las entradas, la configuración determina un routing libre de desviaciones es NP-hard también. Para los dos tipos de anomalías de routing, mencionaremos condiciones suficientes que garantizan la correctitud.

Nos concentraremos en las configuraciones iBGP que utilizan reflectores de rutas. Por otra parte, las confederaciones BGP son otra alternativa que permiten generar configuraciones iBGP escalables. Aunque las confederaciones pueden introducir asimetrías en los caminos que pueden conducir a las anomalías de routing esbozadas aquí, no las cubriremos explícitamente ya que muchos de los argumentos son similares.

Creemos que las observaciones que aquí se presentan son significantes más allá de los detalles técnicos estrechamente relacionados con la configuración de

BGP. Los diseñadores de protocolos deben ser conscientes de las potenciales anomalías que pueden surgir cuando las relaciones entre los caminos utilizados para el intercambio de datos (forwarding) y los utilizados en el plano de control (signaling path, será definido posteriormente) no están restringidas.

Como hemos mencionado, en este capítulo veremos cuando iBGP es correcto, y nos centraremos en la interacción entre iBPG y el routing interno. Aunque algunos de los ejemplos que mostraremos no surgen directamente de la práctica, pueden mostrar que la implementación de algoritmos para el chequeo automático de la correctitud de la configuración de iBGP puede ser más complejo de lo esperado. Adicionalmente, las malas configuraciones de BGP son muy comunes, e interactúan con las fallas en la red de formas sorprendentes e inesperadas.

Estudiaremos un solo sistema autónomo que puede o no utilizar reflectores de rutas. Un sistema autónomo consiste en una red de routers conectados por alguna red física subyacente donde a cada enlace físico le ha sido asignado una distancia. Los routers intercambiarán información de routing vía iBGP. En cualquier momento, cualquier cantidad de routers de borde pueden anunciar rutas eBGP dentro del sistema autónomo. Llamaremos a la descripción de la conectividad física, la conectividad iBGP, y los routers de borde la *configuración* del sistema autónomo. Todo nuestro análisis es realizado en términos de un destino externo particular, que llamaremos el *origen*.

Las rutas externas que son inicialmente anunciadas dentro del sistema autónomo serán llamadas *caminos de egreso* y los routers de borde que anuncien estas rutas serán llamados *nodos de egreso*. En presencia de reflectores de rutas un clúster refiere al conjunto de reflectores de rutas R y el conjunto de clientes C donde cada router de R actúa como reflector para cada cliente en C.

Cada mensaje de actualización de una ruta tiene un número de atributos asociados y la ruta elegida por cada router está basada en estos atributos. El procedimiento estándar de selección de rutas de BGP selecciona primero aquellas rutas con los valores de LOCAL_PREF mayores, de estas selecciona aquellas con menor largo en AS_PATH y de estas últimas selecciona las que tienen menores valores de MED (para aquellas rutas con el mismo next hop). Para cada instante de tiempo, puede haber un conjunto de caminos de egreso para el origen y consideraremos el subconjunto de estos caminos que no han sido filtrados por tener una bajo valor en el LOCAL_PREF, un largo de AS_PATH elevado, y un valor que MED que es mayor a alguno de los otros caminos de egreso al mismo sistema autónomo. Así, de acuerdo con las reglas de preferencia para la selección de rutas de iBGP, la elección de cada router en el sistema autónomo de los caminos de egreso que restan es tomada en base a:

i. Primero selecciona los caminos de egreso aprendidos por eBGP sobre los aprendidos por iBGP.

ii. De los anteriores, selecciona aquellos nodos de egreso más cercanos de acuerdo con los valores de distancia asignados por los enlaces en la red física.

iii. Finalmente, para los caminos de egreso restantes, se utiliza el largo de la lista CLUSTER_LIST (lista de los clúster que el mensaje ha atravesado) y el pid (el id del router desde dónde el anuncio de la ruta fue recibido).

Se debe tener en cuenta que iBGP no requiere que cada router en el sistema autónomo elija el mismo punto de egreso para un destino dado.

Configuración iBGP

Definiremos una configuración iBGP C como el par $C = (G_p, G_s)$, donde G_p es el grafo físico y G_s es el grafo de señalización.

Definamos el grafo físico. Primero definiremos un grafo con pesos $G_p = (V, B, E_p, d)$ que llamaremos el grafo físico para representar la topología física subyacente. Cada nodo $u \in V$ representa a un router. Vamos a referirnos a menudo al router u haciendo referencia al router representado por el nodo u. El conjunto $B \subseteq V$ es el conjunto de routers de borde que tienen conectividad física con redes externas. Existe la arista $e = \{u, v\} \in E_p$ si hay un enlace físico entre el router representado por u y por v. La distancia $d(e)$ de una arista es la distancia asignada al enlace físico y que es típicamente utilizada por protocolos como OSPF para determinar el camino más corto entre dos nodos. En general, indicaremos un camino P como una secuencia de nodos:

$$P = u_1 u_2 \dots u_n$$

Concatenaremos los caminos $P = u_1 u_2 \dots u_n$ y $Q = v_1 v_2 \dots v_n$ para formar un nuevo camino $PQ = u_1 u_2 \dots u_n v_1 v_2 \dots v_n$ si $u_n \neq v_1$ o $PQ = u_1 u_2 \dots u_n v_2 \dots v_n$ si $u_n = v_1$. Definiremos el largo del camino P como la suma de las distancias $d(e)$ sobre todas las aristas e de P. Sea P un camino entre los nodos u y v tal que no hay otro camino entre los nodos u y v cuyo largo es menor al de P. Entonces el largo de P es escrito como $M(u, v)$ (o $M(v, u)$). Tal camino es llamado el camino más corto entre u y v. Para todo par de nodos u y v definimos $sp(u, v)$ en G_p como el camino más corto entre u y v y asumimos que todos los caminos son consistentes en el sentido que si $sp(v_1, v_t) = v_1 v_2 \dots v_t$ entonces $sp(v_i, v_j) = v_i v_{i+1} \dots v_j$ donde $1 \leq i \leq j \leq t$ y $sp(v_t, v_1) = v_t v_{t-1} \dots v_1$.

Definiremos el grafo de señalización $G_s = (V, A_s)$ como sigue. Un arco en G_s representa una sesión iBGP entre dos routers.

Puesto que las sesiones BGP son implementadas por sesiones TCP que deben ser enrutadas, los arcos en G_s corresponden a algún camino enrutado en G_p. El conjunto de aristas A_s será particionado en tres subconjuntos OVER, DOWN y

UP. Un arco $(u,v) \in OVER$ representa el hecho que hay una sesión iBGP común permitiendo el flujo de mensajes de u a v. Este tráfico enrutado (utilizando el routing interno) no requiere que exista un enlace físico que lo realice. Típicamente esperaremos tener el arco $(u,v) \in OVER$ cuando también $(v,u) \in OVER$. Un arco $(u,v) \in DOWN$ indica la señalización de mensajes de actualización de rutas desde el reflector de rutas u a uno de sus clientes v. En este caso, u debe enviar mensajes de actualización de rutas para todas las rutas excepto las anunciadas por v. Finalmente, un arco $(u,v) \in UP$ representa la señalización de mensajes de actualización de rutas desde un cliente u hasta al router v que actúa como un reflector para v. En este caso, u envía mensajes de actualización solo de rutas que son anunciadas dentro del sistema autónomo. Típicamente esperaremos que $(u,v) \in DOWN$ si y solo si $(v,u) \in UP$. Un camino de señalización válido S satisface las siguientes propiedades. El camino S puede ser particionado en tres subcaminos tal que $S = PQR$, donde $P = p_1 p_2 \dots p_a$ para algún $a \geq 0$ donde cada $p_i \in UP$, $R = r_1 r_2 \dots r_b$ para algún $b \geq 0$ donde cada $r_i \in DOWN$ y Q es vacío o contiene una único arco $q \in OVER$. Nótese que cualquiera de los conjuntos P, Q o R puede ser vacío.

Los operadores de red tienen control sobre las configuraciones de sus redes. Sin embargo, no tienen control sobre las rutas que les son enviadas por sus vecinos. Rutas para un destino dado pueden llegar a un subconjunto de los routers de borde, $X \subseteq B$. Podemos pensar que la configuración de un sistema toma como entrada el conjunto X y retorna como salida el routing. Como veremos, tal sistema puede ser no determinista y no siempre está garantizado que la salida sea el routing. Estamos interesados en investigar las condiciones de correctitud que garanticen que la configuración tendrá un comportamiento determinista y termina con el routing.

Aún cuando estas garantías sean encontradas, es posible que el routing resultante tenga un comportamiento de forwarding indeseado, nos referimos a los forwarding loops. Por lo tanto, definiremos condiciones de correctitud adicionales para prevenir estas anomalías.

Una *egress instance* es el par $I = (C,X)$ donde C es una configuración y $X \subseteq B$. Definiremos que una *egress instance* será *signaling correct* si ésta garantiza que se alcance un estado único de routing de forma determinista. Para configuraciones *signaling correct* C, diremos que una *egress instance* es *forwarding correct* si el routing resultante no tiene desviaciones (definiremos esto más formalmente luego). Entonces diremos que una configuración C es forwarding correct si cada egress instance (C,X) es forwarding correct. Una configuración es correcta si es *signaling correct* y *forwarding correct*.

Correctitud de las configuraciones

Ahora presentaremos algunos ejemplos de ambas anomalías de signaling y forwarding en iBGP. Veremos situaciones que pueden ocurrir dentro de un sistema autónomo con iBGP aún en ausencia de problemas debido al atributo MED.

Veamos un ejemplo de anomalías causadas por la ausencia de *signaling correctness*. En la Figura 2.3, podemos ver una *egress instance* que no tiene solución. Utilizaremos las siguientes convenciones en lo que resta de la presentación. Un cliente dentro de un clúster es representado con un círculo, un reflector de rutas es un diamante, y los elementos de un clúster están encerrados en un óvalo punteado. Las líneas sólidas en las figuras son los enlaces físicos y son usualmente etiquetados con su peso IGP. Las líneas punteadas entre dos routers indican que intercambian mensajes iBGP. Los clústers están encerrados en una caja representando el sistema autónomo. Los routers externos (para un origen) son mostrados con flechas entrantes al sistema autónomo.

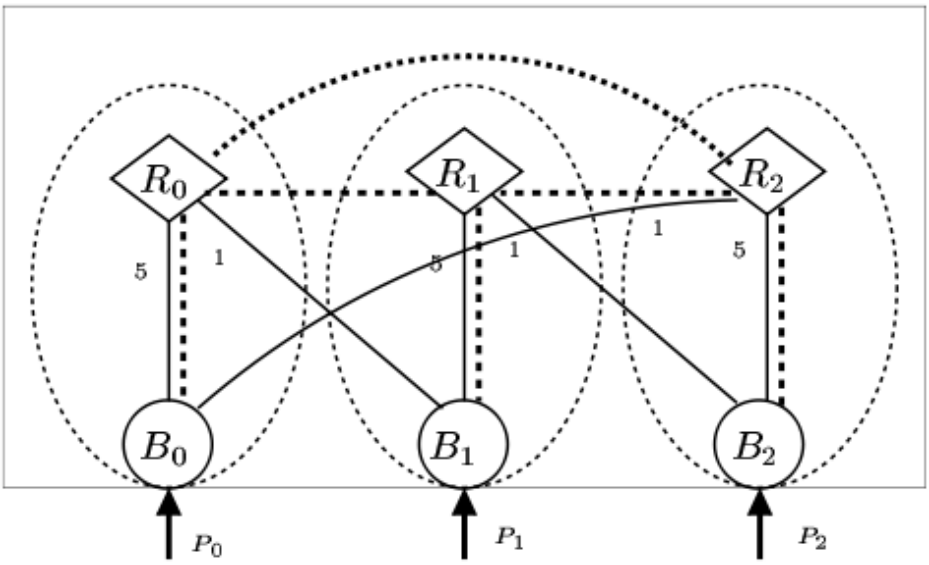


Figura 2.3: Sin solución, no es signaling correct

En la Figura 2.3, cada R_i actúa como reflector con respecto a B_i y cada B_i tienen un punto de egreso inmediato P_i que es aprendido vía eBGP, $i = 1,2,3$. Los pesos marcados en cada arista hacen que la elección del camino de egreso de cada R_i sea tal que R_i prefiere P_{i+1} sobre P_i, cuando los índices son interpretados módulo 3. Es fácil verificar que en esta situación, los routers nunca van a converger a un conjunto de rutas. Por lo tanto, no tiene sentido tan siquiera preguntarse si el sistema evita los forwarding loops debido a que nunca puede establecer un conjunto de rutas fijas.

Realizaremos algunas convenciones gráficas para el resto del capítulo que describimos a continuación. Para evitar el desorden excesivo de las figuras, las sesiones iBGP (las líneas punteadas) no serán mostradas pero estarán implícitas por las estructuras de los clústers. Esto es, asumiremos que cada cliente tiene una sesión BGP entre él y cada reflector de rutas dentro del clúster y todos los reflectores del mismo nivel de la jerarquía están completamente mallados, es decir mantienen sesiones con una configuración

full mesh entre ellos. Utilizando estas convenciones, la configuración mostrada en la Figura 2.3 puede ser presentada como podemos ver en la Figura 2.4.

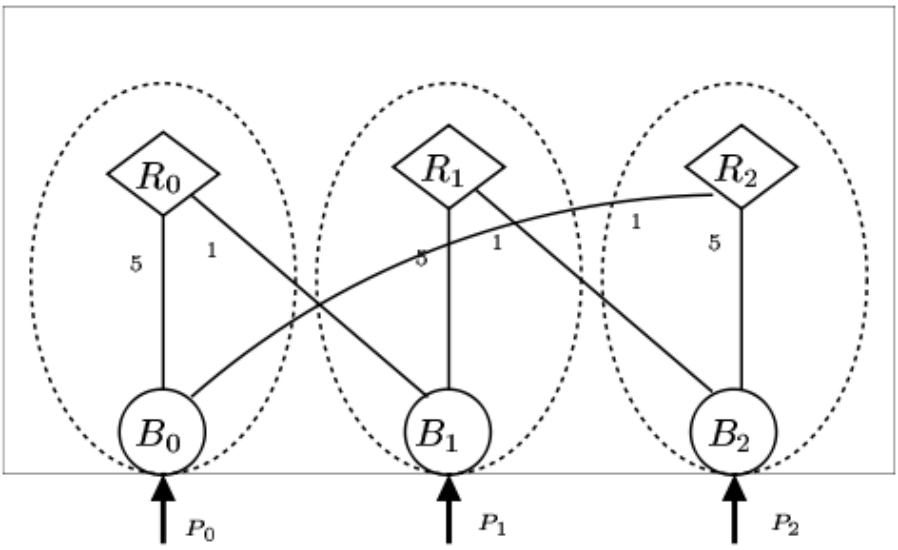


Figura 2.4: sin solución, no es signaling correct. Notación simplificada

Las configuraciones de las Figuras 2.3 y 2.4 no tienen soluciones y resultan en la divergencia de BGP. En contraste, un sistema BGP puede tener múltiples soluciones resultados de un routing no determinista (el protocolo puede elegir una solución basada en el orden aleatorio en el que son procesados los mensajes). La Figura 2.5 presenta una configuración que tiene dos soluciones.

El router R_i actúa como reflector con respecto a C_i donde C_i tiene un camino de egreso P_i aprendido por sesiones eBGP, $i = 1,2$. Hay dos soluciones posibles. El tiempo en el que los mensajes de actualización de rutas llegan a los routers R_1 y R_2 determina cual de las dos soluciones va a ser utilizada. En esta situación cada C_i elegirá su camino de egreso inmediato P_i. Si R_1 informa a R_2 que ha elegido P_1 primero, entonces R_2 también elegirá P_1. De forma similar, si R_2 informa a R_1 que ha elegido P_2 entonces P_2 será el camino de egreso preferido.

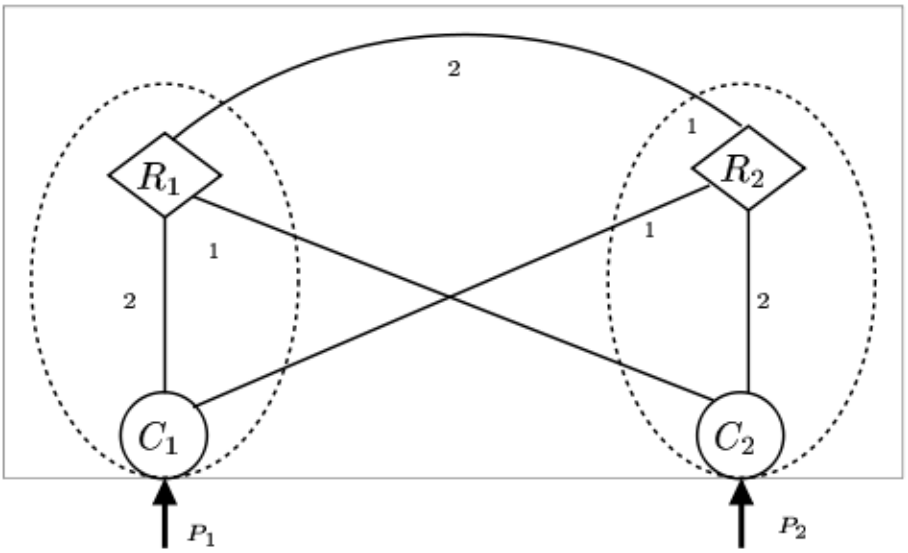


Figura 2.5: Dos soluciones

Ahora continuaremos con la presentación de ejemplos de configuraciones que sufren problemas causados por la ausencia de forwarding correctness. Definiremos intuitivamente la noción de la existencia de desviaciones en un camino dado. Diremos que el camino $P = p_1 p_2 \dots p_t$ en G_p, utilizado por $u \in V$ para enviar paquetes a su punto de egreso $e \in V$, seleccionado por iBGP en u, posee desviaciones si para algún $p_i \in P$ su punto de egreso $e' \in X$ seleccionado por iBGP en p_i es tal que $e \neq e'$. Diremos que una configuración es *forwarding correct* si las rutas resultado en cada nodo no crean desviaciones.

Considere un nodo $u \in V$ y el camino $P = p_1 p_2 \dots p_t$ en G_p que utiliza para enviar paquetes a su punto de egreso $e \in V$. Sea $p_i \in P$ el primer nodo donde existe una desviación. Ahora consideremos el camino $Q = q_1 q_2 \dots q_z$ el utilizado por p_i para enviar paquetes hacia su punto de egreso $e' \in X$. Sea q_j el primer nodo donde existe una desviación. Ahora consideremos el camino $R = r_1 r_2 \dots r_k$ el utilizado por q_j para enviar paquetes hacia su punto de egreso $e'' \in X$. Continuando así, si tal camino tiene bucles diremos que el camino asignado a u por iBGP tiene *forwarding loops*. Suponga que existe una desviación a lo largo de $p = p_1 p_2 \dots p_t$, donde p es el camino utilizado por $u \in V$ para enviar paquetes a su punto de egreso $e \in X$ (para un destino dado). Si esta desviación se da en p_i, y p_i es un punto de egreso para el mismo destino, entonces diremos que se trata de una *desviación simple*. Note que en este caso p_i debe estar en X. La idea es que los paquetes enviados por u no continúan hacia su punto de egreso e, pero en su lugar son desviados fuera del sistema autónomo por p_i. Estas desviaciones simples no pueden causar bucles dentro del sistema autónomo. Obviamente pueden existir desviaciones más complejas que causen problemas.

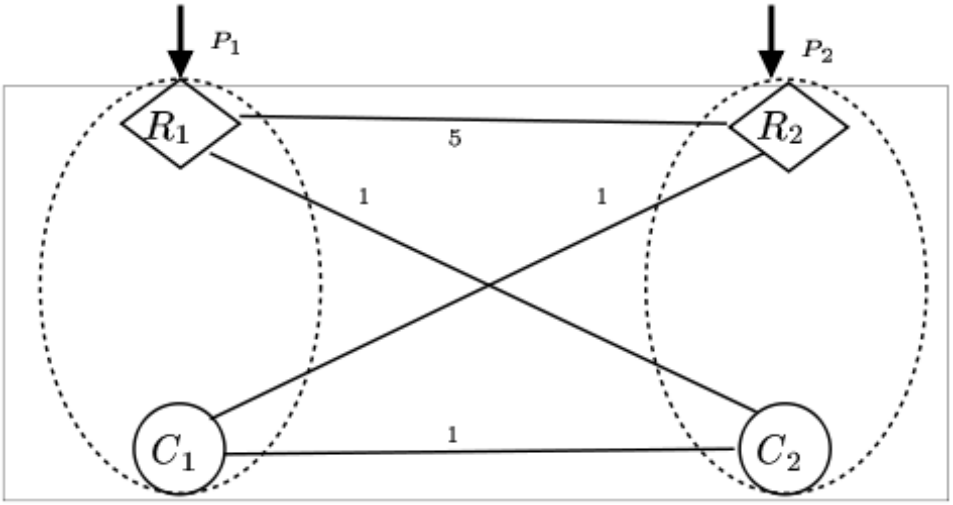


Figura 2.6: Ejemplo de forwarding loops

Un ejemplo de *forwarding loops* puede verse en la Figura 2.6. El sistema tiene dos clústers cada uno con un router R_i actuando como reflector para el router cliente C_i. Así los mensajes de actualización son enviados entre R_i y C_i, $i = 1,2$. La otra sesión iBGP ocurre entre R_1 y R_2. Cada R_i es además un router de borde anunciando un camino de egreso P_i dentro de la red (para un mismo destino dado). Claramente R_i elige P_i y anuncia esta ruta a C_i. Entonces C_i sólo conoce la ruta P_i y también la elige. Pero nótese que el camino más corto para C_1 a P_1 es $C_1C_2R_1$ y el camino más corto para C_2 a P_2 es $C_2C_1R_2$ y entonces podemos observar un *forwarding loop* entre C_1 y C_2. Cuando C_1 envíe paquetes hacia el camino P_1 dirigirá los paquetes hacia C_2. Por su parte, el router C_2 utilizará su camino para el destino, que es P_2. Para enviar paquetes hacia P_2 debe enviarlos hacia C_1, de esta manera generándose el *forwarding loop*.

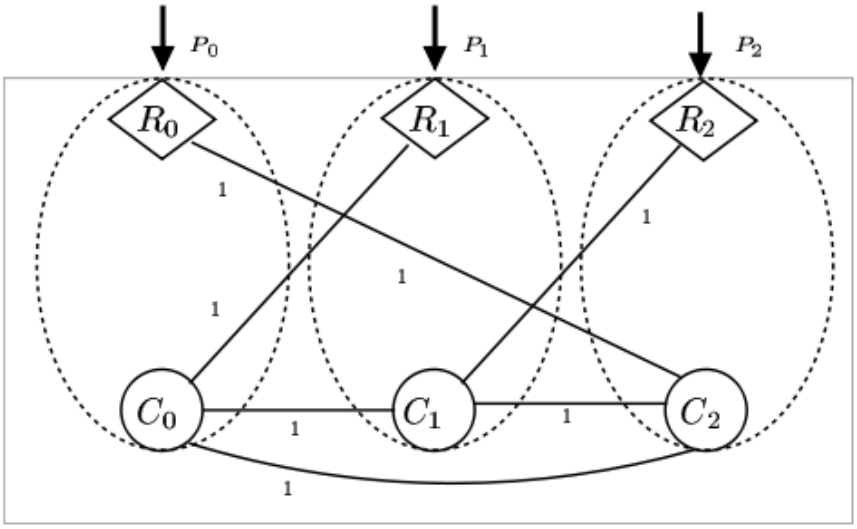


Figura 2.7: Forwarding loop causado por tres caminos de forwarding

Pueden existir *forwarding loops* más complicados que consistan de cualquier número de aristas e involucren cualquier número de caminos de forwarding. En la Figura 2.7 podemos ver un ejemplo donde tres caminos de forwarding juntos forman un *forwarding loop* y la generalización de este ejemplo a cualquier número de caminos de forwarding es fácil de hacer.

En este ejemplo cada cliente C_i prefiere a R_i como punto de egreso (debido a que es la única ruta que su reflector de rutas R_i le anuncia). Todos los enlaces físicos tienen costo 1 y entonces el camino más corto de C_i a R_i es $C_iC_{i-1}R_i$ (donde los subíndices son evaluados módulo 3). Así, se genera el *forwarding loop* $C_oC_1C_2C_0$. Debemos resaltar que cuando la unión de los caminos de forwarding contiene un ciclo, esto no necesariamente implica la existencia de *forwarding loops*. Por ejemplo considere la Figura 2.8.

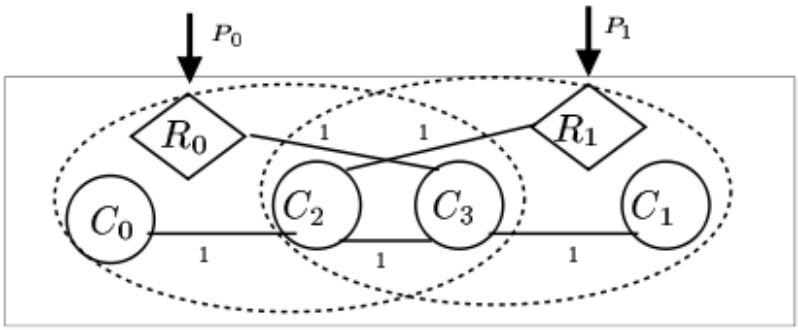


Figura 2.8: Unión de caminos de forwarding que forman un ciclo pero no existen forwarding loops

Es simple verificar que en la solución de este sistema el camino de forwarding para C_1 es $C_1C_3C_2R_1$ y para C_0 es $C_0C_2C_3R_0$. Así la unión de estos caminos de forwarding forma un ciclo $C_2C_3C_2$. Sin embargo la solución del camino de forwarding para C_2 es C_2R_1 y el camino de forwarding para C_3 es C_3R_0. Por lo tanto, no existen forwarding loops debido a que el camino de forwarding de C_0 y C_1 se desvía por C_2 y C_3 respectivamente.

Las anomalías de signaling y forwarding pueden estar ambas presentes en la misma configuración. Considere la configuración mostrada en la Figura 2.9.

Nótese que el subsistema formado por R_2, R_3, C_3 y C_4 es la misma configuración mostrada en la Figura 2.5 y entonces tiene dos posibles soluciones. También, el subsistema formado por R_1, R_2, C_1 y C_2 es esencialmente la configuración mostrada en la Figura 2.6, que vimos que tenía forwarding loops. Obsérvese que para cualquier camino de egreso que elija R_i, el router C_i debe elegir el mismo ya que solo aprende el camino de egreso vía R_i, para i=1,2.

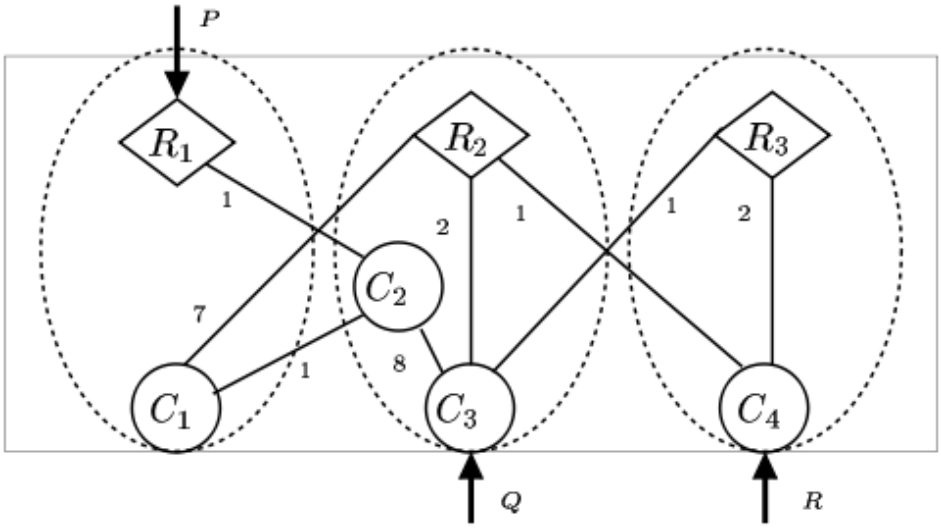


Figura 2.9: Forwarding loop dependiente del orden de llegada de los mensajes

El router R_i siempre elegirá P por lo que C_1 también elegirá P, el camino de forwarding de C_1 siempre será $C_1C_2R_1$. Además, R_2 y R_3 nunca elegirán a P porque Q y R siempre están disponibles para ellos a mejor costo. El camino de forwarding para R_2 y R_3 será el enlace que mantienen con C_3 o C_4. Así el único camino de forwarding que puede causar forwarding loops es el utilizado por C_2. Considere la solución donde R_2 y R_3 seleccionan el camino de egreso Q. Entonces el camino de forwarding utilizado por C_2 es C_2C_3 y no se formarán forwarding loops. Sin embargo, suponga que R_2 y R_3 ambos eligen a R. Entonces el camino de forwarding de C_2 a R es $C_2C_1R_2C_4$ con largo 9 (se prefiere sobre el camino $C_2C_3R_2C_4$ con largo 11) y esto causa el forwarding loop entre C_1 y C_2. Esto muestra que un routing no determinista torna imposible determinar estáticamente si la configuración tiene desviaciones.

Resultados referentes a la signaling correctness: condiciones de suficiencia

Estudiaremos ahora más profundamente la singnaling correctness y mostraremos algunos de los resultados más importantes existentes en la literatura actual. Los ejemplos anteriores muestran que solo tiene sentido preguntar si una configuración es forwarding correct si de hecho, la configuración establece de forma determinista una única instancia de routing. No definiremos aquí de forma más precisa que significa que una configuración establezca de forma determinista una única instancia de routing. La definición formal involucra un problema adicional, el SPP (The Stable Paths Problem). Es posible establecer una transformación en tiempo polinomial de nuestro problema al SPP. En base a esto se define que una configuración es signaling correct si el problema SPP asociado tiene una única configuración. No diremos más al respecto, se puede consultar [3] por más detalles del SPP.

Consideremos ahora la complejidad del problema que consiste en determinar si una configuración es signaling correct. Desafortunadamente se demuestra que este problema es NP-hard. La demostración de que este problema es NP-hard se reduce a encontrar una transformación, en tiempo polinomial, de nuestro problema al 3-SAT, un problema del cual se conoce que es NP-complete. Por detalles respecto a 3-SAT consulte [4].

Hemos mencionado que es difícil determinar si una configuración es signaling correct. Sin embargo, podemos mostrar condiciones suficientes que garantizan que una configuración es signaling correct. Estas condiciones garantizan la signaling correctness mostrando que no pueden existir dispute wheels (una estructura derivada del modelo formal de SPP, esta representa una secuencia circular de routers con configuraciones incompatibles). El resultado utiliza el hecho de que la ausencia de dispute wheels implica la signaling correctness.

Presentaremos ahora las políticas valley free utilizadas en el intercambio de información de routing entre sistemas autónomos. Los modelos de customer-provider y peer-peer dicen que para cada sistema autónomo u, y cada sistema autónomo v con el cual u tiene una sesión eBGP, v está exactamente en uno de los tres conjuntos $customer(u)$, $provider(u)$ o $peer(u)$, dependiendo de la relación establecida entre u y v. Estos conjuntos son tales que si $v \in customer(u)$ entonces $u \in provider(u)$ y si $v \in peer(u)$ entonces $u \in peer(v)$. Entonces las reglas de exportación de rutas están determinadas por estas relaciones como sigue:

1. u exporta sólo sus rutas y aquellas que pertenezcan a sus clientes a $\forall\, v \in provider(u)$.
2. u exporta todas sus rutas a $\forall\, v \in customer(u)$.
3. u exporta sólo sus rutas y aquellas que pertenezcan a sus clientes a $\forall\, v \in peer(u)$.

Definimos el grafo dirigido D con un nodo por cada sistema autónomo y un arco (u, v) si y solo si $v \in customer(u)$. Entonces puede verse que si D es un grafo dirigido acíclico y cada sistema autónomo prefiere rutas de sus customers sobre rutas de sus peers y sus providers entonces podemos garantizar que el sistema converge a un estado de routing estable.

Es fácil ver que decir que tenemos un arco $(u, v) \in UP$ es equivalente a tener $v \in provider(u)$, tener un arco $(u, v) \in DOWN$ es equivalente a $v \in customer(u)$ y tener un arco $(u, v) \in OVER$ es equivalente a tener $v \in peer(u)$. Esta condición de suficiencia es análoga a las políticas valley free existentes entre los sistemas autónomos.

Por lo tanto, se tiene una condición de suficiencia que garantiza que una configuración sea signaling correct: si el grafo dirigido formado por todos los arcos $(u, v) \in DOWN$ no tiene ciclos y todos los nodos u prefieren rutas de egreso escuchadas desde sus clientes sobre rutas que aprenden de no clientes, entonces la configuración será signaling correct.

Resultados referentes a la forwarding correctness: condiciones de suficiencia

Ahora consideraremos el problema de determinar si una configuración que sabemos que es signaling correct, es de hecho forwarding correct. Supondremos una configuración signaling correct dada. Nos gustaría saber si para todo posible conjunto de caminos de egreso resulta una configuración libre de desviaciones. Desafortunadamente, este problema resulta ser NP-hard.

Dada una configuración signaling correct, nos gustaría ser capaces de determinar si ésta es de hecho una configuración forwarding correct, por lo tanto correcta (más adelante veremos que existen topologías signaling correct y forwarding correct que aún presentan problemas graves).

Ya que este problema es NP-hard, tal vez sea posible encontrar condiciones suficientes que garanticen la correctitud. Antes de hacer esto veremos algunos ejemplos de topologías que son signaling correct pero sin embargo tienen simples desviaciones o forwarding loops.

El sistema en la Figura 2.10 muestra una configuración con un router R_1 actuando como reflector de ruta para dos clientes C_1 y C_2.

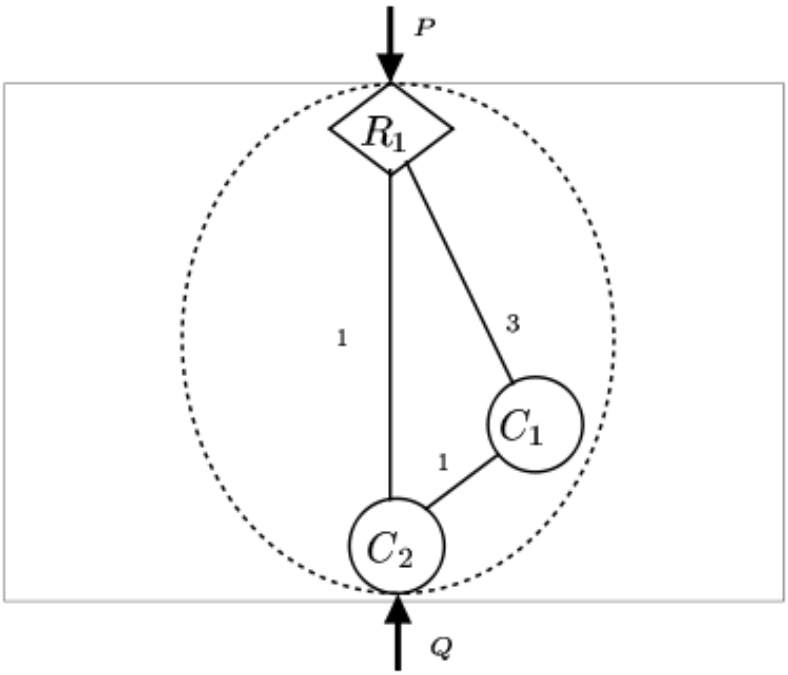


Figura 2.10: Desviación simple en un sistema trivial

Solo existen sesiones iBGP entre R_1 y C_i, con $i = 1,2$. El resultado es que R_1 siempre elige a P (una ruta aprendida por eBGP), y así C_1 solo aprende P, y por eso la elegirá. Pero C_2 elige Q, y C_2 forma parte del forwarding path de C_1 para P. Este ejemplo muestra desviaciones simples que no pueden causar forwarding loops pero hacen que los paquetes enviados por C_1 "tomen un atajo" por un camino de egreso inesperado. Este ejemplo puede ocurrir en

configuraciones muy sencillas a simple vista. Tenga en cuenta que en este ejemplo solo existen sesiones iBGP típicas entre los clientes y su reflectores de rutas y el grafo físico es full mesh.

La Figura 2.11 es un ejemplo bastante simple que muestra que los forwarding loops pueden ocurrir en configuraciones que permiten más sesiones iBGP de las que normalmente se definen.

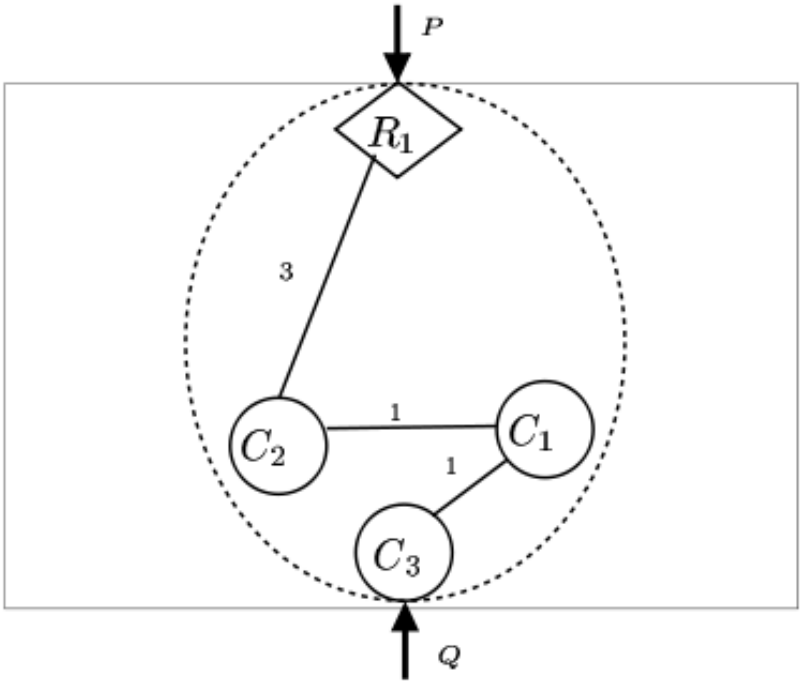


Figura 2.11: Sistema trivial con forwarding loops

Las sesiones iBGP son típicamente una entre el reflector de rutas R_1 y cada cliente C_i. En este ejemplo hay una sesión iBGP adicional entre C_3 y C_2. La topología física es la mostrada en la figura. Esto resulta en que C_1 elige P porque nunca escucha a Q ya que R_1 elige P. Pero debido a la sesión iBGP extra entre C_2 y C_3, C_2 si escucha a Q y elige esta. Así se establece un forwarding loop entre C_1 y C_2.

De hecho, la presencia de desviaciones simples puede en realidad causar forwarding loops entre sistemas autónomos. Un ejemplo de ello es mostrado en la Figura 2.12.

En esta red hay dos sistemas autónomos llamados $AS1$ y $AS2$ intentando establecer rutas para un origen dado. Solo los enlaces físicos son mostrados en la figura etiquetados con sus costos. Ambos sistemas autónomos, $AS1$ y $AS2$ no utilizan el chequeo del largo del as-path (o son sistemas autónomos miembros

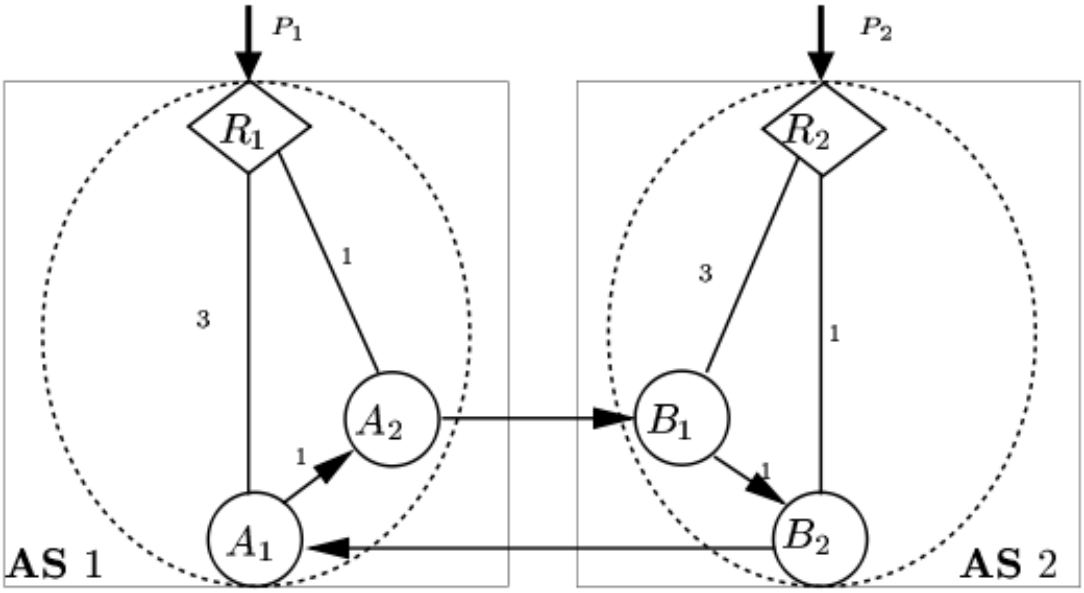


Figura 2.12: Forwarding loops externos causados por desviaciones internas

de una confederación). El router R_1 actúa como reflector para los clientes A_1 y A_2 y estas sesiones iBGP entre R_1 y estos clientes son las únicas sesiones iBGP dentro del sistema autónomo $AS1$. Similarmente, R_2 es un reflector de rutas para los clientes B_1 y B_2 y no hay sesiones iBGP entre los clientes B_1 y B_2. Imaginemos que R_2 aprende la ruta P_2 y la anuncia a sus clientes. Entonces B_1 selecciona esta ruta y la anuncia a A_2 vía eBGP. Entonces A_2 seleccionará la ruta correspondiente al camino $A_2B_1R_2P_2$. Mientras tanto, R_1 aprende la ruta P_1 y la anuncia hacia sus clientes. Esto resulta en que A_1 selecciona a P_1, ya que es la única ruta que aprendió mientras que A_2 selecciona la ruta $A_2B_1R_2P_2$ ya que fue aprendida por eBGP. Sin embargo, el camino más corto para A_1 hacia R_1 pasa por A_2 y esto resulta en una desviación. Ahora A_1 anuncia vía eBGP la ruta correspondiente al camino $A_1R_1P_1$ y B_2, seleccionará esta ya que es la única que aprendió. Esto causa que los paquetes enviados por B_1 a R_2 sean desviados en B_2 ya que este está sobre el camino más corto de B_1 a R_2. Así existe un forwarding loop $B_2A_1A_2B_1B_2$ (está indicado en la figura por enlaces dirigidos).

Cuando el forwarding path no es un signaling path válido, es posible obtener forwarding loops. En la Figura 2.13, el router R_0 es un reflector de rutas para el R_2, donde R_2 a su vez actúa como reflector de rutas, junto con el segundo reflector R_1 en un clúster conteniendo al cliente C_1.

El camino $C_1R_2R_1$ es el camino más corto entre C_1 y R_1 y es además un signaling path válido desde C_1 a R_1. Pero R_2 prefiere a R_0 como nodo de egreso sobre C_1 en virtud de los costos establecidos en el grafo IGP, por esto no satisface la restricción que establece que deben preferirse rutas de los clientes sobre cualquier otra ruta. Como $R_0R_1R_2$ no es un signaling path válido R_2 elige a C_1 y causa una desviación en R_1.

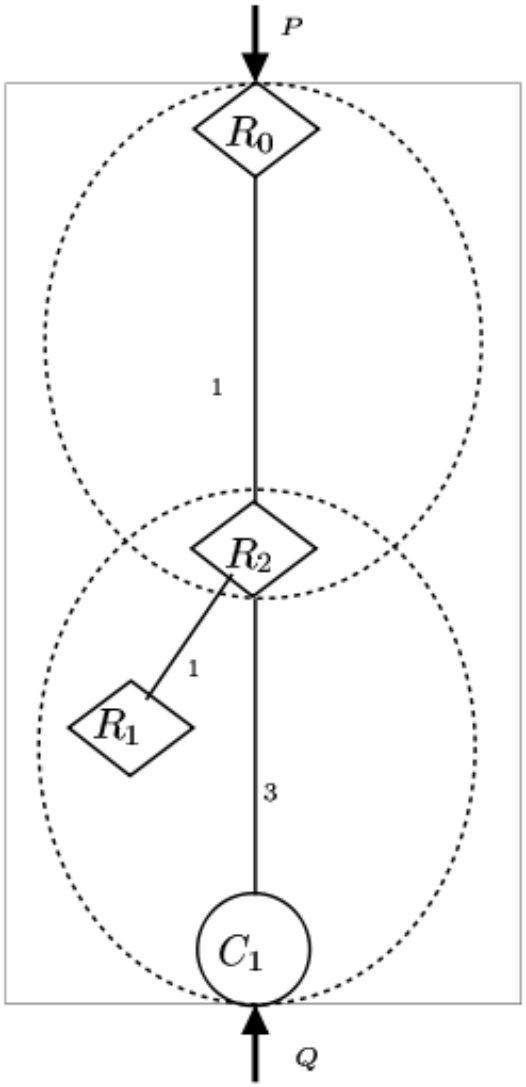


Figura 2.13: Favoreciendo una ruta que no es de cliente

Dado que determinar si las desviaciones pueden ocurrir o no es un problema NP-hard queremos encontrar condiciones suficientes que garanticen que las desviaciones (y por tanto los forwarding loops) no pueden ocurrir. Así si tenemos una configuración signaling correct que cumple con estas condiciones suficientes, será de hecho forwarding correct.

Denotamos como $\lambda^u(v) < \lambda^u(w)$ para los nodos $v \neq w$ el hecho de que el camino más corto de u a v es mayor al camino más corto de u a w, y en caso de ser iguales el $oid(v) > oid(w)$.

Las condiciones suficientes son:

1. Si $(v,u) \in UP$ y $(w,u) \notin DOWN$ debe ser que $\lambda^u(uwP^rO) < \lambda^u(uwQ^rO)$ para algún signaling path válido uwP y uwQ.

2. Para algún par de nodos u y v, $sp(u,v) = P$ para algún signaling path válido P.

La primera condición es motivada por el ejemplo de la Figura 2.13. La segunda condición está motivada por los ejemplos de las Figuras 2.10 y 2.11.

2.6 Algo más que correctitud

Lo que hemos visto hasta ahora es el resultado de un significante esfuerzo en la última década de la comunidad de investigación. Hemos visto que se ha formalizado la ausencia de anomalías de routing y forwarding introduciendo dos propiedades fundamentales de las configuraciones iBGP, que llamamos signaling y forwarding correctness. Como ya hemos mencionado, la signaling correctness asegura que BGP siempre converge a un estado de routing estable, mientras que la forwarding correctness garantiza la ausencia desviaciones de paquetes a lo largo del forwarding path. Luego, vimos que los autores proponen soluciones que garantizan la forwarding correctness y la signaling correctness, ambas requiriendo propiedades especiales a las configuraciones iBGP, como hemos visto, o proponiendo modificaciones propias en el protocolo. Esta subsección se basa en el trabajo presentado en [19].

Ahora veremos que las reglas de propagación de rutas juegan un rol fundamental en asegurar la correctitud de las configuraciones iBGP con reflectores de rutas. Veremos ejemplos simples donde pueden crearse agujeros negros en el tráfico por el efecto combinado de las reglas de propagación de rutas de iBGP y el algoritmo de selección de rutas de iBGP. Aún peor, veremos ejemplos en los que distintos prefijos no siempre pueden ser analizados por separado. Para modelar la ausencia de anomalías debido a las reglas de propagación de rutas de iBGP, definiremos una nueva propiedad, llamada dissemination correctness. Veremos como la dissemination correctness llena el hueco existente entre la signaling correctness y la forwarding correctness. Desafortunadamente, se ha demostrado que verificar la dissemination correctness es computacionalmente intratable. Aún peor, verificar si la adición de una sola sesión iBGP a una configuración afecta la propiedad de dissemination correctness es también computacionalmente intratable. Sin embargo, se han propuesto condiciones suficientes que aseguran la dissemination correctness y utilizando ésta se han definido guías para el diseño de topologías iBGP. En particular se busca la ausencia de un tipo especial de sesiones iBGP (sesiones spurious OVER, luego las definiremos) que garantiza la ausencia de anomalías en la propagación de las rutas BGP. Aunque son poco utilizadas, las sesiones spurious OVER a veces están presentes en las redes de la vida real. En efecto, éstas pueden ser utilizadas por los operadores de redes para resolver problemas de forwarding y proporcionar diversidad de rutas, esto ha sido sugerido en algunos trabajos recientes. Desafortunadamente, gran parte del trabajo realizado anteriormente, asumen incorrectamente que la signaling correctness implica la dissemination correctness. Ya que la presencia de sesiones spurious OVER afectan la generalidad de esta suposición, discutiremos cómo se relacionan los trabajos que ya hemos mencionado con la dissemination correctness.

Resumamos brevemente los conceptos que hemos utilizado hasta ahora. El comportamiento de un router iBGP principalmente consiste de tres fases: primero, recoge la información de routing desde las routers iBGP vecinos;

segundo, selecciona la mejor ruta; tercero, propaga esta mejor ruta seleccionada a sus routers vecinos.

Con reflectores de rutas, los vecinos iBGP de cada router están separados en tres conjuntos: *clients*, *peers* y *route-reflectors*. Cada router iBGP propaga su mejor ruta de acuerdo con las siguientes reglas: si la ruta es aprendida desde un p*eer* o un route-reflector, entonces será transmitida solo a sus clientes; en otro caso la ruta es transmitida a todos los routers iBGP vecinos. Organizando los routers en una jerarquía de clientes y reflectores de rutas permitimos que iBGP escale. Un *clúster* consiste en uno o más route-reflectors y todos sus clientes. Siempre que no se declare explícitamente asumimos que cada clúster tiene un único reflector. Cada clúster es identificado a través de un CLUSTER_ID único. Los mensajes mantienen un atributo (CLUSTER_LIST) que representa el camino iBGP que ha atravesado y se utiliza para evitar bucles en el plano de control.

Nos hemos referido a las sesiones entre un cliente y su reflector como una sesión UP si está establecida desde el cliente hacia el reflector, y como DOWN en otro caso. Además, nos referimos a las sesiones entre peers como sesiones OVER. Hemos llamado a la organización de las sesiones iBGP como la topología iBGP.

La selección de la mejor ruta es realizada según el proceso de decisión de BGP, anteriormente presentado en la sección 2.1.

A través de este capítulo, hemos considerado que las rutas son igualmente preferidas de acuerdo con los primeros cuatro puntos del proceso de decisión de BGP. Hemos denotado a los routers que reciben rutas eBGP para un prefijo dado como el *egress point* para ese prefijo. Cada router iBGP tiene un identificador, el ROUTER_ID. El EGRESS_ID para una ruta es el ROUTER_ID del egress point que ha anunciado la ruta.

Un ejemplo de una red iBGP puede verse en la Figura 2.14(a).

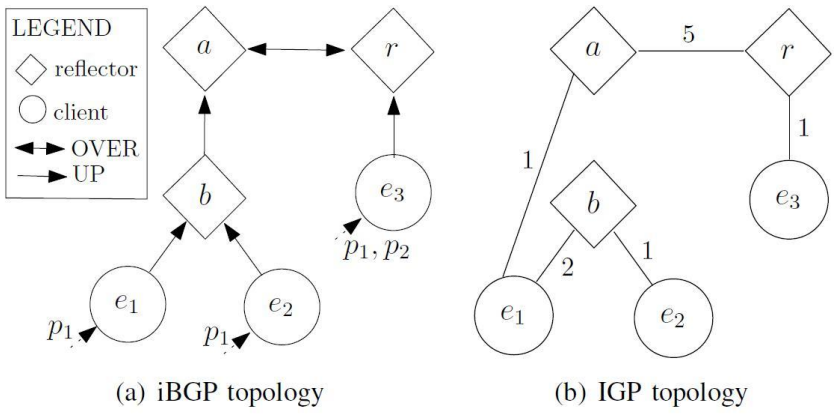


Figura 2.14: Una simple topología que presenta problemas de visibilidad

La convención gráfica que utilizamos en la figura es la misma que hemos utilizado a lo largo de este documento. Los círculos representan clientes iBGP, mientras que los diamantes representan reflectores de rutas. Las sesiones UP y OVER son representadas por flechas simples y dobles respectivamente. Las flechas punteadas etiquetadas con p_1 que ingresan en e_1, e_2 y e_3 representan el hecho de que e_1 y e_2 son egress point para el prefijo p_1. Similarmente e_3 es el egress point para p_1 y p_2. El grafo IGP subyacente está representado en la Figura 2.14(b), las líneas representan enlaces IGP y las etiquetas los pesos IGP asignados a los enlaces.

Consideremos el prefijo p_1. Debido al paso 5 del proceso de decisión de BGP, los routers e_1, e_2 y e_3 elegirán su ruta externa que denotamos por R_1, R_2 y R_3, respectivamente. Por lo tanto, anunciarán su mejor ruta a todos sus vecinos iBGP, es decir b (por e_1 y e_2) y r (por e_3). El router b recoge las rutas de sus clientes, selecciona la mejor ruta y la anuncia a sus vecinos. Por el paso 6 del proceso de decisión de BGP, b selecciona la ruta R_2 porque e_2 es un egress point más cercano que e_1. Por lo tanto, por las reglas de propagación de iBGP b anuncia R_2 a su reflector de rutas a. Cada router continuará acumulando las rutas, seleccionando y propagándolas hasta que el protocolo BGP converja y no se propaguen más mensajes. Luego de la convergencia, el router r seleccionará la ruta R_3 y el router a seleccionará la ruta R_2.

Obsérvese que el router a no tiene conocimiento de la ruta R_1 porque éste solo recibe la ruta R_2 desde b y la ruta R_3 desde r. De hecho, los reflectores de rutas introducen una visibilidad de rutas sub óptimas y limitan la diversidad de rutas disponibles para el router a. Otro efecto que es introducido por la utilización de reflectores de rutas es la desviación de paquetes provocada cuando a envía tráfico hacia el prefijo p_1. Más precisamente, el router a cree que el tráfico debe salir por el egress point e_2 por lo que envía los paquetes hacia e_1 que es el next hop hacia e_2. Sin embargo, e_1 es él mismo un egress point para p_1, por lo que desvía el tráfico fuera del sistema autónomo. Como hemos visto, la combinación de múltiples desviaciones puede causar forwarding loops.

Siempre que existen problemas debido a la visibilidad sub óptima de las rutas, solucionarlo mediante la adición de sesiones iBGP adicionales puede parecer una solución fácil y tentadora para los operadores de la red. En nuestro ejemplo agregar una sesión iBGP entre los router a y e_1 pueden proveer al router a de la diversidad buscada y puede permitir que seleccione su egress point óptimo. La adición de sesiones OVER ha sido propuesta para aumentar la diversidad y para reducir el tiempo de convergencia de iBGP. Estudios cuantitativos han demostrado que la utilización de reflectores de rutas conduce a una diversidad muy pobre. Esto, a su vez, puede causar grandes tiempos de convergencia en caso de fallas o cambios en el routing interdominio. Por otra parte, sesiones adicionales pueden proveer mejor diversidad de rutas a los routers, haciendo así más fácil para un operador de red corregir su configuración iBGP con el fin de cumplir con las directrices del estado del arte.

En general, agregar sesiones iBGP no implica necesariamente el agregar sesiones OVER; esto es, éstas pueden ser sesiones UP. Sin embargo, los operadores prefieren utilizar sesiones OVER para no introducir un costo adicional de memoria y un intercambio de mensajes mayor, ya que sólo un subconjunto de rutas es anunciado a través de sesiones OVER.

Desafortunadamente, agregar sesiones OVER a una topología iBGP puede provocar efectos secundarios indeseables. Consideremos ahora la topología iBGP de la Figura 2.15 que es una versión simplificada de la Figura 2.14.

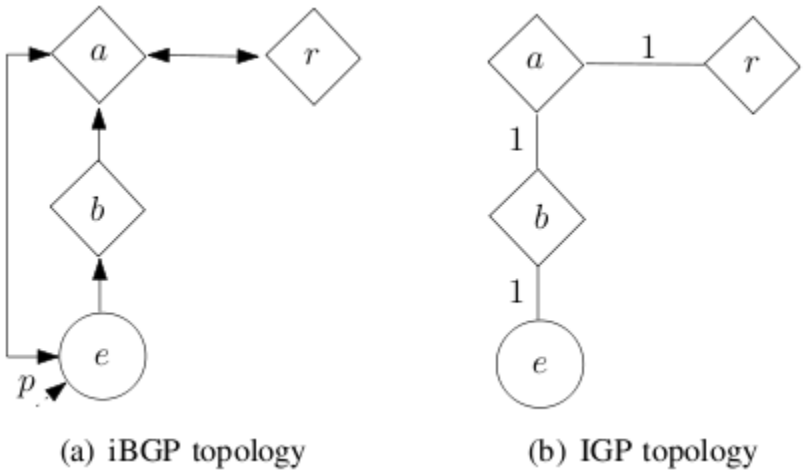


Figura 2.15: Sesiones spurious over

Se ha agregado una sesión OVER entre los routers a y e. Ya que e es el único egress point para p, a preferirá la ruta que aprende por medio de la sesión OVER debido al paso 8 del proceso de selección de rutas de BGP. Entonces, como la ruta es aprendida por medio de un peer, a no la propaga hacia r, así r no tendrá una ruta para el prefijo p. Ahora, si r tiene una ruta para un prefijo menos específico que incluye a p (por ejemplo, una ruta por defecto) éste utilizará ésta ruta para el tráfico dirigido hacia p posiblemente causando desviaciones en el forwarding. Por otra parte, si r no tiene una ruta para un prefijo menos específico que p, r crea un agujero negro en el tráfico. Observe que ambos tipos de anomalías son debidos únicamente a la topología iBGP. La topología IGP es irrelevante en este caso porque sólo hay un egress point para p.

Las topologías iBGP están usualmente organizadas en jerarquías donde no hay ciclos consistentes únicamente de sesiones UP. En efecto, tales ciclos son un signo de un mal diseño de topología y pueden causar anomalías. En una jerarquía, cada router BGP puede ser asignado a una capa. Denotamos el conjunto de routers que pertenecen a la capa superior de un topologías iBGP B como T_B. Un router pertenece a la capa superior T_B si no es cliente de ningún reflector de rutas.

Ahora introduciremos el concepto de sesiones spurious OVER. Además veremos cómo sus efectos secundarios pueden invalidar simples supuestos sobre las topologías iBGP, y que son utilizados en los trabajos que presentan las ideas que hemos visto hasta el momento.

Dada una topología iBGP B, una sesión OVER entre los routers x e y es spurious si uno de los dos routers no está en la capa superior, esto es, si $x \notin T_B$ o $y \notin T_B$.

Las sesiones spurious OVER no son frecuentes en las redes de los ISP hoy en día. Las directrices de los proveedores sugieren no utilizarlas. Sin embargo, las sesiones spurious OVER han sido propuestas para solucionar problemas de visibilidad, y algunos trabajos previos muestran que los grandes ISP a veces las utilizan. Por otra parte, las sesiones spurious OVER pueden ser introducidas involuntariamente en la topología iBGP. Por ejemplo, las buenas prácticas utilizadas actualmente para remplazar configuraciones iBGP de full mesh con reflectores de rutas sugieren agregar progresivamente sesiones UP antes de remover la configuración full mesh. Por lo tanto, las sesiones OVER inicialmente presentes en el full mesh probablemente se conviertan en sesiones spurious OVER en configuraciones intermedias.

Como ya hemos discutido, la Figura 2.15 provee un ejemplo de cómo sesiones spurious OVER mejoran la visibilidad de los egress point en algunos routers, pero potencialmente empeora la visibilidad de otros. En este ejemplo, el efecto secundario provocado por la sesión spurious OVER es poco intuitivo porque introduce cambios en el proceso de diseminación de rutas en el router r sin afectar el egress point seleccionado por él. Esto contradice la intuición que utilizamos para decir que una topología iBGP conexa garantiza que cada router eventualmente aprenda al menos una ruta para cada prefijo.

Desafortunadamente, algunos de los trabajos previos están basados en esta intuición. En particular, algunos trabajos asumen que agregando sesiones OVER solo pueden mejorar la visibilidad de rutas, mientras que otros trabajos asumen que un reflector de rutas r puede ocultar una ruta a un router vecino v solo si tiene un egress point alternativo más cercano.

Con más generalidad, sesiones spurious OVER muestran que el concepto del signaling path válido no es una buena abstracción de la habilidad actual de los routers para aprender una ruta a un prefijo dado. En busca de un mejor entendimiento de esta propiedad, ahora introduciremos el concepto de dissemination correctness.

Sea B una topología iBGP singaling correct. Entonces B es dissemination correct si todos los routers en B están garantizados de recibir al menos una ruta para el prefijo p en un estado estable, para cualquier conjunto no vacío de egress point para p.

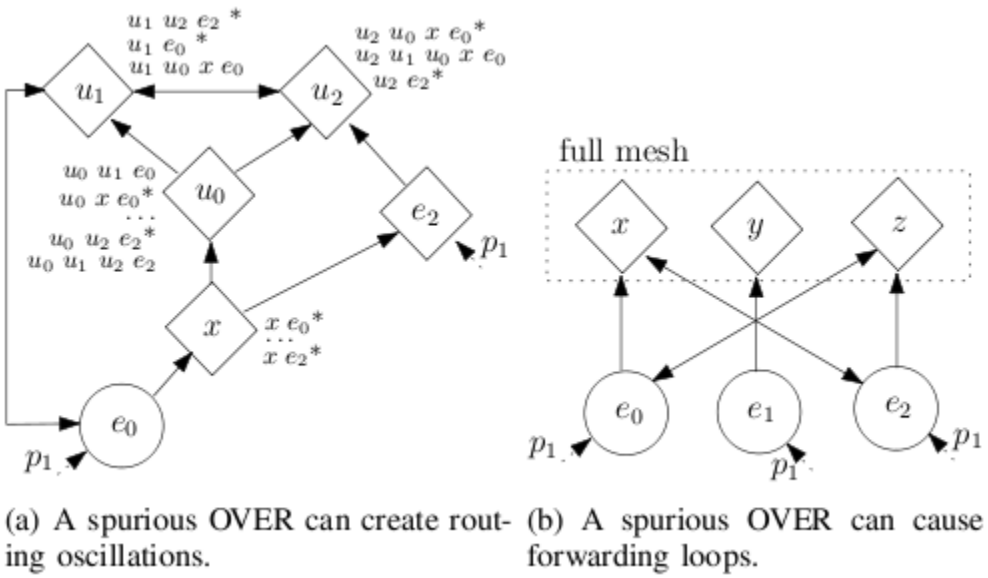
Nótese que la dissemination correctness no depende del routing interdominio ni del conjunto de egress point que actualmente aprenden rutas para prefijos dados. Es decir, esto es una propiedad de la topología. La dissemination correctness difiere tanto de la signaling correctness como de la forwarding correctness. En efecto, una topología signaling correct no está garantizado que sea dissemination correct. Por otra parte, topologías dissemination correct no garantizan ser forwarding correct. Las tres propiedades en realidad se complementan: la signaling correctness se ocupa de las anomalía en el routing que provoquen que BGP no converga; la dissemination correctness se ocupa de los problemas en el proceso de distribución de rutas; y la forwarding correctness se ocupa de las anomalías de forwarding causadas debido a la interacción entre iBGP y IGP.

Además de afectar la dissemination correctness, las sesiones spurious OVER pueden impedir que una topología iBGP sea signaling o forwarding correct, como podemos ver en la Figura 2.16(a).

Cada router mantiene una lista de los signaling path válidos, ordenados en orden decreciente de preferencia. Nótese que (u_1, e_0) es una sesión spurious OVER. Veremos que iBGP no podrá converger en estas condiciones. Asumiremos por absurdo, que un estado estable existe y consideraremos las elecciones tomadas por el router u_2. Ya que u_2 recibe una ruta directamente desde e_2 no es posible que no seleccione alguna ruta para el prefijo para p_1.

Por lo tanto tenemos los siguientes casos:

- u_2 selecciona (u_2, e_2). En este caso, u_1 utilizará como su camino preferido a (u_1, u_2, e_2), impidiendo que u_0 seleccione (u_0, u_1, e_0). Así, u_0 seleccionará (u_0, x, e_0) y eventualmente anunciará este camino a u_2. Debido a la preferencia de los caminos, u_2 deberá cambiar su camino a (u_2, u_0, x, e_0), dándose una contradicción ya que hemos partido de la hipótesis de que u_2 selecciona (u_2, e_2).
- u_2 selecciona (u_2, u_1, u_0, x, e_0). Esto implica que u_1 seleccione (u_1, u_0, x, e_0), llevándonos a una contradicción, ya que el camino (u_1, e_0) está siempre disponible para u_1 y es preferido sobre (u_1, u_0, x, e_0).
- u_2 selecciona (u_2, u_0, x, e_0). Esto implica que u_0 seleccione (u_0, x, e_0), y u_1 se ve forzado a seleccionar (u_1, e_0), ya que no recibe el camino (u_2, e_2) desde u_2. Esto nos lleva a una contradicción: u_0 eventualmente aprenderá y seleccionará (u_0, u_1, e_0), impidiendo que u_2 seleccione (u_2, u_0, x, e_0) ya que u_0 no anunciará (u_0, x, e_0).



(a) A spurious OVER can create routing oscillations.

(b) A spurious OVER can cause forwarding loops.

Figura 2.16: Dos casos en los cuales agregar sesiones OVER crean anomalías de singaling y forwarding.

Todos los casos nos llevan a un absurdo, por lo tanto no existe un estado estable en la topología de la Figura 2.16(a). Veamos ahora que topología y políticas pueden llevarnos a una configuración como la mostrada en la figura. Las preferencias de caminos mostrados en la figura pueden resultar del proceso estándar de selección de rutas de BGP si la topología IGP es tal que $dist(x, e_0) < dist(x, e_2)$, $dist(u_0, e_0) < dist(u_0, e_2)$, $dist(u_2, e_0) < dist(u_2, e_2)$ y $dist(u_1, e_0) < dist(u_1, e_2)$. En este caso x, u_0 y u_2 prefieren caminos basados en el egress point más cercano, mientras que u_1 prefiere rutas eBGP recibidas desde e_2 sobre aquellas recibidas desde e_0 por su EGRESS_ID. Los empates son rotos mediante el largo de la CLUSTER_LIST y el criterio de tomar aquella ruta proveniente del vecino con la menor dirección.

La forwarding correctness también puede verse afectada por la presencia de sesiones spurious OVER. Consideremos ahora la topología de la Figura 2.16(b), y asumamos que x selecciona el camino (x, e_2), mientras que z selecciona (z, e_0), por las distancias del IGP. Como estos caminos son aprendidos por medio de sesiones OVER, x y z no van a propagar su mejor ruta a y, por lo tanto y está forzado a seleccionar la ruta desde e_1. Si y está en el camino más corto de x a e_2 y si x está en el camino más corto de y hacia e_1, entonces un bucle surge para p_1.

Condiciones de suficiencia para la dissemination correctness

Ahora que conocemos el problema, veremos las condiciones suficientes propuestas y discutiremos su aplicabilidad. Es posible demostrar que las condiciones suficientes que hemos propuesto anteriormente para la correctitud de una topología también aseguran la dissemination correctness. Desafortunadamente, se ha encontrado que estas condiciones son difíciles de asegurar, especialmente con el diseño de las buenas prácticas propuestas actualmente. Por esto, mostraremos condiciones de suficiencia más simples.

Cualquiera de las siguientes condiciones aseguran que una topología iBGP B signaling correct será además dissemination correct.

1. *prefer-client*: todos los routers iBGP en B prefieren rutas propagadas por sus clientes (en un camino UP*) que cualquier otra ruta.
2. *no-spurious-OVER*: B no contiene sesiones spurious OVER.

Ahora discutiremos como pueden ser aplicadas estas condiciones de suficiencia en topologías iBGP de la vida real. En teoría, la condición de *prefer-client* puede ser asegurada con un diseño cuidadoso de la topología iBGP. Sin embargo, esta condición es algo restrictiva en topologías de la vida real. De hecho, para satisfacer la condición de *prefer-client* cada router debe priorizar las rutas recibidas de acuerdo con el primer next hop en el camino de señalización iBGP, mientras que el proceso de selección de BGP usa el criterio de desempate basados en el último hop en el camino de señalización (es decir, el EGRESS_ID) o el largo del propio camino (es decir, el CLUSTER_LIST). En particular, una consecuencia directa de la restricción de *prefer-client* es que, si el router r tiene un camino de señalización válido $P = (r, s, \dots, e)$ con r cliente de s (posiblemente $s = e$) entonces cualquier camino de señalización válido entre r y e debe tener un cliente de r como next hop o debe ser más largo que P. Por lo tanto, satisfacer la condición de *prefer-client* requiere de una profunda evaluación de todos los pasos del proceso de selección de iBGP. Por esta razón, se convierte en una tarea muy difícil cuando se despliega una topología con reflectores de rutas redundantes, aún en topologías muy simples. Consideremos por ejemplo, la configuración de la Figura 2.17 que consiste de una simple topología con reflectores de rutas redundantes diseñada de acuerdo con las prácticas recomendadas actualmente.

Los clientes e_1 y e_2 están conectados a ambos reflectores de rutas r_1 y r_2 (r_1 y r_2 pertenecen a diferentes clústers). Tanto e_1 como e_2 son egress points para el prefijo p_1. Aún en este escenario tan simple, la condición de *prefer-client* no se cumple, cualquiera sea la topología IGP: los caminos subrayados violan la condición de *prefer-client*.

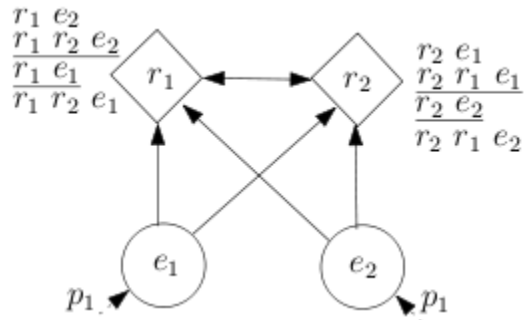


Figura 2.17: Topología redundante donde es difícil satisfacer la condición prefer-client

En efecto, considere el router r_1, y asumiendo que e_2 es el egress point más cercano de acuerdo con las métricas IGP. En este caso r_1 prefiere todas las rutas recibidas por e_2 sobre todas las rutas recibidas desde e_1, debido al paso 6 del proceso de selección de BGP.

Por lo tanto, r_1 prefiere rutas aprendidas por el camino (r_1, r_2, e_2) sobre aquellas aprendidas por (r_1, e_1). Esto viola la condición de *prefer-client*. Una violación similar ocurre si $dist(r_1, e_1) < dist(r_1, e_2)$. Este tipo de violaciones a la condición de *prefer-client* pueden ser solucionadas por un diseño más inteligente de los clústers de reflectores de rutas. En efecto, si r_1 y r_2 pertenecen al mismo clúster, entonces r_1 siempre descarta las rutas propagadas por r_2 y viceversa.

En configuraciones iBGP redundantes, los reflectores de rutas redundantes deben pertenecer al mismo clúster. Las buenas prácticas de diseños de topologías iBGP de la actualidad no contemplan esta restricción.

La condición de *no-spurious-OVER* es relativamente fácil de garantizar, ya que ésta solo implica restricciones sobre la topología iBGP y no requiere evaluar al proceso de decisión de BGP en cada router. Sin embargo, existen algunos casos donde la adición de sesiones spurious es deseable para solucionar algunos problemas de forwarding o mejorar la diversidad de las rutas, como lo hemos discutido antes. En estos casos, sesiones UP pueden ser utilizadas en lugar de sesiones spurious OVER, sin la adversidad de afectar la dissemination correctness.

Cuando es necesaria la adición de sesiones para solucionar problemas de visibilidad, deben utilizarse sesiones UP, para asegurar la *no-spurious-OVER.* Debemos notar que la utilización de sesiones UP tiene algunos costos adicionales, por ejemplo, se modifican las capas de la jerarquía, impactan en la memoria de los routers, etc. Algunos de estos efectos secundarios pueden ser mitigados, por ejemplo configurando filtros de rutas que permitan la propagación de rutas en una única dirección.

2.7 Algoritmos que intentan resolver el problema

El problema de la localización de reflectores de rutas ha sido estudiado y se han diseñado varios algoritmos que intentan resolverlo. En este trabajo se estudiaron e implementaron varios de estos algoritmos, así como también varias heurísticas utilizadas comúnmente por los operadores de red. Los algoritmos considerados fueron los siguientes:

Óptimo Buob: El algoritmo óptimo de Buob busca construir una topología iBGP que cumpla las propiedades deseables de una configuración full mesh. La optimalidad se refiere a lo que se llama fm-optimalidad, significando que el algoritmo diseñado elige para cada punto de egreso la mejor ruta que hubiera elegido en una configuración de full mesh. Por más información consultar [5].

BGPSep: Algoritmo que utiliza la noción de un grafo separador (un pequeño conjunto de nodos cuya eliminación parte a un grafo en componentes conexas de aproximadamente el mismo tamaño) para elegir reflectores de rutas y sesiones iBGP de manera de garantizar la correctitud. Por más información consultar [6].

BGPSepD: Basado en BGPSep, mejora el algoritmo original mediante la eliminación de algunos vértices cuyos grados satisfacen algunas condiciones, gradualmente a partir del grafo IGP. Como resultado se obtienen topologías iBGP con mucho menor grado máximo y un número mucho menor de sesiones iBGP que la producida por BGPSep. Por más información consultar [7].

BGPSepS: También basado en BGPSep, este algoritmo construye una topología iBGP teniendo en cuenta el grado de los vértices, los vértices separadores y los caminos más cortos entre los vértices en el grafo IGP subyacente. Por más información consultar [8].

Full mesh: Genera una configuración iBGP de full mesh, que consiste en una configuración en la cual existe una sesión iBGP entre todo par de routers del AS.

Se tratarán en más profundidad estos algoritmos en la sección de implementación.

Además, las heurísticas implementadas fueron las siguientes:

Zhang: Esta heurística consiste en diseñar una configuración iBGP con herencia, donde los reflectores de rutas del nivel n son clientes de los reflectores de rutas del nivel n-1. Se definen a lo sumo dos o tres niveles.

Bates: Esta heurística recomienda que para configurar una topología iBGP con uno o múltiples reflectores de rutas pertenecientes a un mismo PoP en la red,

todos los routers en el PoP (que no son reflectores) deben ser clientes de todos los reflectores de rutas del PoP. A su vez, los reflectores de rutas deben estar configurados en full mesh. Finalmente, se recomienda establecer un full mesh entre todos los routers del PoP que no son reflectores.

Ambas heurísticas son muy utilizadas en la práctica. Se profundizará sobre estas heurísticas nuevamente en la sección de implementación.

[3]

Framework

3.1 Motivación

Se buscó diseñar e implementar un framework que simplifique la implementación de algoritmos que busquen resolver el problema de localización de reflectores de rutas dentro de un sistema autónomo debido a que no existe algo similar y nos resultaría de gran utilidad en el desarrollo del proyecto.

El objetivo principal del framework es brindar las herramientas necesarias que faciliten la implementación de nuevos algoritmos. El framework deberá lograr una abstracción del algoritmo particular que resuelve el problema (verlo como si fuese una caja negra) y definir interfaces de entrada y salida que simplifiquen el manejo de datos y la integración con otras herramientas.

3.2 Toolbox of Traffic Engineering Methods (TOTEM)

El framework fue implementado integrado a TOTEM [2], una herramienta para trabajar con ingeniería de tráfico.

TOTEM provee un conjunto de herramientas donde los investigadores pueden integrar sus algoritmos de ingeniería de tráfico. Estos algoritmos por lo tanto pueden ser aplicados en modelos de redes reales. TOTEM también ofrece a los operadores de red la oportunidad de experimentar nuevos algoritmos de ingeniería de tráfico en su propia red.

Se decidió implementar el framework integrado en TOTEM principalmente por las siguientes razones:

- TOTEM es de código abierto e implementado en lenguaje Java.

- El framework abierto proporcionado por TOTEM permite una rápida integración de nuevos algoritmos. Esto permite la reutilización de componentes ya existentes, como por ejemplo la representación de la topología IGP. La ejecución de los algoritmos es realizada en un hilo separado de la interfaz, característica especialmente útil para algoritmos complejos que requieren un largo tiempo de ejecución.

- Presenta una muy buena interfaz de usuario que permite mostrar las topologías de una manera flexible y conveniente. La interfaz gráfica de usuario utiliza la librería JUNG [9]. También puede crear gráficos a partir de los datos de los dominios cargados.

- Utiliza un formato XML interoperable para representar topologías, rutas etiquetadas y configuraciones BGP (volveremos sobre este formato más adelante en esta sección). La matriz de tráfico también se representa mediante un archivo XML. Estos datos pueden ser importados a partir de una variedad de fuentes tales como configuraciones de routers, MRT [RFC 6396], NetFlow [RFC 3954], SRLG [RFC 4203], etc.

- Incluye C-BGP [10], un simulador eficiente del proceso de decisión de BGP. Esta herramienta se puede utilizar para evaluar el impacto de las políticas de entrada/salida en las tablas de routing de los sistemas autónomos. También se puede utilizar para experimentar con un proceso de decisión modificado y atributos adicionales BGP.

3.3 La arquitectura de TOTEM

La arquitectura de TOTEM se encuentra dividida en dos componentes principales: el repositorio de algoritmos y un conjunto de módulos, cada uno con una o varias funcionalidades específicas.

El núcleo de TOTEM es el repositorio de algoritmos, que se encuentra agrupado en varias categorías (ver Figura 3.1).

- IP: algoritmos que sólo usan información IP (por ejemplo, optimización del peso IGP).
- MPLS [RFC 3031]: algoritmos basados en etiquetas MPLS TE (por ejemplo, algoritmos LSP).
- BGP: algoritmos inter-dominio (por ejemplo, redistribución de tráfico).
- Generic: algoritmos de búsqueda útil para otras partes de la herramienta (por ejemplo, búsqueda tabú [16]).

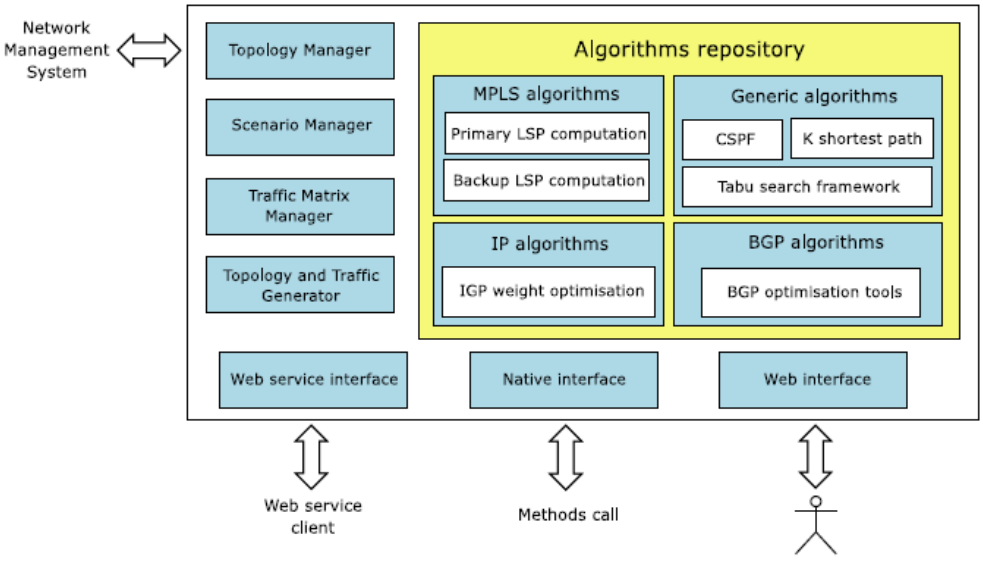


Figura 3.1: Arquitectura de TOTEM

Además de este núcleo, el manejador de topologías (Topology Manager) contiene todos los datos topológicos (por ejemplo, nodos, enlaces, IGP, BGP e información MPLS). Este módulo es el punto de acceso de referencia para la representación de la topología en la herramienta. El manejador de configuración (Configuration Manager) configura todos los parámetros globales de la herramienta y los de los diferentes algoritmos. Finalmente, el módulo de

Web Interface presenta las funcionalidades de la herramienta al usuario y el módulo Web Service Interface proporciona funcionalidades para la interoperabilidad con herramientas externas.

3.4 Un formato estándar para la representación de una red (XML)

Un aspecto común para todos los algoritmos es que deben utilizar una representación de la red tanto como para la entrada como para la salida. TOTEM utiliza el lenguaje XML, ya que es ampliamente utilizado y existen muchas herramientas para su manejo. El formato XML que representa a una topología de red puede ser visto como una interfaz común entre diversos algoritmos o herramientas.

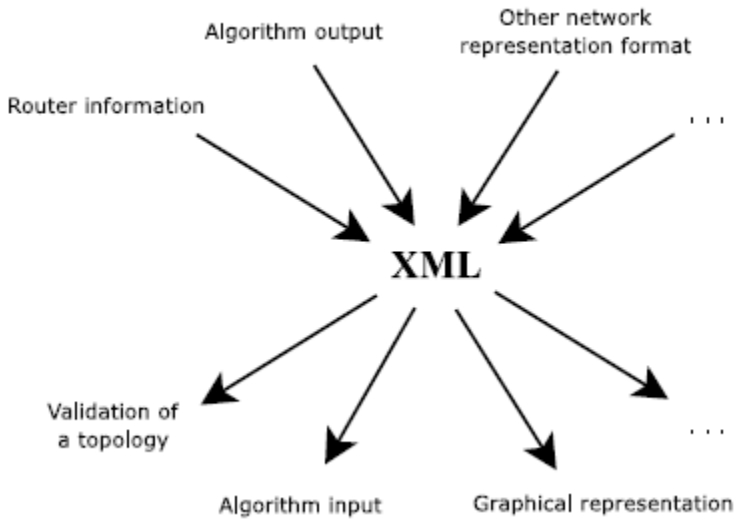


Figura 3.2: Formato XML como un interfaz común

TOTEM proporciona algunas herramientas para convertir este formato en otros formatos comunes (por ejemplo, BRITE [11], ns-2 [12], GT-ITM [13], etc) y viceversa, permitiendo la interoperabilidad entre distintas herramientas.

En cuanto a la capacidad de descripción del formato de archivo XML definido por TOTEM se puede representar:

- A nivel de IGP; nodos y links. Para los nodos nos permite representar los conceptos de id, IP, ubicación e interfaces. En cuanto a los links respecta se puede representar el id, destino, origen, tipo, ancho de banda y delay.

- A nivel BGP; podemos representar la configuración IBGP; reflectores de rutas, router BGP clásicos, router BGP virtuales, sesiones (virtuales y reales), filtros de entrada/salida para los routers BGP y routers de borde.

Además dado un archivo XML como entrada (representando una red) TOTEM puede producir una representación de la misma a nivel de objetos, lo que denomina dominio, y que facilita enormemente el trabajo ya que los algoritmos trabajan sobre los dominios. No es necesario además trabajar directamente sobre el archivo XML, TOTEM provee un editor de topologías capaz de crear y modificar topologías en el formato XML definido. Finalmente, TOTEM valida la consistencia de la topología a través de la creación de un esquema XML. Los esquemas XML permiten validar una topología de modo de asegurar que una instancia XML satisface la estructura del formato definido. Por ejemplo, es posible verificar que todos los enlaces están conectados a los nodos presentes a la topología, o que los identificadores son únicos en todo el fichero.

3.5 Cómo integrar un nuevo algoritmo al framework

Existen dos formas de agregar un algoritmo a TOTEM, la primera por implementación de la interfaz nativa que la herramienta provee (*TotemAlgorithm*) y la segunda por implementación de la interfaces específicamente diseñadas para facilitar la implementación de algoritmos que resuelvan el problema de localización de reflectores de rutas (*RRLocAlgorithm y BindAlgorithm*). Estas interfaces facilitan el desarrollo brindando funcionalidades comunes para este tipo de algoritmos.
Cuando un algoritmo es agregado al framework éste se integra al mismo sin necesidad de hacer cambios sustanciales, no son necesario cambios ni en la interfaz (para determinar la forma en la que se debe acceder al algoritmo) ni en la estructura de clases (en lo que respecta al diseño del propio TOTEM). La implementación de estas interfaces permite al usuario focalizarse en la propia implementación de su algoritmo.

TotemAlgorithm

Esta interfaz es la provista nativamente por TOTEM para agregar nuevos algoritmos. A continuación se presenta dicha interfaz.

```
public interface TotemAlgorithm {

    /**
     * Used to start and initialize the algorithm
     */
    public void start(HashMap params) throws AlgorithmInitialisationException;

    /**
     * Used to stop the algorithm
     */
    public void stop();

    /**
     *  Returns the optional parameters that can be given when starting the algorithm
     */
    public List<ParameterDescriptor> getStartAlgoParameters();

    /**
     * Returns the parameters given when the algorithm was started
     */
    public HashMap getRunningParameters();

}
```

Describiremos brevemente cada uno de los métodos de la interfaz.

```
public void start(HashMap params) throws AlgorithmInitialisationException;
```

Este método es llamado para ejecutar el algoritmo. Recibe como parámetro el hash *params* donde el tipo de clave y el tipo de los valores mapeados es string. Es decir, *params* es un conjunto de tuplas *(name_attribute, value_attribute)*

que indica para cada parámetro recibido por el algoritmo (determinado por el usuario) su valor correspondiente. El nombre del parámetro funciona como clave en el hash, por lo tanto el usuario debe cuidar que los nombres de los parámetros recibidos por el algoritmo sean únicos (no existe ningún tipo de control a nivel de la herramienta).

```
public void stop();
```

Este método es invocado al finalizar el algoritmo. Puede ser un método vacío, su tarea depende del algoritmo. Un ejemplo de uso podría ser restablecer el estado inicial del algoritmo para futuras ejecuciones.

```
public List<ParameterDescriptor> getStartAlgoParameters();
```

Este método es llamado antes de ejecutar el algoritmo (método start()) y es utilizado por TOTEM para crear el formulario de parámetros de entrada del algoritmo. Es decir, para ejecutar el algoritmo se necesita indicar qué valores tendrán los parámetros de entrada del mismo. TOTEM soluciona este problema creando un formulario dónde se muestran tres columnas, la primera el nombre del atributo (o parámetro), la segunda el valor a tomar por el atributo (es un campo editable) y finalmente una breve descripción de la función del atributo en el algoritmo. La clase *ParameterDescriptor* se utiliza para representar un parámetro del algoritmo y está constituido por un nombre, una descripción, un tipo, un conjunto de valores posibles (esto es opcional) y un valor predeterminado. De esta forma, una vez llamado el algoritmo TOTEM soluciona de forma elegante la introducción de parámetros por parte del usuario mostrando en pantalla un formulario donde se muestran todos los parámetros necesarios para la ejecución. Por más información de cómo ejecutar un algoritmo desde la interfaz de TOTEM consulte el manual [2].

```
public HashMap getRunningParameters();
```

El uso habitual de este método es retornar una copia del hash de parámetros *params* recibidos en el método start(), siempre y cuando el usuario haya tenido el cuidado de guardar el hash *params* antes de ejecutar el algoritmo. Puede ser utilizado por el usuario para determinar con que parámetros fue llamado el algoritmo en tiempo de ejecución.

RRLocAlgorithm y BindAlgorithm

Ambas interfaces fueron creadas con el fin de facilitar el desarrollo de algoritmos de ubicación de reflectores de rutas. Su principal motivación surgió de la necesidad de poder ejecutar los algoritmos independientes de la interfaz de TOTEM (es decir independientes de *TotemAlgorithm*) y además evitar al programador escribir código repetitivo. Es decir, las tareas en la ejecución de este tipo de algoritmos (de reflectores de rutas) son siempre las mismas. Se pueden identificar a grandes rasgos las siguientes etapas; *a)* determinar los parámetros de entrada (obligatoriamente la topología), *b)* ejecución del algoritmo y finalmente *c)* impacto de los resultados obtenidos (éstos pueden abstraerse a un lista de sesiones resultantes). En ese contexto estas interfaces resuelven esas etapas sin requerir trabajo extra del programador (siempre y cuando se sigan ciertas imposiciones).

RRLocAlgorithm

Esta interfaz resuelve el problema de la ejecución independiente de la interfaz de TOTEM (*TotemAlgorithm*). A continuación se muestra dicha interfaz, que posee un único método.

```
public interface RRLocAlgorithm {
    /**
     * Used to start and initialize the algorithm
     */
      public int run(Object in_params, Object out_result);

}

public int run(Object in_params, Object out_result);
```

Este método es llamado para ejecutar e inicializar el algoritmo. Es un método abstracto capaz de adecuarse a cualquier forma de algoritmo con variados parámetros de entrada y de salida. Como se puede observar, tanto *in_params* como *out_result* son de tipo Object, permitiendo al programador el pasaje de cualquier tipo de parámetro tanto de entrada como de retrono (sugerimos al programador tener cuidado con los casteos de tipos). Este método debe encapsular sólo y únicamente el comportamiento algorítmico (lo que correspondería a la etapa *b)*).

Como se puede ver la implementación de esta interfaz desasocia el uso del algoritmo a la interfaz de TOTEM. Por ejemplo, un *RRLocAlgorithm* puede ejecutarse desde cualquier programa principal u otra GUI distinta de TOTEM). Está claro que además puede ejecutarse desde el propio TOTEM.

BindAlgorithm

Esta es una clase abstracta que resuelve todo lo concerniente a las etapas *a)* y *c)*. Implementa a la interfaz *TotemAlgorithm*, provee un conjunto de métodos ya implementados y además permite la ejecución en background a los algoritmos. En definitiva, provee funcionalidades propias de la asociación (binding) de un algoritmo de tipo *RRLocAlgorithm* con TOTEM. A continuación se presenta la interfaz.

```
public abstract class BindAlgorithm implements Runnable, TotemAlgorithm {

	protected Logger logger = null;
	protected RRLocAlgorithm algorithm = null;
	protected ArrayList<ParameterDescriptor> params = null;
	protected String name = null;
	protected Thread thread = null;
	protected Domain domain = null;

	/**
	 * Used for initialize the parameters of input
	 *
	 */
	public abstract Object getAlgorithmParams(HashMap params);

	/**
	 * It is called for initialize the result parameter
	 *
	 */
	public abstract Object initAlgorithmResult();

	/**
	 * Used in debug mode, log the result of algorithm in logger
	 *
	 */
	public void log(Object algorithmResult) {..};

	/**
	 * It is called when a algorithm end and is necessary impact
	 * the changes in the domain
	 *
	 */
	public void dumpResultInDomain(Object algorithmResult) throws Exception {..};

	public RRLocAlgorithm getAlgorithm() {..};

	@Override
	public void start(HashMap params)throws AlgorithmInitialisationException {..};

	@Override
	public void stop() {..};

	@Override
	public HashMap getRunningParameters() {..};

	@Override
	public List<ParameterDescriptor> getStartAlgoParameters() {..};

	@Override
```

```
        public void run() {..};

}
```

Para usar todo los beneficios dados por BindAlgorithm y no tener que reimplementar todos sus métodos se deberán tener en cuenta los siguientes requerimientos.

1) *algorithm* deberá implementar la interfaz *RRLocAlgorithm*, donde el tipo del parámetro resultado (*out_result*) deberá ser *List<iBGPSession>*. La clase *iBGPSession* representa el tipo de sesión existente entre un par de routers, ésta puede ser cliente o par.

2) deberá tener como parámetro obligatorio de entrada una topología (sobre la cual se ejecutará el algoritmo) y esta deberá ser asignada al atributo *domain* en el método *getAlgorithmParams*(..).

3) la clase que extiende a *BindAlgorithm* deberá iniciar en su constructor obligatoriamente los siguientes parámetros: *logger*, *algorithm*, *params* y *thread*.

Si se cumplen los anteriores requerimientos no será necesario modificar ningún método implementado por *BindAlgorithm*.

A continuación mostraremos un ejemplo típico de inicialización de los parámetros obligatorios en el constructor, en este caso tomaremos como ejemplo al algoritmo *BGPSep.*

```
public BGPSep() {
        logger = Logger.getLogger(BGPSep.class);
        params = new ArrayList<ParameterDescriptor>();
        algorithm = new BGPSepAlgorithm();
        name = "BGPSep";
        thread = new Thread(this, name);

        params.add(new ParameterDescriptor("ASID",
                            "Domain ASID (leave blank for default).",
                            Integer.class,
                            null));

        params.add(new ParameterDescriptor("Separator",
                            "Methauristic utiliced for graph separator",
                            String.class,
                            "GRASP",
                            new String[]{"GRASP", "EA"}));
}
```

Como se puede observar es necesario asignar un log (*logger*), la descripción de los parámetros de entrada (*params*) usada por TOTEM para crear el formulario de ingreso de los mismos al algoritmo, el nombre del algoritmo (*name*), asignar el algoritmo (*algorithm*) que implementa *RRLocAlgorithm* y finalmente

asignar el hilo (*thread*) que proporcionará la ejecución desasociada de la interfaz.

A continuación pasamos a detallar los métodos de la clase *BindAlgorithm.*

```
public abstract Object getAlgorithmParams(HashMap params);
```

Recibe como parámetro el hash *params* donde el tipo de clave y el tipo de los valores mapeados es string. Es decir, *params* es un conjunto de tuplas *(name_attribute, value_attribute)* que indica para cada parámetro recibido por el algoritmo (determinado por el usuario) su valor correspondiente. El nombre del parámetro funciona como clave en el hash por lo tanto se debe cuidar que los nombres de los parámetros recibidos por el algoritmo sean únicos. Retorna un tipo Object que es usado como entrada (*in_params*) en el algoritmo *algorithm*. En este mismo método en necesario asignar al atributo *domain.* El método es llamado antes del método start().

```
public abstract Object initAlgorithmResult();
```

Este método se encarga de inicializar los parámetros de salida (*out_result*) del algoritmo *algorithm*. Generalmente se pide la memoria para guardar los resultados. El método es llamado antes del método start().

```
public void log(Object algorithmResult) {..};
```

En modo debug realiza la impresión de los resultados obtenidos (en forma de *List<iBGPSession>*) por el algoritmo. Si no cumple con esa representación de los resultados se deberá reimplementar el método. El método es llamado al finalizar la ejecución del algoritmo.

```
public void dumpResultInDomain(Object algorithmResult) throws Exception {..};
```

Realiza la bajada de los resultados obtenidos (en forma de *List<iBGPSession>*) por el algoritmo sobre el dominio *domain*. Si no cumple con esa representación de los resultados se deberá reimplementar el método. El método es llamado al finalizar la ejecución del algoritmo.

```
public void start(HashMap params)throws AlgorithmInitialisationException {..};
```

Llama a ejecutar el algoritmo *algorithm*, este método ejecuta el método *run(..)* el cual permite la ejecución desasociada de la interfaz.

[4]

Algoritmos implementados

4.1 Óptimo Buob

El algoritmo óptimo de Buob busca construir una topología iBGP que cumpla las propiedades deseables de una configuración full mesh. La optimalidad se refiere a lo que se llama fm-optimalidad, significando que el algoritmo diseñado elige para cada punto de egreso la mejor ruta que hubiera elegido en una configuración de full mesh. La presentación de esta sección se basa en el documento que describe al algoritmo, puede ser consultado en [5].

Antes de presentar como abordar el diseño del algoritmo recordemos las principales ventajas y desventajas de configuraciones full mesh y configuraciones con reflectores de rutas. El full mesh es determinista y siempre resulta en un routing óptimo. La convergencia es rápida y la red es lo más robusta posible. Sin embargo, ésta no es escalable, muchas sesiones iBGP son necesarias, y agregar o remover routers implica un costo adicional de configuración muy significativo. Además, un cambio en la mejor ruta provoca mensajes de actualización hacia todos los routers. Por otra parte, con reflectores de rutas la escalabilidad es mejorada en favor del costo adicional de configuración, la convergencia, y el tamaño de las tablas de routing. Sin embargo, como ya hemos mencionado, esto es pagado con la disminución de la diversidad de las rutas, se induce normalmente un routing subóptimo, puede no ser determinista, pueden existir oscilaciones, y pueden ocurrir desviaciones que provocan forwarding loops. Además, el comportamiento de las topologías que utilizan reflectores de rutas cuando ocurren fallas o cambios en la topología IGP no es claro.

El algoritmo presentado respetará simultáneamente varios requerimientos esenciales:

- Fm-optimalidad: los reflectores de rutas son una alternativa escalable en relación al full mesh, pero como mencionamos, no existe el compromiso de selección de la ruta óptima que se elegiría en caso de utilizar full mesh. Se presenta un algoritmo que toma lo mejor de ambos mundos, si bien no es trivial, es posible.
- Correctitud: como hemos mencionado, verificar la correctitud de una topología es un problema NP-hard. Pero gracias a la fm-optimalidad, podemos asegurar sin embargo que la red es libre de loops (debido a que será libre de desviaciones), y determinista, y por lo tanto correcta.
- Fiabilidad: el diseño de la topología iBGP seguirá el grafo IGP lo más posible. Solo se establecerán sesiones multi hop (sesiones iBGP que atraviesan otros routers BGP además de los dos extremos de la sesión) cuando sea necesario.
- Robustez: se construirán topologías robustas que soporten fallas en enlaces IGP y routers. Además, luego de la falla de un enlace o un router la topología continuará siendo fm-óptima.

- Escalabilidad: se construirán topologías con la menor cantidad de sesiones iBGP posibles (teniendo en cuenta que deben cumplir con todos los requisitos que listamos anteriormente).

Terminología

Grafo IGP: Sea $G_{igp}(V_{igp}, E_{igp})$ la topología física de la red, cada vértice de V_{igp} un router y cada arco con peso (u, v) de E_{igp} un enlace físico y su métrica. Denotaremos $dist : V_{igp}\ x\ V_{igp} \rightarrow \mathbb{N}$ la función que retorna el peso del camino más corto entre dos routers.

Grafo BGP: Denotaremos como N el conjunto de los posibles next hops BGP (los routers de borde) en las rutas aprendidas por el sistema autónomo, y R el conjunto de los routers BGP dentro del sistema autónomo. El grafo $G_{bgp}(V_{bgp}, E_{bgp})$ describe la topología de routers reflectores. $V_{bgp} = R \cup N$. E_{bgp} denota el conjunto de sesiones entre los routers. Cuando dos routers comparten una sesion iBGP, agregamos dos aristas entre los routers etiquetadas con UP desde el cliente hacia uno de sus reflectores, y con DOWN desde el reflector de rutas hacia uno de sus clientes, u OVER entre peers.

Asumiremos que los routers de borde del sistema autónomo serán los next hop BGP para las rutas (esto es $N \subseteq R$). Denotaremos como $L = UP, OVER, DOWN$ el tipo de las sesiones iBGP y como $label: E_{bgp} \rightarrow L$ la etiqueta de un enlace dado. Además denotaremos por $sym: L \rightarrow L$ la función que retorna la etiqueta simétrica de una etiqueta dada: $sym(UP) = DOWN$, $sym(DOWN) = UP$ y $sym(OVER) = OVER$.

Un camino BGP en el grafo G_{bgp} es un camino válido si se compone de cero o más arcos UP, seguidos de cero o un arco OVER, y seguido de cero o más arcos DOWN. Cualquier secuencia de etiquetas válida respeta la siguiente expresión regular: $(UP) * (OVER)? (DOWN) *$.

Sea $(n, r) \in N\ x\ R$, n un determinado next hop para r. Cuando consideramos este par (n, r) asumimos que:

- Existe un prefijo p y varias rutas (llamadas las rutas actuales) para este prefijo en el sistema autónomo, las cuales no empatan en su costo relacionado en el grafo IGP.
- n es el next hop BGP más cercano de r en el grafo IGP.

Trataremos de asegurar de que r siempre es capaz de aprender la ruta anunciada por n, el next hop BGP más cercano. Sólo debemos considerar el siguiente conjunto de next hops:

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

Si existe un camino iBGP válido de n a r tal que cada router w de este camino selecciona la ruta anunciada por n, entonces r aprende la ruta anunciada por n. Llamaremos router blanco a un router que cumple con esta propiedad. El conjunto de los routers blancos relacionado a un par (n,r) dado es definido como:

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

Nótese que siempre n y r pertenecen a $W(n,r)$. Además, $N(n,r) \cap W(n,r) = \emptyset$. Diremos que un camino es un camino blanco si está formado sólo por routers blancos. Si para cada $(n,r) \in NxR$ existe al menos un camino blanco, entonces la topología será fm-optimal. Nótese que la fm-optimalidad es independiente del prefijo. Este criterio asegura un "buen comportamiento" de la red para cualquier conjunto de rutas BGP.

La Figura 4.1 provee un breve ejemplo que ilustra la fm-optimalidad.

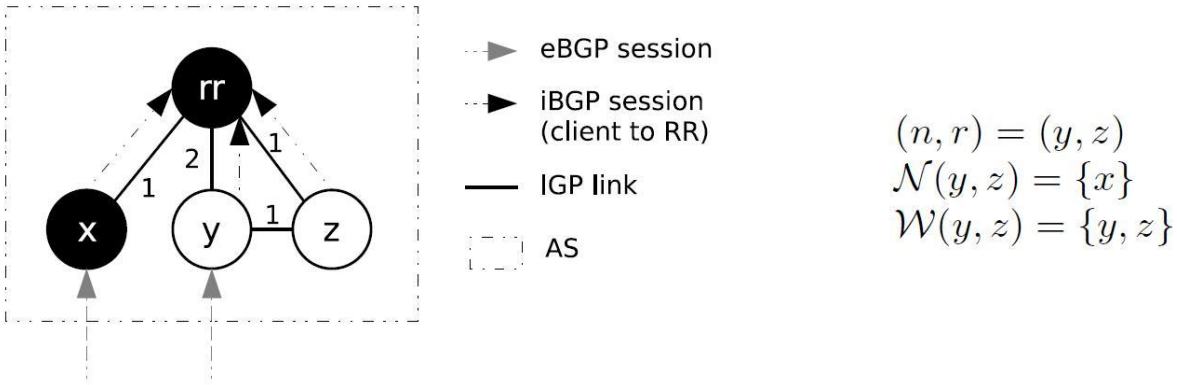


Figura 4.1: Ejemplo de routing sub óptimo

En esta topología, (y,rr,z) es un camino iBGP válido de y a z. Sin embargo $rr \notin W(y,z)$. El camino (y,rr,z) no es un camino blanco. En efecto, rr selecciona la ruta anunciada por x y z y es incapaz de aprender la ruta anunciada por su next hop BGP más cercano y.

Diseño del algoritmo

Ahora detallaremos el diseño del algoritmo, que asegura la construcción de topologías fm-óptimas. Como entrada necesitamos el conjunto de next hops BGP (N), el conjunto de routers BGP (R), y la topología IGP (G_{igp}).

En el primer paso, se presenta un acercamiento para resolver el problema de diseño de topologías iBGP cuando las fallas no son posibles, a este caso lo denominaremos el caso nominal. Para cada par $(n,r) \in NxR$, construiremos un problema satélite el cual será satisfecho si y sólo si al menos un camino blanco existe desde n a r.

En el segundo paso, detallaremos como introducir restricciones para agregar tolerancia a fallas. Construiremos un problema satélite para cada tripleta (n, r, f) para una falla dada f.

Caso nominal

El problema principal

Variables: Para cada (u, v), sesión iBGP candidata, $(u, v) \in R \; x \; R, \; u \neq v$, definimos dos variables booleanas: $up(u, v)$ (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: Diseñaremos una topología iBGP lo más cercana posible a la topología IGP mientras minimizamos el número de sesiones iBGP instaladas. Denotaremos con F a la función objetivo definida como:

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

Donde $R(u, v)$ caracteriza el número de saltos en la topología IGP necesarios para establecer una sesión iBGP entre u y v (este valor es igual al largo del camino más corto en el grafo IGP desde u a v).

Restricciones: El problema principal está compuesto por dos conjuntos de restricciones:

- Restricciones de dominio: cada par $(u, v) \in RxR$ está conectado por 0 o 1 sesión iBGP, y además $label(u, v) = sym(label(v, u))$. Esto nos conduce a las siguientes restricciones lineales:
  - $\forall \, u, v \; \in R, up(u, v) + down(u, v) \leq 1$
  - $\forall \, u, v \; \in R, up(u, v) = down(v, u)$
- Restricciones de corte mínimo y flujo máximo: Al comienzo, este conjunto de restricciones está vacío. Más adelante explicaremos como se agregan restricciones a este conjunto.

En cada iteración, el problema principal consulta a los problemas satélites para verificar la fm-optimalidad para su par (n, r) correspondiente. Cada problema satélite consultado insatisfecho inserta una nueva restricción de corte mínimo y flujo máximo dentro del problema principal. El conjunto de restricciones de corte mínimo y flujo máximo asegura la propagación de rutas entre cualquier par de routers (n, r) a través del grafo iBGP. Si todos los problemas satélites son satisfechos, la resolución del problema retorna una solución fm-óptima que minimiza la función objetivo F.

Problemas satélites

Grafo extendido: Para garantizar que solo puedan ser construidos caminos iBGP válidos utilizaremos una transformación del grafo introducido en [23]. Transformaremos cada vértice de V_{bgp} en un meta-nodo compuesto de dos nodos (llamados nodo origen y nodo destino) y un arco (llamado arco interno) de acuerdo a la Figura 4.2.

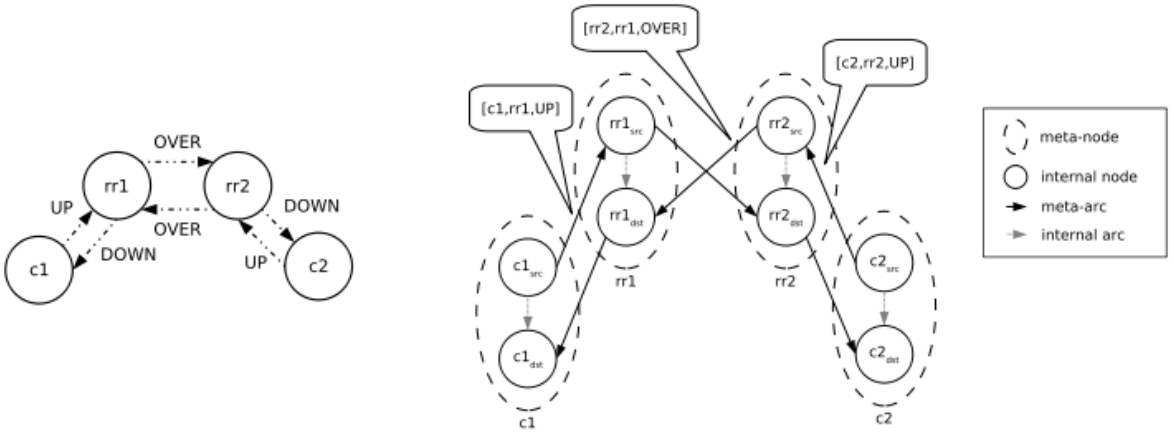


Figura 4.2: Un ejemplo de un grafo iBGP y su correspondiente grafo extendido

La forma en la que se enlazan dos meta-nodos depende de la relación iBGP entre los dos routers relacionados. En el grafo extendido sólo podemos construir caminos iBGP válidos. Llamaremos meta-arco al arco que conecta dos vértices pertenecientes a distintos meta-nodos. En este grafo, cada meta-arco es mapeado a una sesión iBGP y los dos routers establecerán esta sesión iBGP. Denotamos como $[u, v, rel]$ al meta-arco que se asigna a los meta-nodos u y v, y a la sesión iBGP rel. Cada camino válido de G_{bgp} desde $s \in V_{bgp}$ y hacia $t \in V_{bgp}$ es así asignado a exactamente un camino en el grafo extendido desde s_{src} a t_{dst}, donde s_{src} es el nodo de origen perteneciente al meta-nodo s y t_{dst} el nodo destino perteneciente al meta-nodo t.

Grafo satélite: Para asegurar la fm-optimalidad dado un par (n,r), construiremos un problema satélite. Para cada problema satélite construiremos el grafo satélite $G_w(n,r)$. Cada vértice de este grafo pertenece a $W(n,r)$. Para reducir el número de sesiones iBGP candidatas, sólo consideraremos las sesiones (u,v) que verifican las siguientes propiedades:

1. $u, v \in W(r,n)$: la ruta BGP enviada por n sólo atraviesa routers que no ocultan la ruta.
2. $dist(n,u) \leq dist(n,v)$ y $dist(v,r) \leq dist(u,r)$: los mensajes BGP atraviesan el arco (u,v) incrementando la distancia a n y decrementando la distancia a r.

Las sesiones iBGP que no verifican el punto 1 podrían causar que las rutas fm-optimales no sean propagadas, por esto buscamos no utilizar estas sesiones. El punto 2 previene que mensajes iBGP anunciados por n sigan un camino muy largo desde n a r. Nótese que la configuración de una topología iBGP full-mesh es una solución posible porque la sesión directa entre n y r está presente.

Así, el grafo $G_w(n,r) = (W(n,r), E_w(n,r))$ mantiene los caminos blancos iBGP candidatos para satisfacer el par (n,r). Si para todo $(n,r) \in NxR$, existe al menos un camino válido de n a r en $G_w(n,r)$, entonces la topología iBGP será fm-óptima.

Problema satélite: Para cada par (n,r) construimos el grafo extendido $G_w^{ext}(n,r)$ de $G_w(n,r)$ y todos los meta-arcos candidatos. Denotaremos como n_{src} el nodo origen perteneciente al meta-nodo n y r_{dst} el nodo destino que pertenece al meta-nodo r. Asignaremos para cada arista $(i,j) \in G_w^{ext}(n,r)$ una capacidad, como se puede ver en la Figura 4.3.

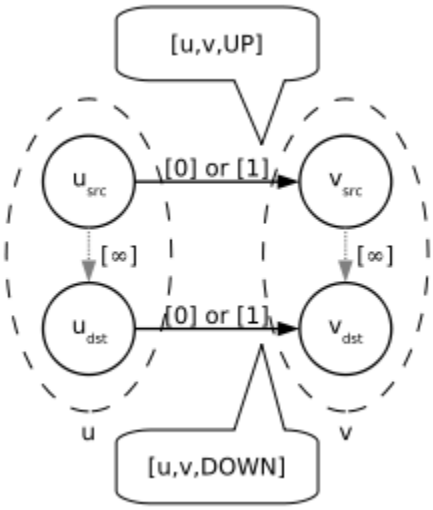


Figura 4.3: Dos sesiones iBGP candidatas

- Si i y j pertenecen al mismo meta-nodo, establecemos la capacidad de (i,j) como infinito.
- En otro caso (i,j) es un meta-arco. Sea $rel \in UP, DOWN$ la relación iBGP asignada a (i,j),r_i el meta-nodo al que pertenece i, y r_j el meta-nodo al que pertenece j. Si la sesión iBGP rel es establecida desde i hacia j, entonces establecemos la capacidad del arco (i,j) igual a 1 (0 en otro caso). Por esto, a lo sumo será establecido un meta-arco desde el meta-nodo r_i hacia el r_j con capacidad igual a 1.

Si el máximo flujo enviado desde n_{src} (el origen) a r_{dst} (el destino) es mayor o igual a 1 entonces el par (n,r) es satisfecho. En otro caso, no puede existir flujo desde el origen al destino. Buscamos el corte mínimo de flujo máximo en

este grafo. Denotaremos como $C(n,r,it)$ el conjunto de meta-arcos que intersecta este corte durante la iteración it. Agregaremos las siguientes restricciones de flujo máximo dentro del problema principal:

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

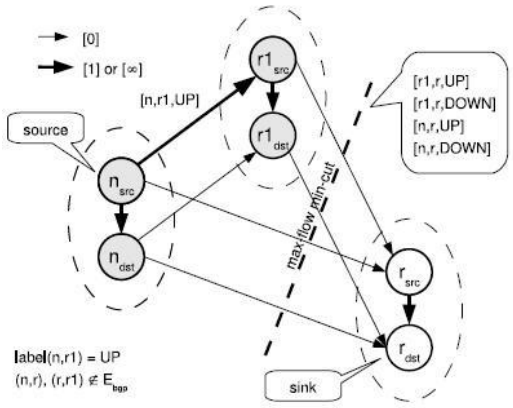


Figura 4.4: Este corte mínimo de flujo máximo agrega la siguiente restricción:

$$up(r_1,r) + down(r_1,r) + up(n,r) + down(n,r) \geq 1$$

La Figura 4.4 provee un ejemplo de un corte mínimo de flujo máximo. En este ejemplo $W(n,r) = \{n, r_1, r\}$ y las iteraciones previas han determinado que $label(n,r_1) = UP$, $label(r_1,r) = label(n,r) = NOT$. Cuando el satélite relacionado a (n,r) es consultado, debido a las restricciones de corte mínimo y flujo máximo se inserta la siguiente restricción lineal dentro del problema:

$$up(r_1,r) + down(r_1,r) + up(n,r) + down(n,r) \geq 1$$

Fallas IGP

Ahora detallaremos como tener en cuenta las fallas en el IGP. Los routers BGP utilizan el camino más corto en el grafo IGP para establecer las sesiones con sus vecinos BGP. Cuando ocurre una falla en el IGP, los mensajes BGP actualizan las rutas del camino más corto en el IGP luego de la falla y se restablece la sesión BGP perdida de acuerdo con esta nueva topología. Si no existe conectividad entre dos pares BGP, la sesión BGP no se restablecerá. Denotaremos $f = (V_{igp}^f, E_{igp}^f)$ una falla IGP, donde $V_{igp}^f \subset V_{igp}$ representa el conjunto de routers involucrados y $E_{igp}^f \subset E_{igp}$ al conjunto de enlaces IGP involucrados. Denotaremos por ∅ la falla vacía.

Metodología: Una falla f en el IGP implica recalcular el costo IGP entre cada par de routers pertenecientes a la misma componente conexa. Para cada falla IGP considerada como entrada, debemos aplicar el razonamiento utilizado en

el caso nominal en cada componente conexa. Vamos a considerar el par (n,r) tal que n y r pertenecen a la misma componente conexa en el grafo IGP C. Sólo consideramos en $G_w(n,r,f)$ los routers blancos pertenecientes a C. Una sesión iBGP puede ser establecida solamente si los dos routers BGP que comparten la sesión pertenecen a la misma componente conexa. Así, solo tenemos que considerar pares tales que n y r pertenezcan a la misma componente conexa y $n \notin V_{igp}^{f}$, y $r \notin V_{igp}^{f}$.

Agregación de satélites: Si construimos el satélite (n,r,f) para cada falla IGP (incluyendo $\emptyset$), puede verse que se generan problemas satélites redundantes. Por ejemplo, si la falla f no afecta a un par (n,r) dado, será inútil construir el satélite (n,r,f) porque (n,r,f) y $(n,r,\emptyset)$ insertan las mismas restricciones de flujo. Además, consideremos dos fallas f y f' para un par dado (n,r), y sean $G_w(n,r,f)$ y $G_w(n,r,f')$ los grafos de flujo relacionados. Si $G_w(n,r,f) \subseteq G_w(n,r,f')$, las restricciones introducidas por $G_w(n,r,f)$ son más restrictivas que las introducidas por $G_w(n,r,f')$. Así podemos omitir sin problemas el satélite $G_w(n,r,f')$.

Acerca de nuestra implementación

Realizamos una implementación del algoritmo utilizando la librería de programación matemática de IBM CPLEX [14] para el manejo del problema principal y los problemas satélites. Es importante notar que no se tomaron en cuenta las fallas IGP. En la sección de evaluación se presentan los resultados de ejecución de la implementación sobre varios ejemplos y se verifica que los mismos son correctos según la especificación.

4.2 BGPSep

BGPSep utiliza la noción de un grafo separador (un pequeño conjunto de nodos cuya eliminación parte a un grafo en componentes conexas de aproximadamente el mismo tamaño) para elegir reflectores de rutas y sesiones iBGP de manera de garantizar la correctitud. La presentación de esta sección se basa en el documento que describe al algoritmo, puede ser consultado en [6].

Los autores aseguran que luego de evaluar una implementación del algoritmo BGPSep en varias redes del mundo real y en topologías simuladas se encontró que las configuraciones iBGP generadas por BGPSep tienen entre 2,5 y 5 x menos sesiones iBGP que una configuración full mesh.

El algoritmo simplificado

Sea $G = (V, E)$ el subgrafo IGP inducido por los routers eBGP, dónde V es el conjunto de routers eBGP. Ahora se describirá cómo construir una configuración iBGP básica que satisfaga las propiedades de visibilidad completa, routing libre de bucles y robustez a fallas IGP, sin requerir una configuración full mesh. El siguiente apartado optimiza esta construcción básica.

- Paso 1: Considerar un separador de grafo S de G. Hacer todos los routers de S reflectores de rutas.

- Paso 2: Para cada $u, v \in S$, configurar a los routers u y v, como pares iBGP. Haciendo esto se forma un full mesh en el nivel superior de la jerarquía de reflectores.

- Paso 3: Hacer a cada router en $V - S$ cliente de cada reflector de ruta en S.

- Paso 4: Para cada componente conexa G_i en la que S separa G, establecer una sesión iBGP entre los routers de G_i (es decir, construir una configuración de full mesh dentro de cada componente conexa G_i).

Aunque esta construcción básica satisface las propiedades de correctitud y tiene un número menor de sesiones iBGP en comparación con full mesh (routers en diferentes componentes conexas ya no necesitan conectarse entre sí, solo necesitan conectarse a los reflectores de rutas en S), se sigue utilizando una configuración de full mesh dentro de cada uno de los componentes individuales. Para reducir aún más el número de sesiones iBGP, hay que observar que el problema de evitar una configuración iBGP de full mesh entre G_1 y G_2 sin violar las propiedades de correctitud, es solo un ejemplo más pequeño del problema original que comenzamos a resolver en G. Por lo tanto, podemos aplicar recursivamente el mismo algoritmo dentro de cada uno

de los componentes. La recursión puede terminar cuando los componentes son suficientemente pequeños para ser configurados en full mesh. Se discute el algoritmo completo en el siguiente punto.

El algoritmo completo

BGPSep
***Input*:** Grafo IGP G , conjunto V de routers BGP
***Output*:** Conjunto I de sesiones iBGP
If $|V| = 1$ **then**
    $I = \emptyset$;
else if $|V| = 2$ **then**
    $\{u, v\} \leftarrow V$;
    $I = \{(u, v, par)\}$;
else
/* Paso 1: Elegir un separador de grafo $S \subset V$. Los routers en S son los reflectores. */
$S \leftarrow Graph - Separator(G)$;
$G_1, \ldots, G_m \leftarrow$ componentes de $V - S$;
/* Paso 2: Crear un full mesh entre el conjunto de reflectores. */
foreach $u, v \in S, u \neq v$ **do**
    $I = I \cup \{(u, v, par)\}$;
end
foreach G_i **do**
    /* Paso 3: Hacer cada router en cada componente G_i cliente de cada reflector de ruta en S */
    foreach $u \in G_i, v \in S$ **do**
        $I = I \cup \{(u, v, cliente)\}$;
    end
    /* Paso 4: Aplicar recursivamente BGSep sobre cada componente */
    $I_i = BGPSep(G_i)$;
    $I = I \cup I_i$;
    end
end
return I;

El algoritmo toma un grafo $G = (V, E)$ formado por los routers BGP y produce como salida el conjunto I de sesiones iBGP que se deben establecer entre los routers. Cada elemento en I representa una sesión iBGP y es de la forma (u, v, t) donde u y v son los routers entre los cuales se establece la sesión iBGP y t es el tipo de la sesión. Si $t = cliente$, entonces la sesión iBGP entre u y v es una sesión de cliente-reflector de ruta (con u cliente del reflector de ruta v). Si $t = par$, entonces la sesión iBGP entre u y v es una sesión iBGP común. La recursión se detiene cuando el componente tiene uno o dos routers. El algoritmo utiliza un procedimiento $Graph - Separator$ que es un algoritmo de partición de grafo que toma como entrada un grafo G y retorna un separador de grafo S.

Aunque la recursión que se muestra en el algoritmo termina cuando cada componente tiene uno o dos routers, es fácil modificar el algoritmo para terminar la recursión en una etapa anterior y realizar una configuración de full mesh en cada componente conexa. En la práctica, es probable que el número máximo de niveles de recursión (que es también igual al número de niveles en la jerarquía de reflectores de rutas resultante) sea un parámetro definido por el usuario.

4.2.1 BGPSepD

BGPSepD mejora el algoritmo original mediante la eliminación de algunos vértices, cuyos grados satisfacen algunas condiciones, gradualmente a partir del grafo IGP. El algoritmo mejorado se llama BGPSepD. Se prueba que BGPSepD satisface las tres propiedades de correctitud. Los autores aseguran que resultados experimentales indican que BGPSepD puede generar topologías iBGP con mucho menor grado máximo y un número mucho menor de sesiones iBGP que la producida por BGPSep. La presentación de esta sección se basa en el documento que describe al algoritmo, puede ser consultado en [7].

El algoritmo BGPSepD

Se observó que existe una gran cantidad de vértices colgantes (grado 1) en las topologías IGP de algunos grandes ASs. Por lo tanto, se modificó el algoritmo BGPSep mediante la eliminación de los vértices colgantes, gradualmente a partir del grafo IGP. Si no hay más vértices colgantes en el subgrafo producido, a continuación se aplica BGPSep. Se llamó al algoritmo modificado BGPSepD.

La idea de BGPSepD es simple. Si el grado del nodo u es uno en el grafo IGP, entonces sea u cliente de su nodo v adyacente. Si el nodo v posee visibilidad completa, entonces siguiendo las reglas de reflexión de rutas y las de selección BGP, el nodo u también tendrá visibilidad completa.

Descripción del algoritmo

A continuación se presenta el pseudocódigo del algoritmo:

```
BGPSep_D
Entrada: Grafo IGP G , conjunto V de routers BGP
Salida: Conjunto I de sesiones iBGP
/* Paso 1: removiendo gradualmente los vértices colgantes*/
I = ∅;
pending = true;
G = G';
while pending == true do
        G = G';
        pending = false;
        foreach u ∈ G.V do
                if d_g(u) == 1 then /* d_g(u) es el grado de 'u' en G */
                        v = adj_g(u) ; /* 'v' es el nodo adyacente de 'u' */
                        I = I ∪ {(u, v, cliente)} ; /* 'u' es cliente de 'v' */
                        G' = G' − {u};
                        pending = true;
                end
        end
end
/* Paso 2: Aplicar BGPSep sobre el subgrafo resultante G' */
I_s = BGPSep(G') ;
I_s = I ∪ I_s;

return I;
```

4.2.2 BGPSepS

Este algoritmo construye una topología iBGP teniendo en cuenta el grado de los vértices, los vértices separadores y los caminos más cortos entre los vértices en el grafo IGP subyacente. Se probó que BGPSep_S garantiza una visibilidad completa en situaciones normales.

Aunque la configuración iBGP generada por BGPSep garantiza las tres propiedades de correctitud: visibilidad completa, libre de bucles y solidez a fallos IGP, y el número de sesiones iBGP es más pequeño que en una configuración de full mesh, BGPSep no reduce el número de sesiones iBGP de sus reflectores de rutas de alto nivel.

Los resultados experimentales aportados por los autores muestran que los grados máximos de las topologías iBGP generadas por BGPSep_S para topologías IGP del mundo real se pueden reducir en aproximadamente un 27%-68%, en comparación con full mesh y BGPSep. La presentación de esta sección se basa en el documento que describe al algoritmo, puede ser consultado en [8].

Visibilidad completa y configuración iBGP

Para entender la idea y las características del algoritmo, esta sección discute las relaciones entre las configuraciones iBGP, visibilidad completa, bucles y caminos sub-óptimos.

En primer lugar, se describen algunas notaciones, definiciones y lemas relacionados.

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

Se necesitan definir los siguientes términos.

Cadena de señalización: Una cadena de señalización entre dos routers A y B se define como un conjunto de routers $A(=R_0),\ R_1, R_2, .. R_r, B(=R_{r+1})$ tal que, para $i = 1 \dots r$,

(i) R_i es un reflector de ruta

(ii) al menos uno de R_{i+1} o R_{i-1} es cliente de R_i.

Cadena de señalización de incremento monótono: Una cadena de señalización de incremento monótono entre dos routers A y B es una cadena de

señalización $S{:}A(=R_0),\ R_1,R_2,..R_r,B(=R_{r+1})$ tal que, para $i=1\ldots r+1,\ R_{i-1}$ es cliente de R_i.

Cadena de señalización de decremento monótono: Una cadena de señalización de incremento monótono entre dos routers A y B es una cadena de señalización $S{:}A(=R_0),\ R_1,R_2,..R_r,B(=R_{r+1})$ tal que, para $i=1\ldots r+1,\ R_i$ es cliente de R_{i-1}.

Concatenación de cadenas de señalización: Dadas dos cadenas de señalización $S_1=R_0,R_1,\ldots,R_k$ y $S_2=R_0{'},R_1{'},\ldots,R_l{'}$, si $R_k=R_0{'}$ y $R_0,\ldots,R_k(=R_0{'}),R_1{'},\ldots,R_l{'}$ es una cadena de señalización, entonces decimos que S_1 se puede concatenar con S_2. Se denota la concatenación de S_1 y S_2 como $S_1||\,S_2$.

Superposición de una cadena de señalización a un camino IGP: Dada una cadena de señalización $S{:}A(=R_0),\ R_1,R_2,..R_r,B(=R_{r+1})$ y un camino IGP P de A a B, si para $i=1\ldots r+1,\ R_i\in P$, entonces decimos que S superpone a P, o que P es superpuesto por S.

Cadena de señalización de camino más corto: Para una cadena de señalización $S{:}A(=R_0),\ R_1,R_2,..R_r,B(=R_{r+1})$, si existe un camino más corto IGP P de A a B de tal manera que S superpone a P, entonces decimos que S es una cadena de señalización de camino más corto entre A y B.

Entonces mediante el uso del siguiente lema:

Lema: Para cualquier destino d, si existe una cadena de señalización de camino más corto entre el router A y $Egress_d(A)$, entonces A aprende la mejor ruta a través de $Egress_d(A)$, para el destino d.

Se puede obtener el siguiente teorema:

Teorema: Una configuración iBGP garantiza que la propiedad de visibilidad completa será satisfecha a la hora de enfrentar cambios arbitrarios en la topología IGP si, y solo si, para cualquier camino IGP de un router BGP con otro router eBGP, existe una cadena de señalización que se superpone a este camino.

El algoritmo BGPSepS

Uno de los objetivos de diseño de BGPSepS es encontrar una configuración iBGP que garantice la visibilidad completa sin cambios en la topología IGP, evitando así los bucles y caminos sub-óptimos. De acuerdo a la sección anterior, se tiene que encontrar una configuración iBGP tal que haya una señalización de camino más corto desde un router a cualquiera de sus posibles egresos.

En un grafo IGP, para un vértice u cuyo grado es uno, supongamos que su vértice v adyacente tiene visibilidad completa. Entonces, siguiendo las reglas de reflexión de rutas y las de selección de ruta de BGP, el vértice u tendrá visibilidad completa si u es cliente de v.

Además, si en el grafo IGP podemos encontrar un separador de grafo, un conjunto de vértices cuya eliminación particiona al grafo en un número de componentes conexas, entonces cualquier camino más corto que comience en un componente y que termine en un componente diferente debe pasar a través de uno o más routers en el separador. Si se construye una topología iBGP utilizando una configuración de full mesh entre los routers del separador, utilizando también una configuración de full mesh dentro de cada componente conexa y creando otras sesiones iBGP necesarias de manera que exista una señalización de camino más corto entre cualquier router en un componente y cualquier otro router en el separador, entonces existirá una cadena de señalización de camino más corto entre dos vértices cualquiera.

Si se quita uno o más vértices de los componentes y se agregan al conjunto de vértices del grafo separador, entonces se obtiene un superconjunto del grafo separador, que sigue siendo un grafo separador. Obviamente, los grados máximos de las topologías iBGP generadas sobre la base de diferentes separadores pueden ser diferentes. Esperamos que podamos encontrar un grafo separador óptimo de tal manera que el grado máximo de la topología iBGP generada sea mínimo. Sin embargo, es una tarea muy difícil encontrar un grafo separador óptimo para un grafo IGP grande en la práctica. Entonces, después de encontrar un separador se utiliza un método heurístico para encontrar un superconjunto del mismo. En primer lugar, buscamos un camino más corto desde cualquier router en los componentes hacia cualquier otro router en el separador. A continuación añadimos los vértices de esta ruta en el superconjunto, excepto el vértice inicial.

Descripción del algoritmo

Además de los procedimientos utilizados por BGPSep, este algoritmo utiliza un procedimiento llamado $Shortest - Path$ para encontrar el camino más corto entre dos vértices.

A continuación se presenta el pseudocódigo del algoritmo:

BGPSep_S
Entrada: Grafo IGP G , conjunto V de routers BGP
Salida: Conjunto I de sesiones iBGP
/* Paso 1: removiendo gradualmente los vértices colgantes*/
$I = \emptyset$;
pending = **true**;
$G = G'$;
while pending == **true do**
$G = G'$;
pending = **false**;

```
        foreach u ∈ G.V do
                if d_g(u) == 1 then /* d_g(u) es el grado de 'u' en G */
                        v = adj_g(u) ; /* 'v' es el nodo adyacente de 'u' */
                        I = I ∪ {(u, v, cliente)} ;  /* 'u' es cliente de 'v' */
                        G' = G' − {u};
                        pending = true;
                end
        end
end
/* Paso 2: Elegir un grafo separador S ⊂ G'.V */
S = Graph − Separator(G');
G_1,..,G_m ← componentes de G'.V − S;
/* Paso 3: Encontrar un superconjunto S+ de S */
S+ = S;
foreach u ∈ G_i, v ∈ S do
        if u ∉ S+ then
                P = Shortest − Path( u, v );
                foreach w ∈ P do
                        if w ≠ u then
                                S+ = S+ ∪ {w};
                        end
                end
        end
end
/* Paso 4: Hacer un full mesh con los routers en S+ */
foreach u, v ∈ S+, u ≠ v do
        I = I ∪ {(u, v, par)} ;
end
/* Paso 5: Hacer que cada router en G'.V − S sea reflector de rutas de algún router en S+ */
foreach  u ∈ G'.V − S+, v ∈ S+ do
        P: u(= R_0), R_1, R_2,.. R_r, v(= R_{r+1}) ← Shortest − Path(u, v);
        i = 1;
        while R_i ∉ S+ do
                i++;
        end
        I = I ∪ {(u, v, cliente)} ;
end
/* Paso 6: Hacer un full mesh con los routers en G_i − S+ */
foreach u, v ∈ G_i − S+ do
        I = I ∪ {(u, v, par)} ;
end
return  I;
```

4.3 BatesX

La heurística de Bates recomienda que para configurar una topología iBGP con uno o múltiples RRs pertenecientes a un mismo Point of Presence (PoP, la ubicación física donde están los dispositivos de red que permiten el acceso a Inernet) en la red, todos los routers en el PoP (que no son RRs) deben ser clientes de todos los RRs del PoP. A su vez, los RRs deben estar configurados en full mesh. Finalmente, se recomienda establecer un full mesh entre todos los routers del PoP que no son RR. Es una heurística de uso habitual por parte de los operadores de red.

Se implementaron dos variantes:

Bates Y:

Se toma el router más conectado (el de mayor grado) de cada PoP de la red y se establece a éste como el único RR del PoP. Siguiendo las recomendaciones de Bates, todo router restante en cada PoP se conecta como cliente del RR de su PoP. Finalmente se establece un full mesh entre todos los routers (menos el RR) en cada PoP.

Bates Z:

Es similar a Bates Y con la diferencia que se toman los dos routers más conectado de cada PoP de la red y se establecen a éstos como los RRs del PoP. Todo router restante en cada PoP se conecta como cliente a éstos dos RR. Finalmente, se agrega una sesión entre los dos RR de cada PoP (no requiere establecer un full mesh entre los clientes de cada PoP).

4.4 Zhang

La heurística de Zhang consiste en diseñar una configuración iBGP con herencia, donde los RRs del nivel *n* son clientes de los RRs del nivel *n-1*. Se definen a lo sumo dos o tres niveles.

Explicaremos la heurística para dos niveles (es similar para tres). Primero, todos los routers en el nivel más bajo (que no son RR) son conectados como clientes de los dos RRs de su nivel más cercanos a él, según la métrica de IGP. Segundo, los RRs del nivel más bajo se conectan como clientes de los dos RRs del nivel más alto más cercanos a él, también según la métrica de IGP. Finalmente, se establece un full mesh entre todos los RRs del nivel más alto. Esto debe realizarse para cada PoP.

Esta heurística, al igual que la de Bates, es de uso habitual por parte de los operadores de red.

4.5 Full mesh

Una topología iBGP de full mesh consiste en, como ya fue mencionado anteriormente, en una configuración en la cual existe una sesión iBGP entre todo par de routers del AS. Se implementó un algoritmo que genera una topología iBGP de este tipo principalmente con dos objetivos: utilizarlo en la comparación de algoritmos y verificar que el estado al que convergen los otros algoritmos implementados es el correcto.

[5]

Prueba de concepto

A modo de prueba de concepto hemos incluido un ejemplo de ejecución de la herramienta resultado de integrar a TOTEM nuevas funcionalidades referentes al diseño, desarrollo y prueba de topologías iBGP. Puede consultar la sección 8 por detalles de cómo realizar las ejecuciones que presentamos en esta sección.

Para realizar la prueba de concepto utilizaremos varias topologías IGP para analizar la utilización de reflectores de rutas sobre la topología iBGP resultante de la ejecución de BGP sobre dicha topología.

Nos colocaremos en el lugar de un ISP que desea utilizar reflectores de rutas para mitigar los problemas de escala causados por el gran tamaño de las tablas y los problemas de gestión presentes en topologías iBGP de full mesh. Para cada una de las topologías, y pensando en la topología de nuestro AS, ejecutamos un subconjunto de los algoritmos que hemos desarrollado para obtener una topología iBGP que utilice reflectores de rutas. Luego, compararemos las topologías resultado de cada uno de los algoritmos por medio del comportamiento que obtenemos al ejecutar BGP utilizando esta topología.

Es decir, luego de ejecutar un algoritmo y obtener una topología iBGP, utilizando un escenario que consiste básicamente en los mensajes BGP que son recibidos por cada uno de los routers de borde y por medio del script C-BGP generado por nuestra herramienta simularemos el intercambio de mensajes que ocurre en la ejecución del protocolo.

Luego del desarrollo de las pruebas diseñadas para cada una de las topologías que utilizamos para presentar este ejemplo contamos con una gran cantidad de datos que pueden ayudarnos a definir algún criterio de elección.

En resumen, mostraremos como a partir desde una topología IGP que es típicamente conocida y utilizando el conjunto de las nuevas funcionalidades incluidas a TOTEM es posible el diseño, desarrollo y prueba de topologías iBGP que utilicen reflectores de rutas. Estas son las principales características de la herramienta resultado de integrar a TOTEM nuevas funcionalidades, quedando fuera de esta prueba de conceptos las facilidades provistas para el desarrollo de nuevos algoritmos (por detalles puede consultarse las sección 3).

Comencemos a delimitar las pruebas que hemos realizado. Hemos utilizado un subconjunto de los algoritmos que implementamos para realizar esta prueba de concepto. Los algoritmos que utilizamos son BGPSepD, BGPSepS, Óptimo Buob y a modo de prueba de control y para utilizar como referencia Fullmesh.

Puede notarse que no utilizamos BGPSep ni ninguna de las heurísticas para esta prueba. BGPSep fue excluido del conjunto de algoritmos que utilizamos debido a que BGPSepD está completamente basado en él. La única modificación que realiza es la eliminación de los nodos de grado 1 de la topología IGP para posteriormente ejecutar BGPSep.

Por otra parte, las heurísticas no apuntan a resolver el problema en topologías genéricas y es necesario cierto criterio para elegir que heurística utilizar. No tiene sentido simplemente ejecutar sobre un conjunto arbitrario de topologías. En otras palabras, ayudan a resolver el problema en ciertos casos particulares, y no es útil aplicarlas cuando sabemos que no van a obtener buenos resultados.

Luego de definido el conjunto de algoritmos para los cuales creemos que podemos obtener buenos resultados sobre nuestra topología (recuerde que estamos utilizando un conjunto de topologías) solo resta definir el escenario que se utilizará para simular la ejecución del protocolo BGP que nos permita contar con un conjunto de datos que ayuden a decidir cuál de las topologías es más adecuada para nuestras necesidades.

Antes de mencionar como hemos creado los escenarios de ejecución debemos detenernos en el conjunto de topologías que utilizamos. La elección de la topología es una decisión fundamental a la hora de evaluar algoritmos que construyan topologías iBGP a partir de la misma. No estamos realizando una evaluación de los algoritmos, simplemente presentamos una prueba de concepto de la herramienta resultado de integrar las nueva funcionalidades a TOTEM.

Crear topologías realistas no es una tarea fácil, para esta tarea podemos utilizar IGen [22]. Es necesario tener en cuenta ciertas medidas que ayuden a caracterizar la topología. Podemos mencionar algunas de las medidas que son descriptas en [25]:

- **Distribución de la distancia**: La primera métrica que mencionaremos es la distribución de las distancias entre un par de nodos a través del camino más corto. Esta métrica puede ser considerada un indicador del delay requerido para transmitir paquetes entre estos nodos si asumimos que la mayor parte del delay es causado por la transmisión en los enlaces. En particular, el **diámetro de una red** es el largo del camino más corto más largo. Se calcula la distribución de la distancia calculando la distribución del largo de los caminos más cortos para cada par de nodos en la red.

- **Diversidad de caminos**: Para medir la cantidad de redundancia ofrecida por la red se utiliza la diversidad de caminos. Esta métrica mide la disponibilidad de diversos caminos entre pares de nodos. La disponibilidad de diversos caminos es importante en términos de robustez e ingeniería de tráfico. Es posible computar la diversidad de caminos de un grafo calculando, para cada par de nodos, la cantidad de caminos nodo disjuntos que existen entre ellos.

- **Conectividad**: Otro camino para medir la redundancia de una red es calculando la k-arista conectividad de la red [25]. Esta métrica proporciona el tamaño del corte mínimo en la red. En comparación con la diversidad de caminos descripta anteriormente la conectividad sólo proporciona una idea de la cantidad de caminos nodos disjuntos para todos los partes de nodos, mientras que la diversidad de caminos lo hace de forma individual para cada par de nodos.

- **Grado de los nodos**: La distribución del grado de los nodos es frecuentemente una de las métricas utilizadas para evaluar las topologías. Es comúnmente aceptado que las redes tienen una pequeña cantidad de nodos con un grado alto (en el backbone de la red) y una gran cantidad de nodos con bajo grado (los nodos de acceso).

- **Costo**: Finalmente, una importante restricción operacional es el límite de dinero disponible para construir la red. Sin embargo, es muy dificultoso definir una métrica de costo. El costo de una red se torna dificultoso de evaluar debido a que depende de la tecnología utilizada para los enlaces y los router, el ancho de banda utilizado y el largo de los enlaces. Sin embargo es posible obtener una idea de algunos componentes del costo de una red.

Con estas métricas definidas sobre una red debemos construir topologías siendo capaces de controlarlas. IGen no es capaz de generar topologías que cumplan con requerimientos establecidos por estas métricas. Por otra parte, medir cada uno de estos valores puede no ser trivial, sin embargo es posible utilizar BRIANA para obtener algunos de estos valores. BRIANA es una herramienta incluida en el proyecto BRITE [11].

Aún más, en caso de contar con una herramienta de generación de topologías que permita generarlas a partir de requerimientos sobre estas métricas (o similares): ¿cuáles son los valores que deben establecerse para que la topología generada sea realista?. Esta dificultad se debe principalmente a que los ISP mantienen, generalmente, sus topologías en secreto.

Por otra parte, luego de generada la topología es fundamental para algunos de los algoritmos y posteriormente para la simulación la ubicación de los routers de borde de la red.

Debido a estas dificultades decidimos utilizar un conjunto de topologías que se encuentran públicas por tratarse de topologías de redes académicas (como es el caso de RNP), topologías antiguas (como es el caso de ABILENE) o algunos ISP que han hecho públicas parte de su topología.

Las topologías que utilizamos para realizar esta prueba de concepto son:

- ABILENE: Red ubicada en EEUU. Es ampliamente utilizada en publicaciones académicas.
- CESNet: Red universitaria ubicada en República Checa.
- COX: Red ubicada en EEUU de la proveedora de servicios de red COX Communications.
- CWIX: Red ubicada en EEUU.
- FORTHNet: Red ubicada en Grecia de la proveedora de servicios de red Forthnet.
- GARR: Red universitaria ubicada en Italia.
- ISTAR: Red ubicada en EEUU y parte de Canadá.
- RNP: Red académica ubicada en Brasil.

A continuación presentamos una tabla que intenta caracterizar las topologías. Para cada topología se muestra la cantidad de nodos, la cantidad de enlaces, la cantidad de routers de borde, el grado promedio de los nodos, el grado máximo, el grado mínimo y por último la varianza del grado.

Topología	# Nodos	# Aristas	# RB	Grado (avg)	Grado (máx)	Grado (mín)	Grado (var)
ABILENE	12	60	12	5	8	2	2,5454545455
CESNet	47	232	3	4,0701754386	29	1	25,31641604
COX	33	252	4	7,2	20	1	27,341176471
CWIX	24	116	3	4,8333333333	10	2	3,7971014493
FORTHNet	60	236	3	3,9333333333	38	2	34,436158192
GARR	47	224	8	2,3092783505	28	1	13,299183849
ISTAR	19	76	2	4	10	2	8,4444444444
RNP	28	132	1	4,7142857143	10	2	3,9153439153

Figura 5.1: Tabla de caracterización de topologías

Volvamos a nuestro rol de ISP. Ahora contamos con un conjunto de algoritmos que pensamos tienen buenas posibilidades de obtener buenos resultados sobre nuestra topología (en el contexto de esta prueba de concepto contamos con un conjunto de topologías).

Necesitamos definir un escenario para nuestra topología (nuevamente pensando en nuestro rol de ISP, en el contexto de esta prueba de concepto es necesario definir al menos un escenario para cada topología). Un escenario consiste en los mensajes eBGP que son recibidos por cada uno de los routers de borde del Sistema Autónomo. Para definir esto utilizamos escenarios TRA, un formato de archivo que hemos definido para este propósito. Por detalles al respecto puede consultar la sección 8.

Sólo resta definir que mensajes son recibidos por los routers de borde del sistema autónomo. Los mensajes BGP deben estar en formato MRT [RFC 6396], los mensajes pueden ser extraídos de los archivos de trazas disponibles públicamente, por ejemplo utilizando RouteViews [20] y no es necesario realizar ninguna modificación a ellos.

El proyecto RouteViews fue concebido como una herramienta para que los operadores de red pudieran obtener información en tiempo real sobre el sistema de enrutamiento global desde la perspectiva de varias redes troncales y diferentes lugares de todo Internet.

Los escenarios que utilizamos para esta prueba de concepto se construyeron agregando para cada uno de los routers de borde del Sistema Autónomo 10 mensajes BGP de UPDATE. La mitad de estos mensajes, es decir 5, son repetidos en todos los routers de borde del Sistema Autónomo. Recuerde los ejemplos que hemos desarrollado a lo largo del documento: los problemas aparecen cuando tenemos dos puntos de egreso diferentes para el mismo prefijo.

Luego de realizada la ejecución de los algoritmos utilizando las herramientas que facilitan esto (consulte la sección 8) podemos obtener una gran cantidad de datos que pueden ayudar a optar por una topología iBGP sobre otra en función de las necesidades del ISP.

Mencionaremos ahora algunos de los datos más relevantes que pueden ser recabados luego de la ejecución de la herramienta resultado de integrar a TOTEM nuevas funcionalidades referentes al diseño, desarrollo y prueba de topologías iBGP. Se encuentran disponibles la cantidad de mensajes utilizados por BGP hasta que el protocolo converge, la cantidad de sesiones iBGP utilizadas por la topología iBGP, las tablas de routing de cada uno de los routers, las RIB para cada uno de los routers y las RIB-In y RIB-Out de cada uno de los vecinos de cada router.

Creemos necesario mencionar brevemente en que consiste la RIB. La Routing Information Base (RIB) en un router BGP consiste de tres partes distintas:

1. RIB-In: Mantiene información de ruteo aprendida desde los mensajes UPDATE entrantes que son recibidos por el router BGP. Su contenido representa las rutas que están disponibles como entrada del proceso de decisión de BGP.
2. Loc-RIB: Contiene la información de ruteo local del router BGP que ha sido seleccionada al aplicar las políticas locales a la información de ruteo contenida en Adj-RIBs-In. Estas son las rutas que van a ser utilizada por el router BGP.
3. RIB-Out: Mantiene la información seleccionada por el router BGP para ser anunciada a sus pares.

En resumen, las RIB-In contienen la información de ruteo que fue anunciada al router BGP por sus pares y que no ha sido procesada; la Loc-RIB contiene las rutas que han sido seleccionadas por el proceso de decisión BGP del router; y las RIB-Out organiza las rutas que van a ser anunciadas a un par específico (por medio de mensajes UPDATE).

La información de ruteo BGP que es utilizada por parte del router para el intercambio de paquetes (o para construir la tabla que es utilizada para el intercambio de paquetes) es mantenida en la Tabla de Routing. La Tabla de Routing acumula las rutas hacia las redes directamente conectadas, las rutas estáticas, las rutas aprendidas por los protocolos IGP, y las rutas aprendidas desde BGP. Sí una ruta BGP específica debe ser instalada en la Tabla de Routing, y si una ruta BGP debe sobrescribir una ruta hacia el mismo destino que otra ruta instalada por otro protocolo es una decisión dependiente de las políticas locales. La Tabla de Routing, además del intercambio de paquetes, es utilizada para la resolución de la dirección de los next hops especificados en los mensajes UPDATE.

Realizada esta breve aclaración veamos los datos obtenidos luego de la ejecución de los distintos algoritmos sobre el conjunto de topologías que mencionamos anteriormente. A continuación se puede encontrar una tabla por algoritmo con los resultados obtenidos.

Topology	MSGs	Ses	RT (avg)	RIB (avg)	RT (var)	RIB (var)	RT (max)	RIB (max)	RT (min)	RIB (min)
ABILENE	4104	32	77	65	0	0	77	65	77	65
CESNet	5363	109	66,065217391	20	0,0623188406	0	67	20	66	20
COX	11533	159	57,125	25	0,1129032258	0	58	25	57	25
CWIX	4935	90	43,130434783	20	0,1185770751	0	44	20	43	20
FORTNet	1008	59	74,033898305	15	0,0333138515	0	75	15	74	15
GARR	15577	106	91,152173913	45	0,131884058	0	92	45	91	45
ISTAR	353	19	32,111111111	14	0,1045751634	0	33	14	32	14
RNP	3142	96	37,037037037	10	0,037037037	0	38	10	37	10

Figura 5.2: Resultados obtenidos por BGPSepD

Topology	MSGs	Ses	RT (avg)	RIB (avg)	RT (var)	RIB (var)	RT (max)	RIB (max)	RT (min)	RIB (min)
ABILENE	3832	36	65	0	0	0	77	65	77	65
CESNet	5691	113	66,065217391	20	0,0623188406	0	67	20	66	20
COX	13801	163	57,125	25	0,1129032258	0	58	25	57	25
CWIX	5456	133	43,130434783	20	0,1185770751	0	44	20	43	20
FORTNet	1008	59	74,033898305	15	0,0333138515	0	75	15	74	15
GARR	25180	160	91,152173913	45	0,131884058	0	92	45	91	45
ISTAR	353	19	32,111111111	14	0,1045751634	0	33	14	32	14
RNP	2602	136	37,037037037	10	0,037037037	0	38	10	37	10

Figura 5.3: Resultados obtenidos por BGPSepS

Topology	MSGs	Ses	RT (avg)	RIB (avg)	RT (var)	RIB (var)	RT (max)	RIB (max)	RT (min)	RIB (min)
ABILENE	14673	44	77	65	0	0	77	65	77	65
CESNet	2297	61	65,2	19,13	17,18	17	67	20	46	0
CWIX	2091	38	43,13	20	0,12	0	44	20	43	20
FORTNet	1496	63	74,03	15	0,03	0	75	15	74	15
GARR	24954	132	91,17	45	0,15	0	92	45	91	45
ISTAR	312	20	29,78	11,67	29,48	28,82	33	14	18	0
RNP	324	27	37,04	10	0,04	0	38	10	37	10

Figura 5.4: Resultados obtenidos por Óptimo Buob

Topology	MSGs	Ses	RT (avg)	RIB (avg)	RT (var)	RIB (var)	RT (max)	RIB (max)	RT (min)	RIB (min)
ABILENE	1452	66	77	65	0	0	77	65	77	65
CESNet	3542	1081	66,065217391	20	0,0623188406	0	67	20	66	20
COX	2336	0	57,125	25	0,1129032258	0	58	25	57	25
CWIX	1242	276	43,130434783	20	0,1185770751	0	44	20	43	20
FORTNet	4720	1770	74,033898305	15	0,0333138515	0	75	15	74	15
GARR	5842	1081	91,152173913	45	0,131884058	0	92	45	91	45
ISTAR	698	171	32,111111111	14	0,1045751634	0	33	14	32	14
RNP	1026	378	37,037037037	10	0,037037037	0	38	10	37	10

Figura 5.5: Resultados obtenidos por FullMesh

Si bien no es el objetivo de esta sección, incluimos a continuación algunos de los datos más relevantes presentados en las tablas anteriores en forma de gráficas para permitir una mejor visualización de los resultados obtenidos.

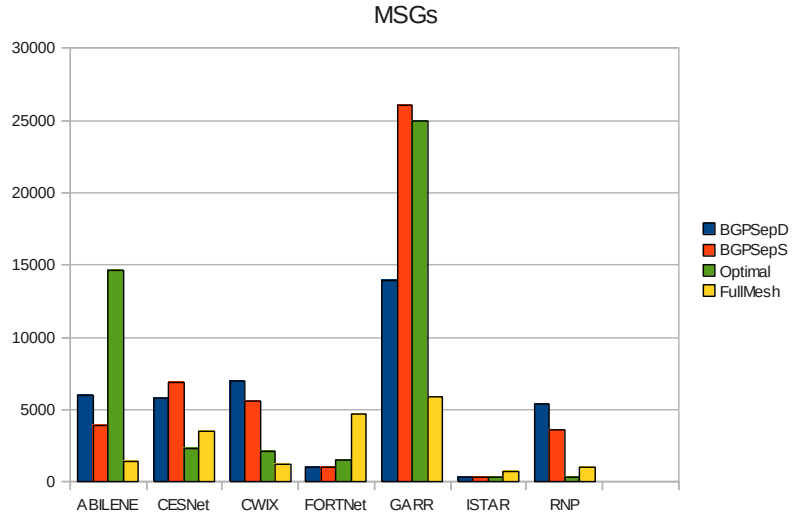


Figura 5.6: Gráfica comparativa de cantidad de mensajes

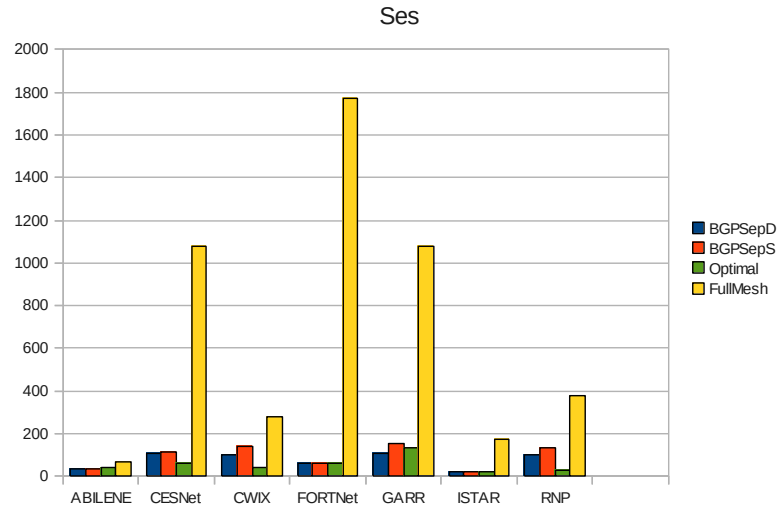


Figura 5.7: Gráfica comparativa de cantidad de sesiones

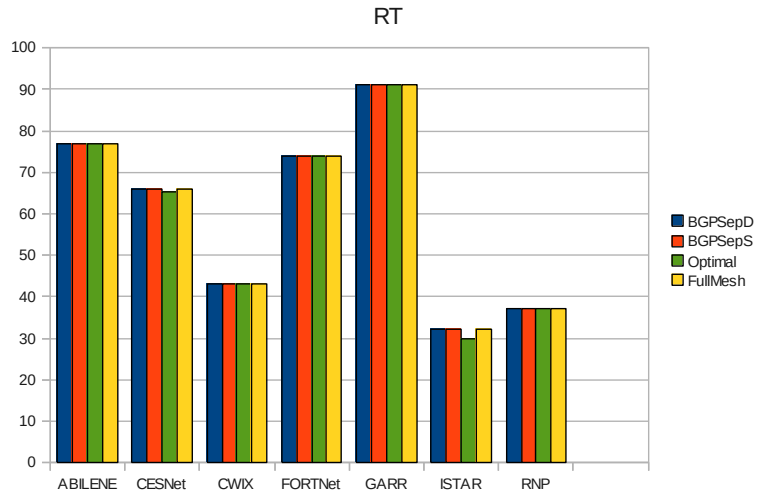


Figura 5.8: Gráfica comparativa del tamaño de las Tablas de Routing

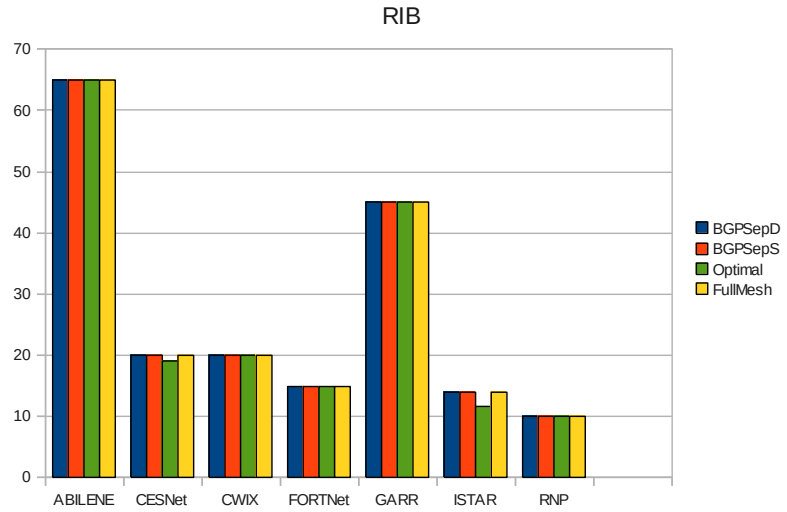


Figura 5.9: Gráfica comparativa del tamaño de las RIBs

Comentemos brevemente los resultados. Primero nótese que el tamaño de las tablas no se ve afectado por la topología iBGP.

Por otra parte, la disminución en la cantidad de sesiones iBGP utilizadas por la topología tiene su costo en cantidad de mensajes intercambiados por BGP hasta que este converge. Sin embargo, podemos notar que en el caso de FORTHNet e ISTAR este costo no existe. Es decir, se disminuye la cantidad de sesiones iBGP de la topología y por esto no se ven incrementados los mensajes transmitidos por BGP hasta que este converge. Por el contrario, la cantidad de mensajes se ve decrementada.

Podemos notar que estas dos topologías obtienen una topología iBGP que se acerca a la topología IGP, es decir, no existen sesiones iBGP que no estén implementadas por medio de un único enlace IGP. Veamos por ejemplo el caso de FORTHNet, a continuación se puede visualizar la topología resultado de aplicar BGPSepD junto con la propia topología. Para detalles de las convenciones utilizadas para representar topologías iBGP se puede consultar la sección 8.

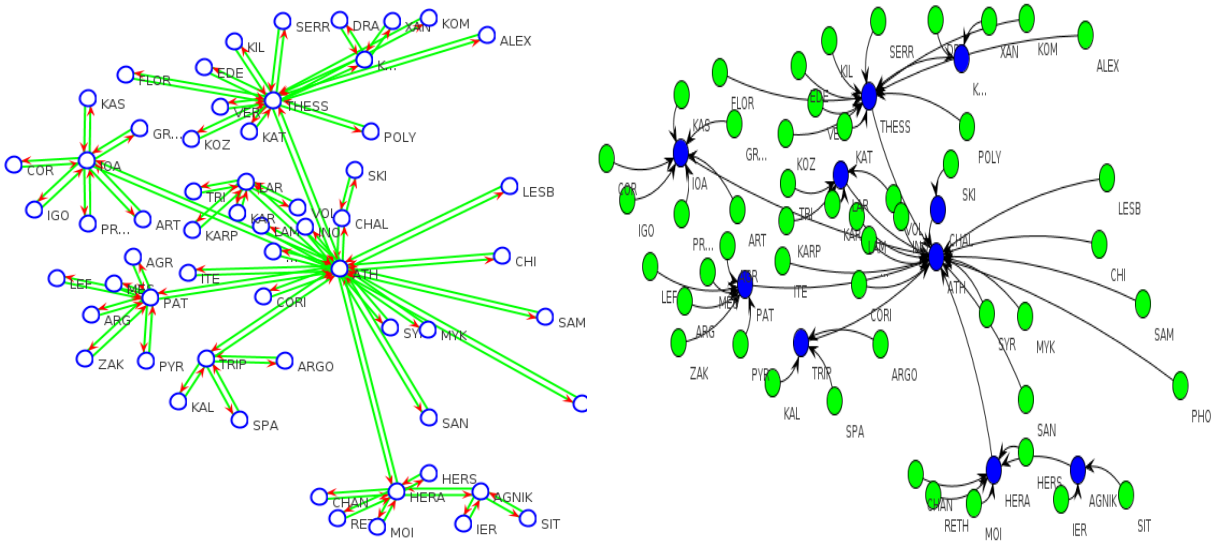


Figura 5.10: Resultado de aplicación de BGPSepD sobre la red FORTHNet

En cambio, en topologías para las cuales es necesario implementar sesiones iBGP por medio de más de un enlace IGP el costo de decrementar la cantidad de sesiones utilizadas por la topología es un significativo aumento en la cantidad de mensajes utilizados por BGP para converger. A continuación se puede visualizar ABILENE y la topología resultado de aplicar BGPSepD.

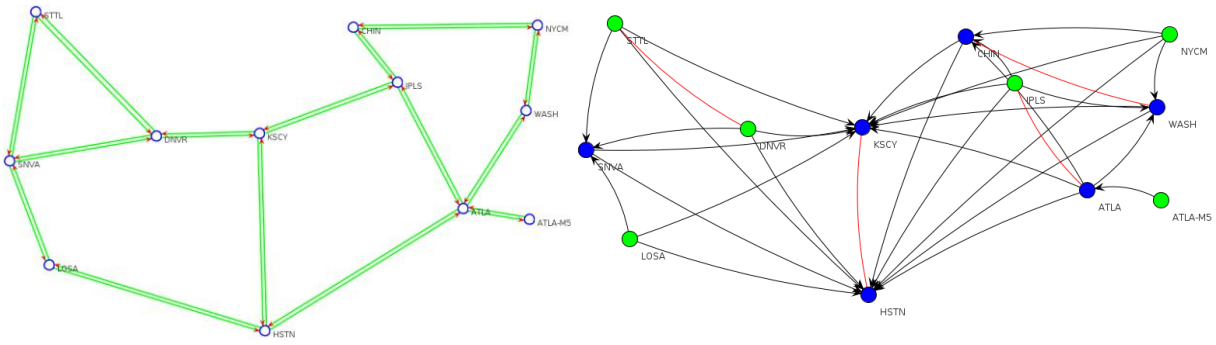


Figura 5.11: Resultado de aplicación de BGPSepD sobre la red Abilene

[6]

Conclusiones y trabajo futuro

6.1 Resultados alcanzados

El principal resultado del proyecto fue la implementación de una herramienta que tiene como objetivo facilitar la implementación de nuevos algoritmos que resuelvan el problema de localización de reflectores de rutas dentro de un AS, así también como su evaluación y simulación con herramientas ya existentes como C-BGP. A su vez se implementaron varios algoritmos que intentan resolver el problema entre los que se incluyen el algoritmo óptimo de Buob y la familia BGPSep. También se implementaron en la herramienta dos heurísticas muy utilizadas por los operadores de red en la actualidad: Bates y Zhang, con sus variantes.

El conjunto de herramientas implementadas agregadas al framework TOTEM proporcionan un entorno de trabajo completo para un usuario que desee implementar, evaluar y simular un algoritmo que resuelva el problema. Esto es de principal importancia ya que no existe una herramienta con similares características a la fecha. También, se generó un manual de usuario que acompaña a este informe en un anexo para facilitar el uso de las herramientas implementadas.

6.2 Dificultades encontradas

La principal dificultad encontrada en el proyecto fue el hecho de que para comprender el problema fue necesario asimilar una gran cantidad de conocimiento. El problema es complejo e involucra una curva de aprendizaje alta al comienzo de su estudio para estudiantes sin experiencia en el área. Es necesaria la lectura y comprensión de documentos que si bien no son extensos en muchos casos son difíciles de comprender en una primera lectura.

También, al tratarse de un tema demasiado amplio y un proyecto de investigación nos encontramos muchas veces utilizando recursos o enfocando nuestro esfuerzo hacia conceptos o técnicas que no fueron utilizadas finalmente.

Otra dificultad encontrada fue con la implementación del algoritmo óptimo de Buob. Este algoritmo no está completamente definido en el documento que lo describe, algunas características necesarias para la implementación (y no triviales) no están cubiertas en el documento. Esto complicó en un principio la implementación. Otro problema pero de índole técnico fue debido a que este algoritmo se plantea como un problema de programación entera (MIP), siendo necesario un solver de MIP para implementarlo. En una primera aproximación se implementó el algoritmo basado en el solver Choco [17], pero no cumplió con nuestras expectativas. Por lo tanto, se solicitó a la Facultad una licencia del solver de IBM llamado CPLEX y se realizó la implementación en base al mismo, obteniendo muchos mejores resultados.

En cuanto a TOTEM, es una herramienta desarrollada hace un tiempo y ha perdido soporte por parte de sus desarrolladores. Nos vimos en la necesidad de utilizar una versión reciente de la biblioteca Jung [9], pero TOTEM no es compatible con dicha versión. En otras palabras, Jung perdió compatibilidad hacia atrás. Para lograr incluir esta nueva versión debimos renombrar los paquetes, compilar e incluir esta nueva versión. Esto debió realizarse para todas las bibliotecas que incluimos y utilizan Jung (por ejemplo Agape [21]).

6.3 Posibles extensiones

Como posibles extensiones al proyecto mencionamos las siguientes:

Como hemos mencionado anteriormente el objetivo de minimizar la cantidad de sesiones que utiliza la topología iBGP parece no tener beneficios claros. Las tablas de routing no se ven significativamente afectadas y no está claro el efecto que esta disminución tiene sobre la cantidad de mensajes intercambiados por el protocolo BGP hasta que este converge. Sin embargo, basamos estos comentarios en los datos que hemos recogido con estas pruebas, que posiblemente sean insuficientes y es necesario realizar un estudio más profundo.

Por otra parte, generalmente este tipo de problemas que poseen una formulación intuitiva simple pero su formulación formal es compleja son buenos candidatos a ser resueltos con buenos resultados por medio de metaheuristícas. Parece interesante intentar aplicar GRASP para solucionar el problema global, sin embargo la aplicación de un Algoritmo Evolutivo no es tan clara por la dificultad de encontrar operadores de mutación y cruzamiento a ser utilizados.

Finalmente, con el conocimiento generado tal vez no esté tan lejana la posibilidad de obtener resultados concluyentes, desarrollando un algoritmo que resuelva el problema de mejor manera que los existentes.

[7]

Referencias

Referencias

1. RFC reflectores de rutas. En línea: https://tools.ietf.org/rfc/rfc4456.txt [Última visita: 25/3/2013]
2. Página principal de TOTEM. En línea: http://totem.info.ucl.ac.be/ [Última visita: 25/3/2013]
3. Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong, "The Stable Paths Problem and Interdomain Routing". IEEE/ACM Transactions on Networking, 2002.
4. M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of NP-Completeness". W.H. Freeman and Co., San Francisco, CA, 1979.
5. Marc-Olivier Buob, Steve Uhlig and Mickael Meulle, "Designing optimal iBGP route-reflection topologies". In proceedings NETWORKING 2008.
6. Vutukuru, M., Valiant, P., Kopparty, S., Balakrishnan, H., "How to Construct a Correct and Scalable iBGP Configuration". In proceedings INFOCOM 2006. 25th IEEE International Conference on Computer Communications.
7. Feng Zhao, Xicheng Lu, Peidong Zhu y Jinjing Zhao, "BGPSep_D: An Improved Algorithm for Constructing Correct and Scalable IBGP Configurations Based on Vertexes Degree". Second International Conference, HPCC 2006, Munich, Germany, September 13-15, 2006.
8. Feng Zhao, Xicheng Lu, Peidong Zhu y Jinjing Zhao, "BGPSep_S: An Algorithm for Constructing IBGP Configurations with Complete Visibility". 8th International Conference, ICDCN 2006, Guwahati, India, December 27-30, 2006.
9. Página principal de JUNG. En línea: http://jung.sourceforge.net [Última visita: 25/3/2013]
10. Página principal de C-BGP. En línea: http://c-bgp.sourceforge.net [Última visita: 25/3/2013]
11. Página principal de BRITE. En línea: http://www.cs.bu.edu/brite [Última visita: 25/3/2013]
12. Página principal de NS-2. En línea: http://nsnam.isi.edu/nsnam/index.php/Main_Page [Última visita: 25/3/2013]
13. Página principal de GT-ITM. En línea: http://www.cc.gatech.edu/projects/gtitm [Última visita: 25/3/2013]
14. Página principal de CPLEX. En línea: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer [Última visita: 25/3/2013]

15. D. Ashlock, "Evolutionary Computation for Modeling and Optimization". Springer Science Business Media, Inc, 2006.
16. Fred Glover, Gary A. Kochenberge, "Handbook of Metaheuristics". Kluwer Academic Publishers, 2003.
17. Página principal de Choco. En línea: http://www.emn.fr/z-info/choco-solver [Última visita: 25/3/2013]
18. Griffin, Timothy & Wilfong, Gordon, "On the correctness of IBGP configuration". SIGCOMM Comput. Commun. Rev. 2002 Volume 32 Number 4 Pages 17-29. http://doi.acm.org/10.1145/964725.633028
19. Stefano Vissicchio, Luca Cittadini, Laurent Vanbever y Olivier Bonaventure, "iBGP Deceptions: More Sessions, Fewer Routes". INFOCOM, 2012 Proceedings IEEE.
20. Route Views Project. En línea: http://www.routeviews.org [Última visita: 25/3/2013]
21. Página principal de Agape. En línea: http://traclifo.univ-orleans.fr/Agape [Última visita: 25/3/2013]
22. Página principal de IGen. En línea: http://informatique.umons.ac.be/networks/igen [Última visita: 25/3/2013]
23. M. Buob, M. Meulle, and S. Uhlig, "Checking for optimal egress points in iBGP routing". Proc. of the 6th IEEE International Workshop on the Design of Reliable Communication Networks (DRCN 2007), October 2007.
24. G. Leduc, H. Abrahamsson, S. Balon, S. Bessler, M. D'Arienzo, O. Delcourt, J. Domingo-Pascual, S. Cerav-Erbas, I. Gojmerac, X. Masip, A. Pescapé, B. Quoitin, S.P. Romano, E. Salvadori, F. Skivéé, H.T.Tran, S. Uhlig, H. Umit, "An Open Source Traffic Engineering Toolbox". 18 May 2005.
25. M. Stoer and F. Wagner, "A simple min-cut algorithm". Journal of the ACM (JACM), July 1997.

Tabla de figuras

Número/s de figura	Referencia
2.1 y 2.2	[6]
2.3 a 2.13	[18]
2.14 a 2.17	[19]
3.1 y 3.2	[24]
4.1 a 4.4	[5]
4.5 a 4.17	Original
5.1 a 5.13	Original
5.14 a 5.75	Original
8.1 a 8.21	Original
9.1	Original

[8]

Manual de usuario del framework

En este manual se describen las funcionalidades nativas utilizadas así como todas aquellas funcionalidades extras de TOTEM implementadas en este proyecto. Entre las funcionalidades implementadas tenemos: **a) ejecución independiente de la interfaz**, **b) exportación a un script C-BGP**, **c) asignación automática de direcciones IP**. Este manual no pretende ser exhaustivo, sólo da una introducción a algunas de las funcionalidades que provee TOTEM, muchas de las cuales están fuera del alcance de este proyecto. Si requiere de más información puede dirigirse al sitio web de TOTEM: http://totem.info.ucl.ac.be/.

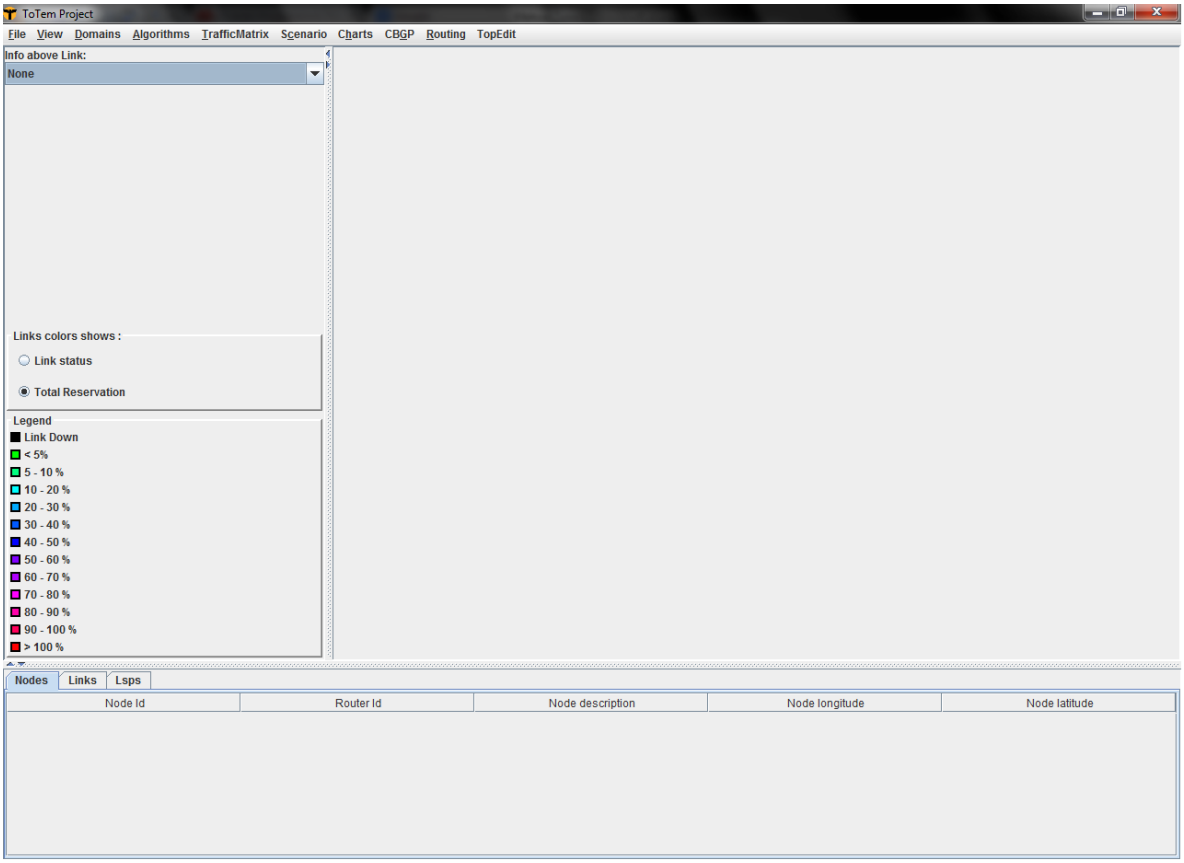


Figura 8.1: Pantalla principal de TOTEM

Contenido

1. Cargar una topología

Para cargar una topología, desde la pantalla principal se debe acceder al menú ***File → Load Topology.***

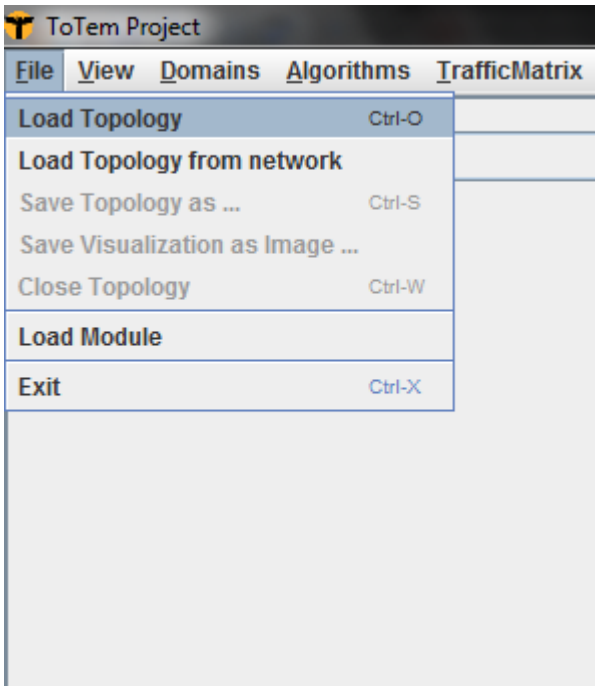


Figura 8.2: Menú "File → Load Topology"

Esto lanzará una ventana ***Load topology***, desde ahí se debe indicar la ruta del archivo que se desea cargar.

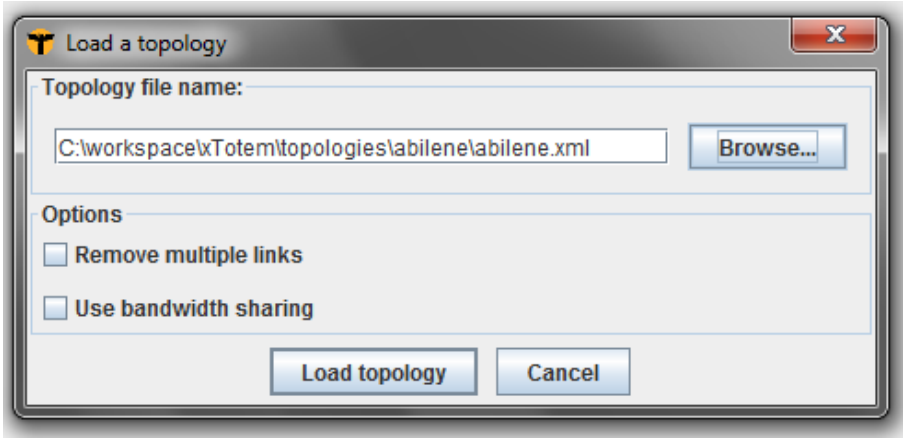


Figura 8.3: Ventana "Load a Topology"

Para seleccionar el archivo se puede abrir un selector de archivos clickeando sobre el botón ***Browse…*** o escribir manualmente la ruta en el campo de texto. Finalmente, se debe clickear sobre el botón ***Load topology*** para efectuar la carga, o presionar el botón ***Cancel*** para cancelar.

Una vez que la topología sea cargada será mostrada en la pantalla principal. La topología mostrada en la pantalla principal es denominada como la topología por defecto.

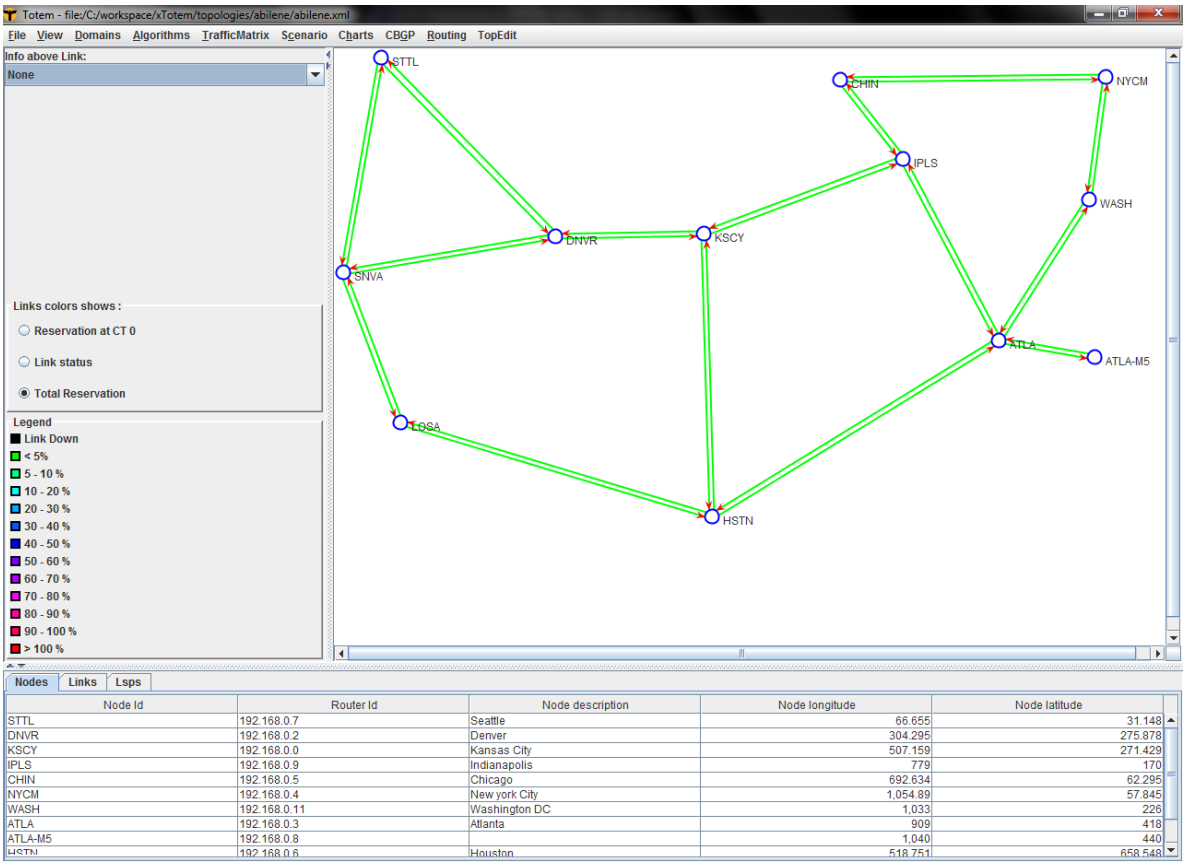


Figura 8.4: Visualizador de dominios de TOTEM

2. Manejo de topologías

TOTEM permite la carga de múltiples topologías al mismo tiempo, por lo tanto será de ayuda poder gestionar las topologías abiertas. En el menú ***Domains*** se podrán visualizar todas las topologías abiertas actualmente. Seleccionando cada topología se podrá cambiar la topología por defecto de una forma rápida y sencilla.

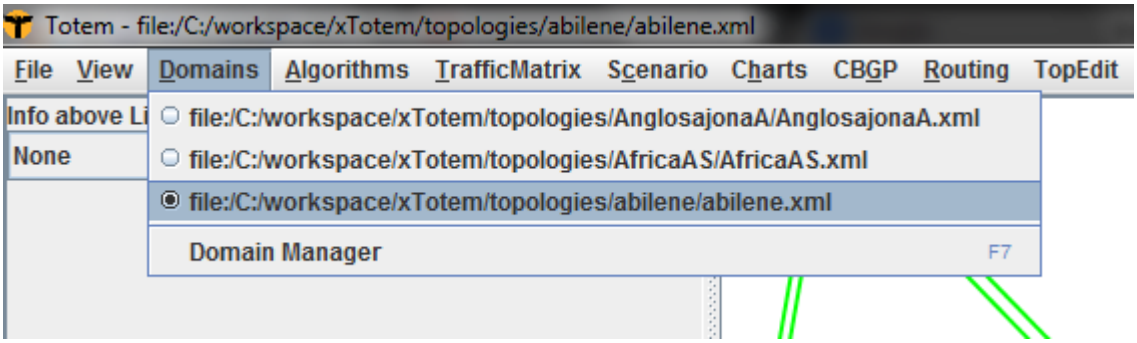


Figura 8.5: Menú "Domains"

Además TOTEM cuenta con un Manager de Dominios (o topologías) el cual permite navegar por las topologías cargadas y a su vez permite cerrarlas. Este manager puede ser accedido en el menú ***Domains*** → ***Domain Manger***. Seleccionando dicha opción se lanzará la ventana ***Domains currently loaded***.

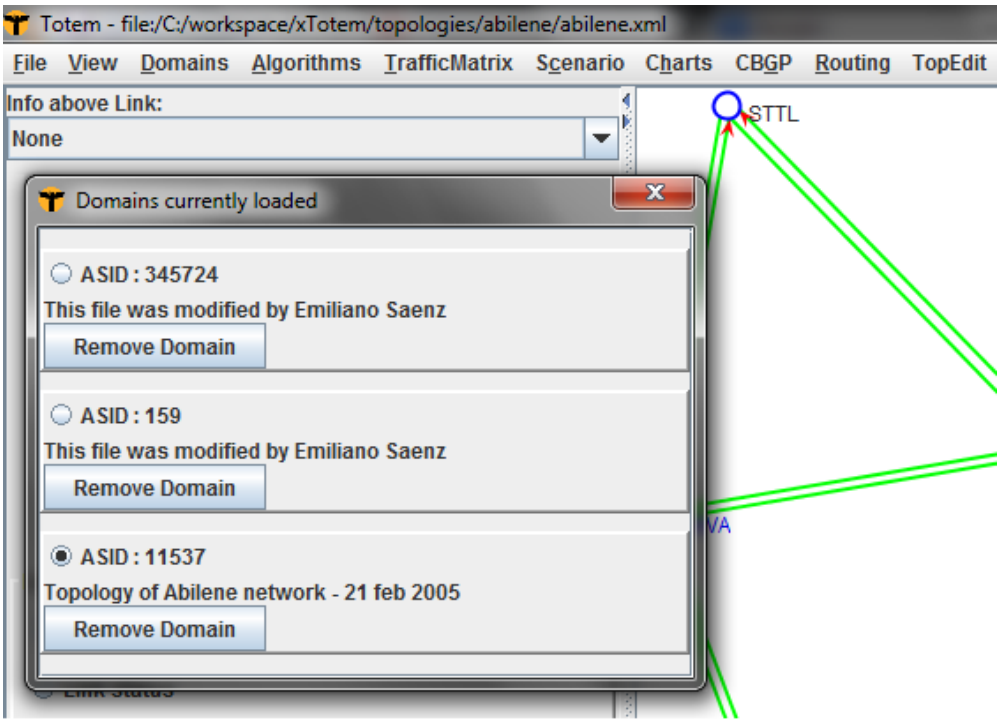


Figura 8.6: Ventana "Domains currently loaded"

3. Ejecutar un algoritmo

Una vez que se ha efectuado la carga de topologías será posible ejecutar los algoritmos disponibles en TOTEM sobre las mismas. Para ejecutar los algoritmos se debe acceder al menú ***Algorithms*** → ***Start***.

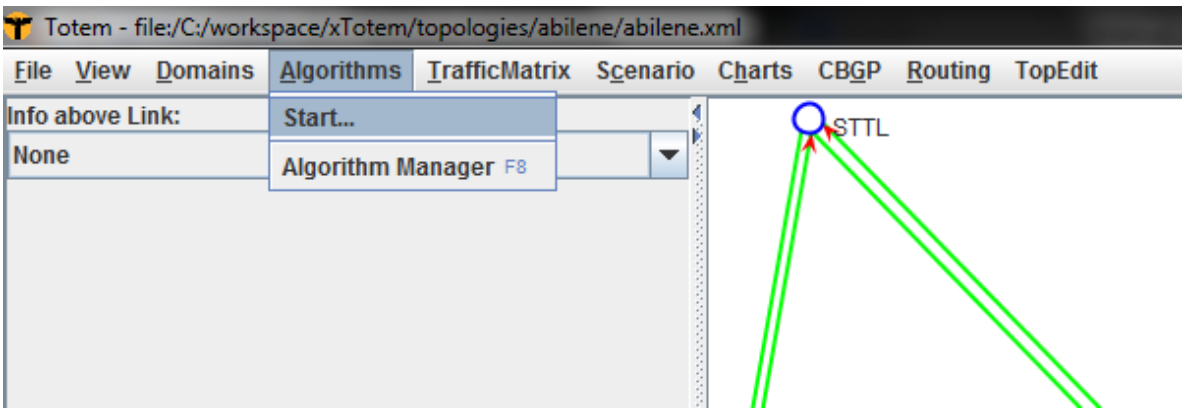


Figura 8.7: Menú "Algorithms → Start"

La elección de esta opción ejecutará la ventana ***Start Algorithms***, desde la cual se podrá elegir qué algoritmo ejecutar y se podrá elegir con que parámetros será ejecutado el mismo.

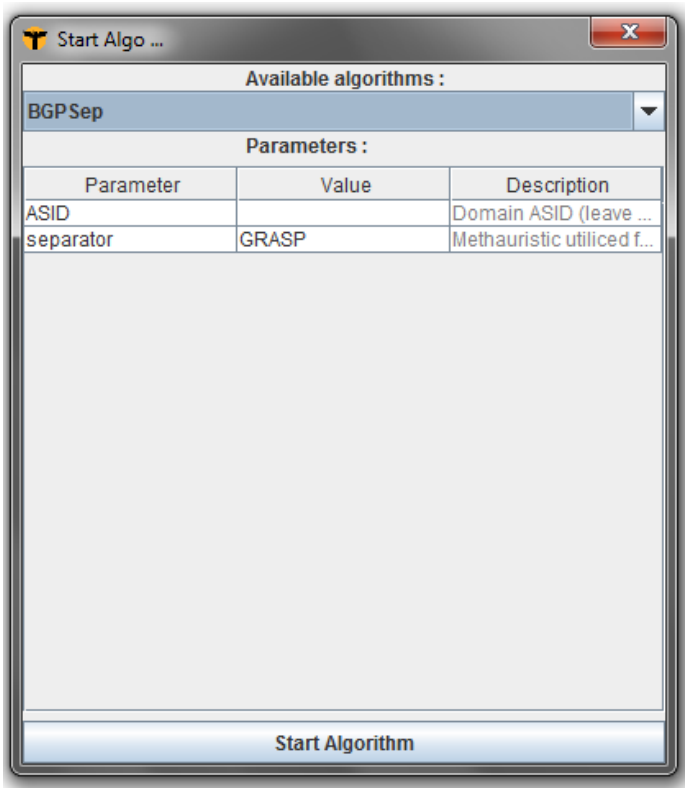


Figura 8.8: Ventana "Start Algorithms"

Para seleccionar el algoritmo se debe clickear sobre la lista de opciones, lo que desplegará la lista de todos los algoritmos disponibles para ejecutar.

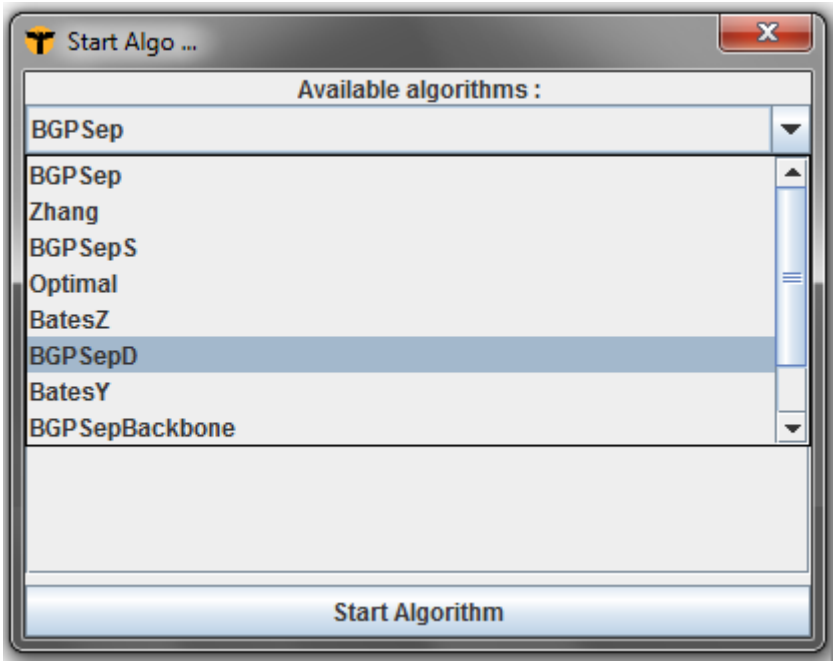


Figura 8.9: Ventana "Start Algorithms"

Una vez elegido el algoritmo se mostrarán en una tabla los parámetros del mismo. Los tipos de parámetros pedidos por los algoritmos son muy diversos desde enteros, archivos o lista de archivos.

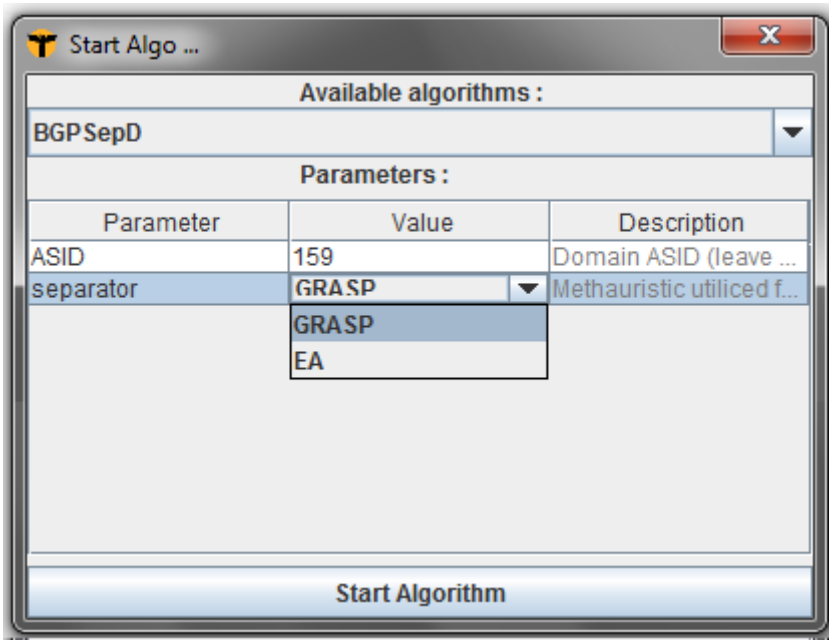


Figura 8.10: Ventana "Start Algorithms"

Una vez que se asignó un valor a cada parámetro para ejecutar el algoritmo se debe clickear sobre el botón ***Start Algorithm***. Dependiendo del algoritmo puede ocurrir que la interfaz quede bloqueada hasta que finaliza su ejecución o no. En nuestro caso, todo los algoritmos que implementamos se ejecutan en background, es decir una vez que se clickea sobre el botón ***Start Algorithm***, no se bloqueará la interfaz permitiendo que se sigan realizando otra tareas en TOTEM, incluso ejecutar otros algoritmos. Cuando el algoritmo que está ejecutando en backgroung termine lanzará una ventana informativa indicando la finalización de la ejecución. Una vez que finaliza la ejecución los resultados del algoritmo son impactados inmediatamente en el dominio donde se ejecutó.

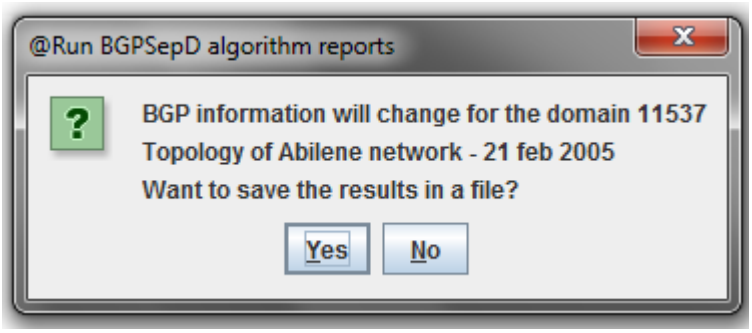


Figura 8.11: Ventana "Algorithm reports"

Esta ventana informativa preguntará si se desea impactar los resultados del algoritmo sobre un archivo. En caso afirmativo se lanzará un selector de archivos y deberá elegirse una ubicación y nombre de archivo de salida. Guardar los resultados del algoritmo permite que éstos no sean sobrescritos por la ejecución de otro algoritmo.

4. Visualizador iBGP

Para visualizar la configuración iBGP de la topología por defecto se debe ir al menú ***View*** → ***Sessions iBGP***.

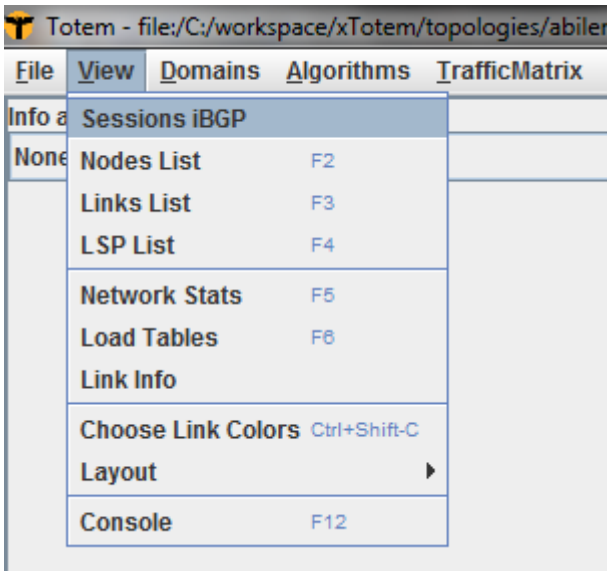


Figura 8.12: Menú "View Sessions iBGP"

Esto lanzará una nueva ventana que mostrará la configuración iBGP actual de la topología cargada. Los nodos azules representan a los reflectores de rutas y los verdes a los routers BGP clásicos. Las aristas rojas representan una sesión peer y las negras dirigidas una sesión cliente. Si u apunta a v $(u \to v)$ entonces u es cliente de v.

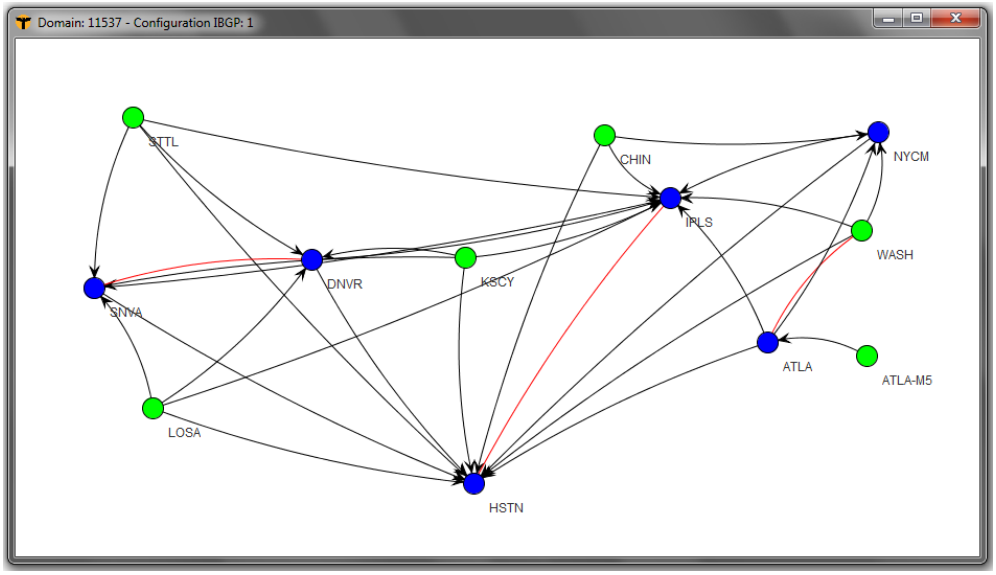


Figura 8.13: Menú "View Sessions iBGP"

5. Exportar topología como un script C-BGP

Para generar un script C-BGP dada una topología se debe acceder al menú ***CBGP*** → ***Export as C-BGP script***.

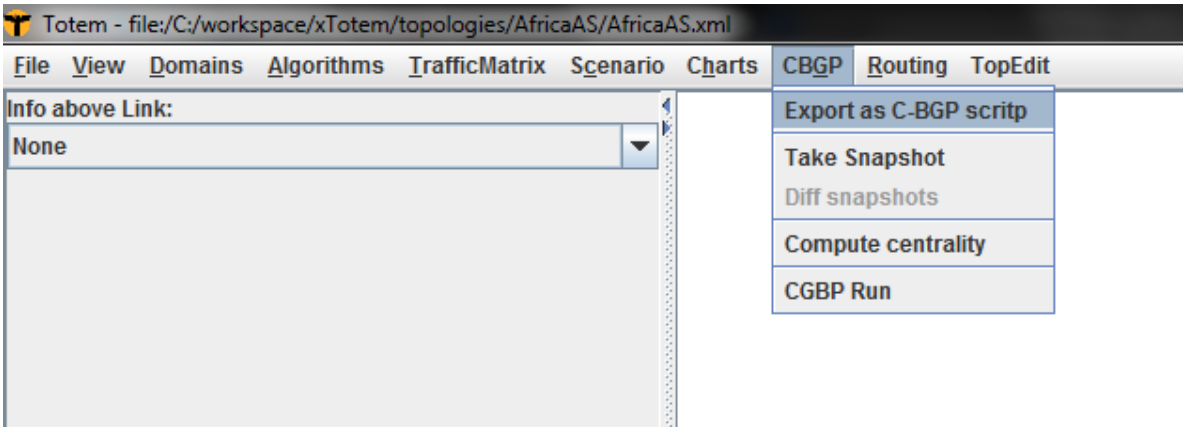


Figura 8.14: Menu "CBGP → Export as C-BGP script"

Esta acción lanzará la ventana ***Export C-BGP***, donde se deberá indicar qué archivo de topología (.xml) y que escenario (.tra) se desea impactar.

Hemos definido un formato simple para generar escenarios de prueba de forma rápida. Estos escenarios son complementados con la topología iBGP generada luego de aplicar alguno de los algoritmos que desarrollamos y genera un entorno de simulación completo. Es necesario definir para cada uno de los routers de borde los mensajes BGP que este recibirá. Los mensajes BGP deben estar en formato MRT. Cada sección de mensajes, para cada uno de los routers de borde, se debe especificar la IP del router y una IP que debe ser única y no estar utilizada para otro fin (sugerimos utilizar direcciones privadas). A continuación podemos ver un ejemplo:

```
[1.1.1.1-11.11.11.11]
BGP4|1359575700|W|193.160.39.1|57821|78.136.196.0/22
BGP4|1359575700|W|193.160.39.1|57821|78.136.192.0/22
BGP4|1359575700|W|193.160.39.1|57821|79.136.234.0/23
BGP4|1359575700|W|193.160.39.1|57821|88.204.108.0/23
[2.2.2.2-22.22.22.22]
BGP4|1359575702|W|176.12.110.8|50300|78.140.46.0/24
BGP4|1359575702|W|176.12.110.8|50300|78.140.34.0/24
BGP4|1359575702|W|176.12.110.8|50300|83.172.44.0/23
BGP4|1359575702|W|176.12.110.8|50300|88.204.58.0/24
BGP4|1359575702|W|176.12.110.8|50300|88.204.50.0/23
BGP4|1359575702|W|176.12.110.8|50300|88.204.54.0/23
BGP4|1359575702|W|176.12.110.8|50300|88.204.98.0/24
```

Es importante destacar que los mensajes son extraídos de los archivos de trazas públicamente disponibles, por ejemplo en [20] y no es necesario realizar ninguna modificación a ellos.

Mediante los botones ***Browse...*** se podrán ubicar ambos archivos, o simplemente se pueden escribir las direcciones en los cuadros de texto.

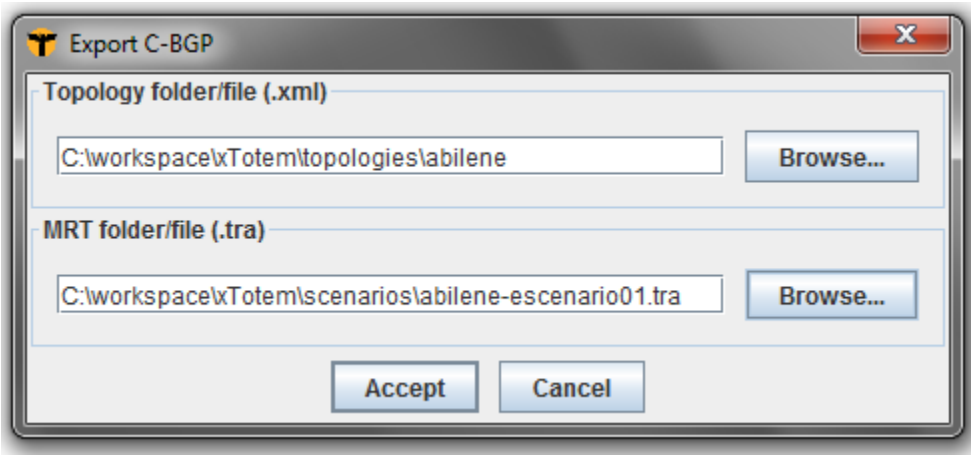


Figura 8.15: Ventana "Export C-BGP"

Una vez que el script se haya generado correctamente se desplegará la siguiente ventana informativa.

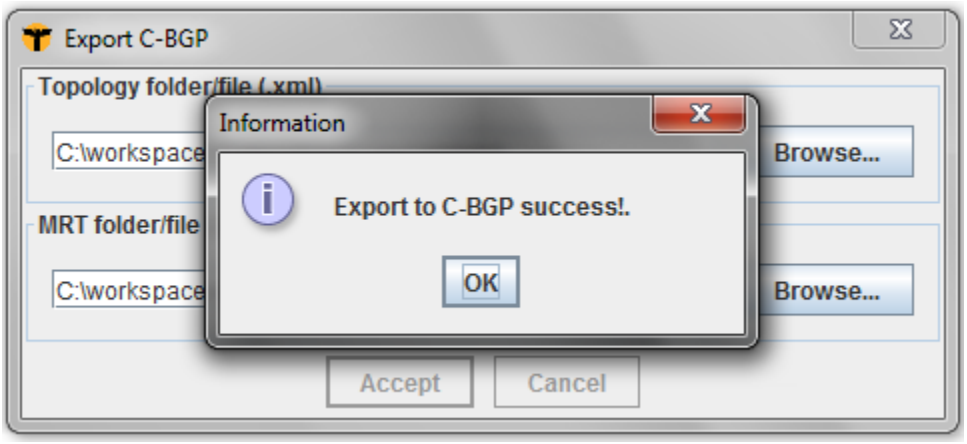


Figura 8.16: Ventana "Information"

6. Editar una topología

Desde TOTEM es posible editar las topologías accediendo al menú ***TopEdit***. Desde allí se podrán crear nuevas topologías (***New***), usar generadores de topologías (***TopGen***), o modificar topologías existentes (***Edit***).

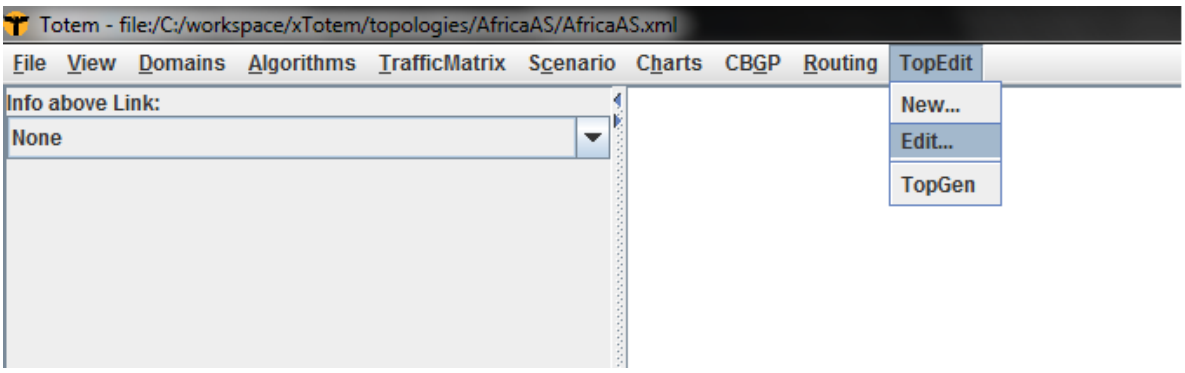


Figura 8.17: Menú "TopEdit"

En caso de elegir alguna de las acciones anteriores se lanzará la ventana del editor de topologías.

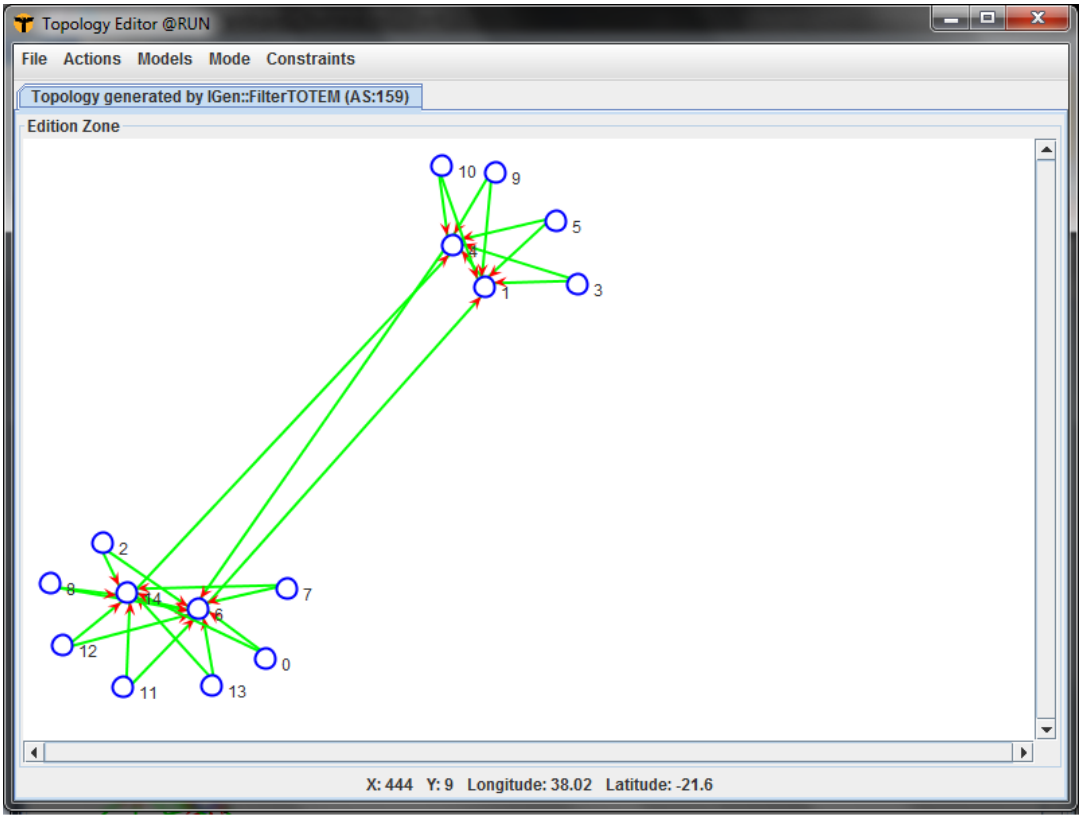


Figura 8.18: Ventana "Topology Editor"

Dentro del menú ***Actions*** del editor encontramos la opción ***Numbering network***, que nos permite asignar números de IP a los routers de la red. Esta funcionalidad fue implementada y agregada al editor, ya que en muchos casos nos vimos con la dificultad de que las topologías que usamos no tenían números de IP asignados correctamente, y es muy común cometer errores al asignar las IP si se hace manualmente. Esta funcionalidad nos permite asignar de forma confiable y automática las IP a los routers. La asignación incorrecta de IP puede ocasionar problemas por ejemplo en las simulaciones.

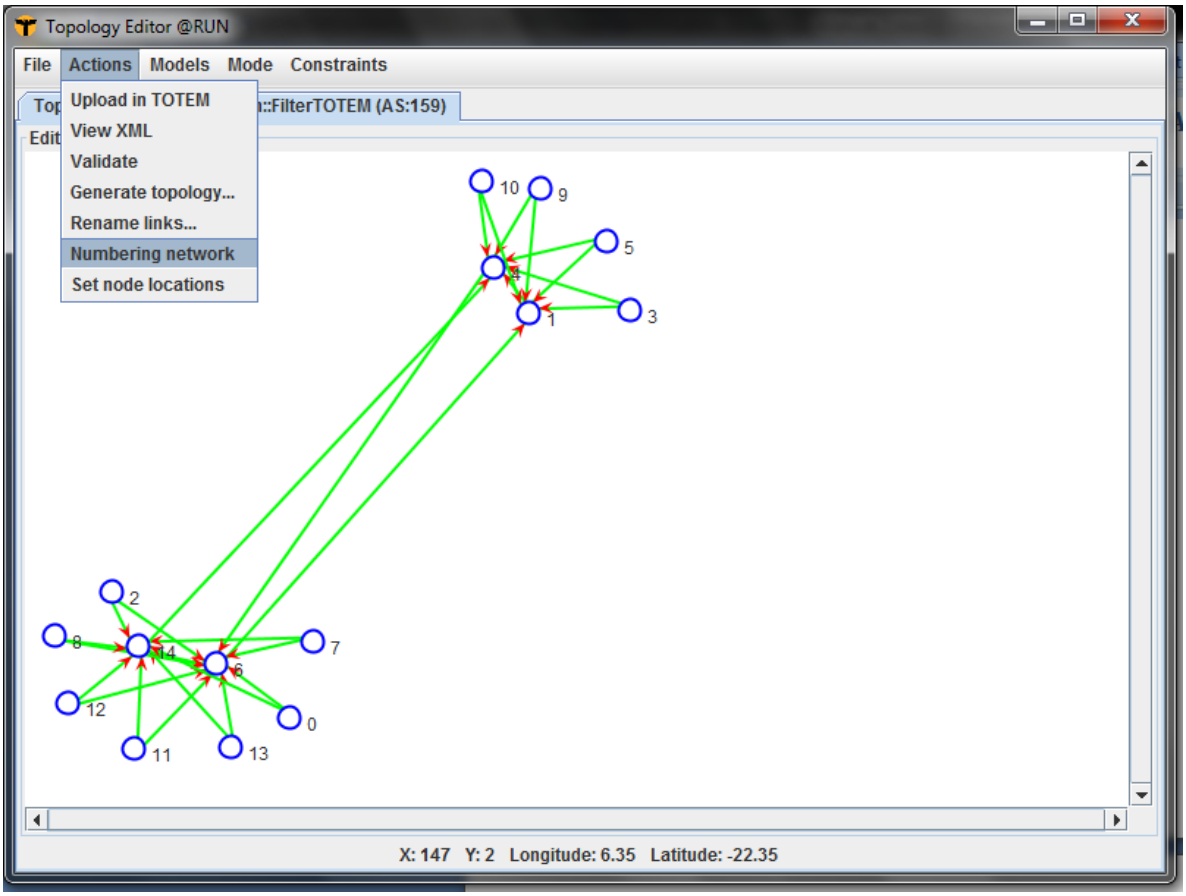


Figura 8.19: Menú "Actions → Numbering network"

7. Guardar una topología

Para guardar topologías se debe acceder al menú ***File→Save Topology as***.

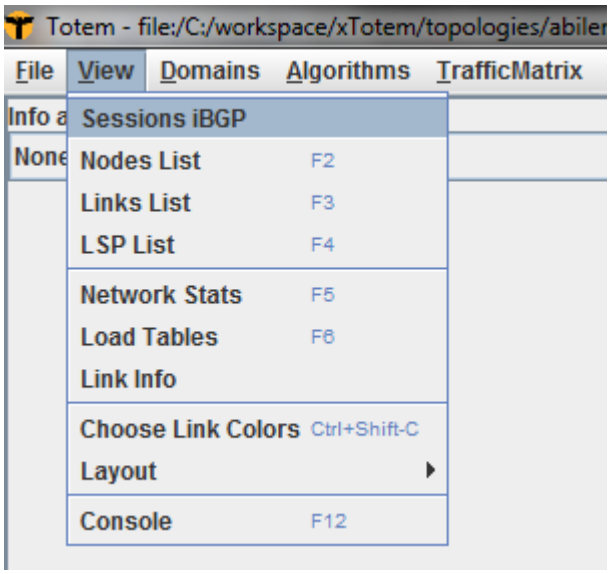


Figura 8.20: Menú "File → Save Topology As"

Esta acción lanzará un selector de archivos dónde se debe indicar la ubicación y el nombre del archivo que se desea guardar.

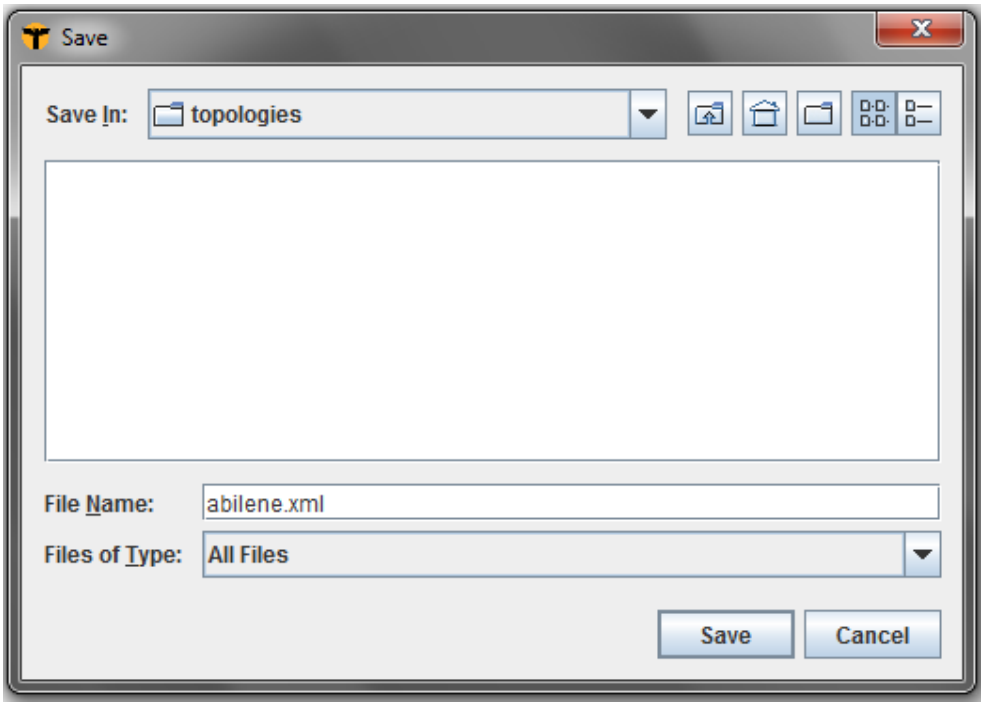


Figura 8.21: Menú "Save"

8. Ejecución independiente de la interfaz

Además de poder utilizar TOTEM por medio de la interfaz gráfica proveemos un conjunto de scripts que pueden ser utilizados para ejecutar por medio de línea de comandos todas las funcionalidades necesarias para realizar el estudio sobre una configuración iBGP. Esto es especialmente útil cuando se desea ejecutar un conjunto de algoritmos sobre una topología, o un algoritmo sobre un conjunto de topologías o cualquier otra combinación que pueda ser útil. Al intentar realizar varias ejecuciones es normal que el tiempo necesario sea demasiado y es deseable contar con la posibilidad de mantener el proceso en background incluso luego de un deslogueo (por ejemplo utilizando el comando nohup)

Comencemos con el script **totem.sh**. Este script simplifica la ejecución de TOTEM por medio de líneas de comando simplemente estableciendo algunas variables de entorno necesarias para la ejecución y actuando como intermediario pasando los parámetros recibidos a TOTEM. Para obtener detalles de los posibles parámetros que puede utilizar simplemente ejecute:

:~ ./totem.sh

Por último hablemos del script que permite ejecuciones masivas: **runner.sh**. Por medio de este script podrá ejecutar el conjunto de algoritmos de desee sobre el conjunto de topologías y escenarios de prueba que sean necesarios. Además deberá proporcionar un script que se encargará de procesar los resultados y obtener, por ejemplo, una planilla CSV como resultado.

Las salidas luego de la ejecución de este script corresponden a la topología resultado (el archivo .xml), el escenario de simulación en C-BGP (el archivo .cli), el archivo que contiene todas las tablas de routing, las RIB in y RIB out para todos los routers de la topología (el archivo .bgp) y por último el archivo que contiene todos los mensajes intercambiados hasta que BGP converge. Estos archivos son generados para cada una de las combinaciones de algoritmos, topologías y parámetros que usted haya decidido ejecutar.

Además de estos archivos de salida es posible obtener como resultado algunas estadísticas generadas a partir de estos archivos resultados por medio de un parser que es recibido como parámetro que es llamado luego de la ejecución de una de las configuraciones a ejecutar. Esto es útil debido a que para ejecuciones grandes es normal que la cantidad de datos aumente rápidamente.

Veamos ahora una descripción del comando:

./runner.sh totem_runner_script parser_script topologies_and_tras_path [params_path] flags

El primero de los parámetros, totem_runner_script, es el script que se utilizará para ejecutar TOTEM (totem.sh). El segundo parámetros corresponde al script que se ejecutará luego de que se obtienen los archivos temporales, es el encargado de obtener las estadísticas deseadas a partir de estos archivos. Proporcionamos dos script desarrollados en Perl que permite contar la cantidad de mensajes intercambiados y los tamaños de las tablas y generar a partir de esto un archivo CSV de salida. Además proporcionamos otro script que cuenta la cantidad de sesiones iBGP en las topologías resultados.

Luego de estos dos scripts es necesario especificar un directorio, topologies_and_tras_path, en este directorio se deben encontrar todas las topologías para las cuales se desea ejecutar los algoritmos que sean posteriormente especificados. Adicionalmente, y en caso de no especificar la flag -ns es necesario que para cada una de las topologías que se encuentren en este directorio se proporcione un archivo .tra (el formato de especificación de escenarios que ya fue explicado) conteniendo el escenario de simulación.

Opcionalmente se puede proporcionar el archivo que contiene las diferentes combinaciones de parámetros que se quieren utilizar. Por ejemplo:

```
-nb_run 5 -n_gen 20000 -sizeP 100 -sizeOf 60 -pmut 0.01 -pcross 0.05
-nb_run 5 -n_gen 20000 -sizeP 100 -sizeOf 60 -pmut 0.07 -pcross 0.05
-nb_run 5 -n_gen 20000 -sizeP 100 -sizeOf 60 -pmut 0.14 -pcross 0.05
-nb_run 5 -n_gen 20000 -sizeP 100 -sizeOf 60 -pmut 0.20 -pcross 0.05
```

Simplemente restan especificar las posibles flags. Puede utilizar -vv o -vvv para incrementar el nivel de mensajes que son mostrados por el script. Con -vv solo verá los mensajes emitidos por la salida de error, mientras que con -vvv verá todos los mensajes. Por defecto no se muestra salida, sólo el porcentaje de trabajo realizado.

Por defecto todos los archivos de salida, el .xml, el .cli, el .bgp y el .msg son borrados y solo se mantienen los resultados generados por el parser a partir de estos archivos temporales. Si desea que no sean eliminados puede utilizar la flag -f para obtener este resultado.

Es posible que en alguna ocasiones no desee realizar las simulaciones, por esto la flag -ns permite realizar las ejecuciones sin simular luego de obtener la topología iBGP.

Luego puede especificar los algoritmos que desee ejecutar utilizando -bgpsep, -bgpsepB, -bgpsepD, -bgpsepS, -fullmesh, -optimal, -zhang, -bates, -batesY y -batesZ.

[9]

Instrucciones de instalación y ejecución del framework

Contenido

1. Introducción

En esta sección se presentan las instrucciones necesarias para compilar y ejecutar el framework. En esta primera parte se presenta la ubicación del código fuente, estructura del mismo y entorno de ejecución requerido. En la segunda y tercera se dan instrucciones de cómo compilar de dos maneras diferentes y de cómo ejecutar. Finalmente en la cuarta se proporcionan links al software de terceros requerido.

Ubicación del código fuente

El código fuente se encuentra en el siguiente repositorio SVN público:

```
http://fingpgrr2012.googlecode.com/svn/xTotem/
```

Se puede realizar el checkout del proyecto mediante el siguiente comando (no es necesaria la autenticación):

```
svn checkout http://fingpgrr2012.googlecode.com/svn/xTotem/
```

También pueden utilizarse diversos clientes SVN para realizar el checkout. Consulte el manual de su cliente SVN para más detalles.

Estructura de directorio

Se adjunta en la carpeta del proyecto (Totem) todo el contenido y recursos utilizados y/o generados. A continuación se detalla la estructura de directorio.

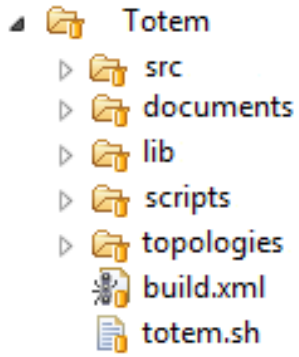


Figura 9.1: Estructura de directorio utilizado en el proyecto.

Totem/build.xml: archivo de compilación.
Totem/totem.sh: script de ejecución.

Totem/src: carpeta en la que se encuentra todo el código fuente generado.
Totem/documents: carpeta en la que se encuentran los documentos auxiliares generados, por ejemplo los documentos resultados de las etapas de calibración.
Totem/scripts: carpeta en la que se encuentran todos los scripts generados, por ejemplo los utilizados para pasear resultados de las etapas de calibración.
Totem/topologies: carpeta en la que se encuentran todas las topologías utilizadas para testing, calibración y evaluación.
Totem/lib: carpeta en la que se encuentran todas las librerías utilizadas en el proyecto.

Entorno de ejecución

El framework ejecuta en un entorno LINUX de 32 bits (probado en Ubuntu 10.04 y 12.10, CentOS 5.5 y Debian Squeeze). Se requiere Java JDK en su versión 1.6 o 1.7 (para algunas updates de la 1.6 el proyecto no compila). La instalación de **IBM ILOG CPLEX Optimizer** (al final de la sección se proporciona un link de descarga) es opcional, en caso de hacerlo utilice la versión v12.2 o superior.

2. ¿Cómo compilar?

En la carpeta ***Totem/*** se encuentra el archivo ***build.xml***. Para compilar el proyecto ubicado en la carpeta ***Totem/*** ejecute el comando **ant**. Existen dos formas de compilación: incluyendo las librerías de **CPLEX** y no incluyéndolas.

Sin CPLEX:

En caso que no contar con una licencia de CPLEX, se permite compilar (y posteriormente ejecutar) el proyecto excluyendo de la compilación el algoritmo **Optimo Buob**. Este algoritmo utiliza la herramienta CPLEX desarrollada por IBM. En este modo de compilación no se podrá ejecutar dicho algoritmo (pero si el resto).

Para compilar, ubicado en la carpeta ***Totem/*** ejecute el comando **ant** con target **build.**

```
[Totem]$ ant build
```

De forma equivalente puede ejecutar solo el comando **ant,** ya que por defecto se compila utilizando el target **build.**

```
[Totem]$ ant
```

Con CPLEX:

En caso de contar con una licencia de CPLEX, se deberá realizar las siguientes modificaciones sobre los archivos ***build.xml*** y ***totem.sh***. Estas modificaciones refieren a la ubicación de la instalación de CPLEX en el sistema.

En el archivo ***build.xml*** se deberá modificar el camino al archivo **cplex.jar** en la propiedad **path** con **id**=”**classpathfull”**.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<project basedir="." default="build" name="Totem">
    <property environment="env"/>
    <property name="target" value="1.6"/>
    <property name="source" value="1.6"/>
    <path id="classpathfull">
        <pathelement location="bin"/>
        .
        .
        .
        .
        <pathelement location="lib/java/ejml-0.19.jar"/>
        <pathelement location="/usr/local/cplex122/cplex/lib/cplex.jar"/>
    </path>

    <path id="classpath">
```

El archivo **cplex.jar** se encuentra en **PATH_TO_CPLEX/cplex/lib**. Modifique la línea resaltada en la imagen anterior con su ubicación.

En el archivo ***totem.sh*** deberá modificar el camino hacia las librerías de CPLEX e indicar la arquitectura del equipo donde se encuentra instalado.

```
#!/bin/sh

CPLEX_PATH_LIB=/usr/local/cplex122/cplex/lib/
CPLEX_PATH_BIN_ARQ=/usr/local/cplex122/cplex/bin/x86_sles10_4.1/
.
.
.
```

CPLEX_PATH_LIB: Contiene el valor **PATH_TO_CPLEX/cplex/lib**, de acuerdo a la ubicación de su instalación.

CPLEX_PATH_BIN_ARQ: Contiene el valor **PATH_TO_CPLEX/cplex/bin/YOUR_ARQUITECTURE**, de acuerdo a la ubicación de su instalación, donde **YOUR_ARQUITECTURE** está dado por la arquitectura para la cual se instaló en su máquina.

Finalmente, luego de realizar estas modificaciones, ubicado en la carpeta ***Totem/*** ejecute el comando **ant** con target **buildfull.**

```
[Totem]$ ant buildfull
```

3. ¿Cómo ejecutar?

Para ejecutar el algoritmo, ubicado en ***Totem/*** ejecute el script ***totem.sh***.

```
[Totem]$ totem.sh
```

4. Software requerido

En esta sección se proporcionan los links para descargar el software requerido por el proyecto.

Java JDK 1.6 y 1.7

http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

IBM ILog CPLEX Optimizer

http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

Como ya se comentó el algoritmo **Óptimo Buob** requiere una licencia de CPLEX para funcionar sin limitaciones. Con una versión de evaluación de CPLEX es posible compilar y ejecutar el algoritmo, pero el tamaño de la red sobre la que se ejecute el algoritmo está limitado a un número pequeño de nodos.

[10]

Grafo separador utilizado por BGPSep

10.1 En busca del grafo separador

En esta sección se define formalmente el problema de hallar un separador de grafo, se nota que el problema es NP-Hard y se presentan y evalúan dos enfoques utilizados en la implementación para encontrar soluciones aproximadas. Los dos enfoques son mediante la utilización de un algoritmo evolutivo (EA) y mediante la utilización de un procedimiento de búsqueda local GRASP.

Especificación del problema

Un grafo separador de un grafo es un conjunto de vértices cuya eliminación separa a un grafo en dos o más componentes conexas. Más formalmente, dado un grafo $G = (V, E)$, con V conjunto de vértices y E conjunto de aristas, con $|V| = n$, un grafo separador es un conjunto $S \subset V$ con las siguientes propiedades:

- El subgrafo resultante de hacer G/S no es conexo.
- $|S| < n$.

En el contexto de nuestro problema es necesario encontrar un grafo separador pequeño y balanceado (el tamaño de las componentes resultantes es aproximadamente igual entre ellas). En general encontrar un grafo separador óptimo se convierte en un problema NP-hard, pero se conocen soluciones rápidas y exactas para casos particulares.

A continuación presentamos dos enfoques metaheurísticos distintos, uno basado en EA (Evolutionary Algorithms) y otro en GRASP (Greedy Randomized Adaptive Search Procedure). Estos procedimientos son conceptualmente simples y encuentran soluciones de buena calidad (no necesariamente hallan la solución óptima) a problemas difíciles, de un modo sencillo y eficiente.

Evolutionary Algorithm (AE)

Los algoritmos evolutivos son una técnica aleatoria guiada que trabaja sobre una población de soluciones que evoluciona mediante mecanismos de selección y construcción de nuevas soluciones candidatas por recombinación de características de las soluciones anteriores. Se pueden distinguir las siguientes etapas principales:

- Construcción de la población inicial.
- Evaluación de la función de fitness (evalúa la calidad de las soluciones).
- Selección de individuos (de acuerdo al fitness).
- Aplicación de operadores evolutivos.
- Reemplazo o recambio generacional.

Existen múltiples variaciones para cada una de las etapas. En definitiva, lo que trata de simular este enfoque es el comportamiento evolutivo de las especies. Existe una población de individuos en la cual aquellos individuos más adaptados van a tener más probabilidad de sobrevivir a su ambiente teniendo la posibilidad de aparearse con otros individuos generalmente también bien adaptados generando así las siguiente generación de individuos que se supone "está mejor adatada" que la anterior (este proceso se repite varias veces). Por más información sobre algoritmos evolutivos consultar [15].

Solución

Nuestra solución se basa en la ubicación de los routers de la red, todo router de una red está ubicado en coordenadas (x, y). La idea es utilizar cortes latitudinales, longitudinales y diagonales en la red para determinar un posible grafo separador de la misma. Estos cortes tienen generalmente alta probabilidad de ser un grafo separador. Usando un enfoque evolutivo nos proponemos evolucionar esos cortes para encontrar un grafo separador pequeño y balanceado. Pueden verse ejemplos sobre un grafo (Figura 10.1) de un corte diagonal (Figura 10.2), uno latitudinal (Figura 10.3) y uno longitudinal (Figura 10.4). Nótese que el corte de la Figura 10.2 es un grafo separador.

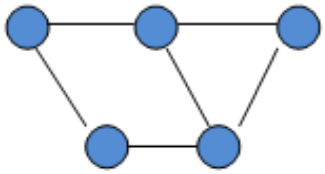


Figura 10.1: Grafo original

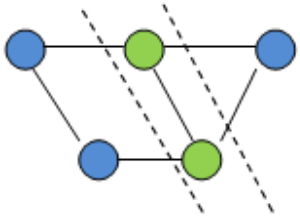


Figura 10.2: Corte diagonal (es un grafo separador)

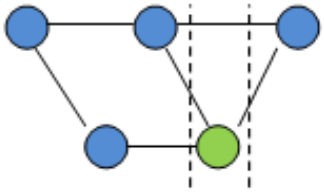


Figura 10.3: Corte latitudinal

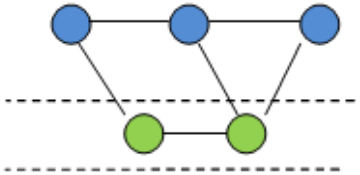


Figura 10.4: Corte longitudinal

Para fijar ideas veamos los siguientes ejemplos. Nos basaremos en el esquema clásico donde se destacan las etapas antes mencionadas.

```
Evolutive_Algorithm(G=(V, E), number_generation, size_population, size_offspring, probability_mutation,
probability_cross)
        S = EMPTY;
        S_Fitness = INFINITY;
        P = Construct_Initial_Population(G, size_population);

        For i=1 to number_generation do
                F = Evaluate_Polpulation(G, P, size_population);
```

```
                S' = Select_Best_Guy(P, F, size_population);

                If Fitness(G, S') < S_Fitness Then
                        S = S';
                        S_Fitness = Fitness(G, S');
                End_If

                P' = Select_Parents(P, F, size_offspring);
                P'' = Reconbine_Parents(P', size_offspring, probability_mutation, probability_cross);
                P = Remplace_Generational(P'', size_population, size_offspring);
        End_For;

        retrun S;
End_EvolutiveAlgorithm;
```

A. Representación de la solución

Para la representación de la solución se eligió una representación binaria, dada una red de tamaño n se asigna a cada router de la red un identificador de 0 a $n-1$. Dada una tira de bits de tamaño n si la posición i de la tira contiene un 1 determina que el nodo con identificador i pertenece al grafo separador, en otro caso no pertenece.

B. Construcción de la población inicial

Para la construcción de la población inicial tomamos una red e identificamos los valores máximos y mínimos tanto en el eje X como en el eje Y que toman las coordenadas de los routers. Esto nos determina el marco de la red. Para fijar ideas veamos el siguiente ejemplo presentado en la Figura 10.5.

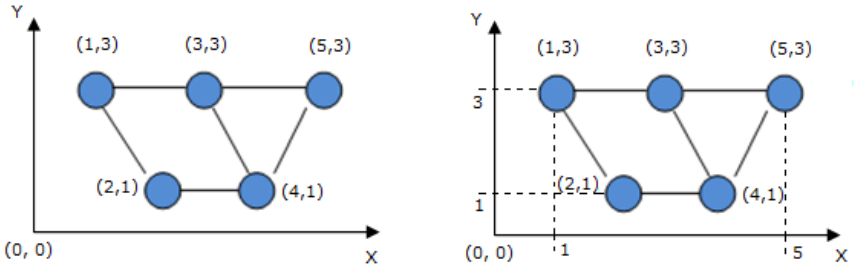


Figura 10.5: El grafo original es enmarcado y obtenemos $(x_{min}, x_{max}) = (1,5)$ *y* $(y_{min}, y_{max}) = (1,3)$

Como puede verse en la figura, luego del proceso anterior obtenemos como resultado dos parejas (x_{min}, x_{max}) y (y_{min}, y_{max}) que son utilizadas para determinar los cortes de la red que forman los individuos de la población inicial. Una vez encontrado el marco todos los cortes son efectuados dentro del mismo. Existen tres tipos de cortes en la red: latitudinales, longitudinales y diagonales.
Para efectuar los cortes dentro del marco sorteamos dos parejas de puntos (x_i, y_i) con x_i entre $[x_{min}, x_{max}]$ e y_i entre $[y_{min}, y_{max}]$. Luego hayamos la recta que pasa por los puntos (x_1, y_1) y (x_2, y_2). Finalmente para lograr un corte buscamos

otra recta paralela a la anterior (basta encontrar otra recta con el mismo coeficiente angular y variar la ordenada de origen).

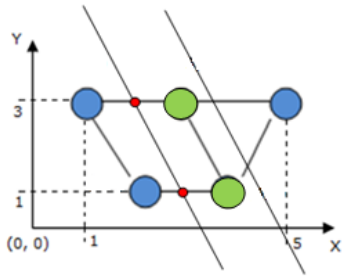


Figura 10.6: Ejemplo de corte diagonal

Todos los nodos con coordenadas dentro de ambas rectas son considerados en el corte. Para efectuar cortes longitudinales hacemos que $x_1 = x_2$. Para efectuar cortes latitudinales hacemos que $y_1 = y_2$.

El tamaño de la población inicial es un parámetro del algoritmo, y se inicializa de la siguiente manera.

```
Construct_Initial_Population(G, size_population)
        P = EMPTY;
        For i=1 to size_population do
                ran = Random(3);
                If(ran==0)
                        P(i) = Longuitudinal_Cut(G);
                Else If(ran==1)
                        P(i) = Latitudinal_Cut(G);
                Else If(ran==2)
                        P(i) = Diagonal_Cut(G);
                End_If;
        End_For;
        retrun P;
End_Construct_Initial_Population;
```

En general cada tipo de corte representará aproximadamente 1/3 de la cantidad de individuos totales.

C. Evaluación de la función de fitness

Como se observó anteriormente, si bien la probabilidad de que los cortes elegidos sean grafos separadores es alta, no podemos asegurar que definitivamente separen al grafo. Por eso es necesario determinar qué hacer al generar soluciones no factibles. Entre las opciones tenemos refactibilizar el individuo, descartarlo o penalizarlo a través de la función de fitness. En nuestra implementación elegimos penalizar a través de la función de fitness. Esto es ventajoso porque puede que el individuo aporte características deseadas en futuras recombinaciones.

Dado un grafo $G = (V, E)$ y $S \subset V$ un grafo separador, definimos el balance como

$$\text{balanced}_G(S) = \sum_{G_i \in G/S} \left| \frac{\sum_{G_i \in G/S} |G_i|}{\sum_{G_i \in G/S} 1} - |G_i| \right|$$

Donde G_i es una componente resultante de la separación G/S.

Entonces, definimos a la función de fitness como, dado $S \subset V$ un individuo de la población

$$\text{fitness}_G(S) = \begin{cases} 1/(S+1) & \text{si} \quad 1 = \sum_{G_i \in G/S} 1 \\ \dfrac{2 \times (|V|+1) \times \text{size}(S)}{(\text{balanced}_G(S) \times |V|) + 1} + 1/(S+1) & \text{otherwise} \end{cases}$$

Donde $1/(S+1)$ es un valor entre $(0,1)$ que se utiliza para no descartar completamente la solución y permitir que características de individuos menos aptos sobrevivan el pasaje entre generaciones y $\text{size}(S)$ determina el tamaño del grafo separador.

Esta función surge como el resultado de muchos intentos en la búsqueda de una función objetivo. La principal característica de la misma es que en la búsqueda de un grafo separador pequeño y balanceado esta función prefiere aquellas soluciones balanceadas. Como ya se observó anteriormente es muy sencillo conseguir soluciones no factibles. En ese contexto, esta función tiene poca presión selectiva por lo tanto en el momento que el algoritmo evolutivo no encuentre soluciones factibles el mismo intentará agregar nodos al grafo separador para lograr que sea efectivamente un separador. Esto se deduce desde el término principal donde se ve que el subtérmino $\frac{\text{size}(S)}{\text{balanced}_G(S)}$ adquiere los mayores valores con tamaños de grafo separador altos y balanceos bajos.

Los individuos son evaluados como sigue.

```
Evaluate_Polpulation(G, P, size_population)
        F = EMPTY;
        For i=1 to size_population do
                F(i) = Fitness(G, P(i));
        End_Foreach;
        return F;
Evaluate_Polpulation;
```

D. Selección de individuos

Para la selección de los individuos utilizamos el operador Roulette Wheel (rueda de ruleta). Un ejemplo de aplicación de este operador puede verse en la Figura 10.7. A cada individuo de la población se le asigna una probabilidad calculada como el fitness del individuo sobre el fitness total de la población (en un tiempo t). Sea P la población de individuos se calcula la probabilidad de S según P como

$$\text{probability}_G(S, P) = \frac{\text{fitness}_G(S)}{\sum_{S_i \in P} \text{fitness}_G(S_i)}$$

Esta probabilidad es usada para determinar que porción de la rueda ocupa dicho individuo. Individuos más adaptados tendrán mayor fitness y por lo tanto ocuparan una sección más grande en la rueda lo que determinará que al momento de "girar la rueda" estos tengan más probabilidad de ser elegidos. Esto alienta a la aparición de individuos dominantes y a la pérdida de la diversidad, efectos no deseados en una población. Más adelante se continúa con esta observación.

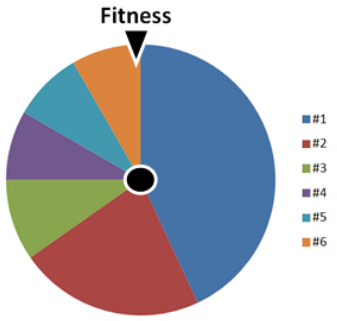


Figura 10.7: Efecto de aplicación del operador de cruzamiento.

Los individuos son seleccionados como sigue.

```
Select_Parents(P, F, size_offspring)
        P' = EMPTY;
        For i=1 to size_offspring do
                P'(i) = Select_Parent_By_Roulette_Wheel(P, F);
        End_For;
        return P';
End_Select_Parents;
```

E. Aplicación de operadores evolutivos

Operador de mutación

Dada nuestra representación (una tira de bits) como operador de mutación elegimos mutación de un bit. Un ejemplo de aplicación de dicho operador puede verse en la Figura 10.8. Dado S un individuo de la población la probabilidad de mutar cada bit es de $\text{probability\_mutation}$ (parámetro del algoritmo). Este operador invierte el valor de bit original (si es 1 pasa a cero, y viceversa). Podemos ver un ejemplo de aplicación en la Figura 10.8.

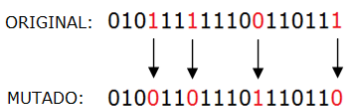


Figura 10.8: Efecto del operador de mutación sobre una tira de bits

Operador de cruzamiento

Como operador de cruzamiento elegimos cruzamiento de dos puntos. Un ejemplo de aplicación de este operador puede verse en la Figura 10.9. Dados dos individuos de la población S_1, S_2 la probabilidad de aplicar el operador de cruzamiento es de probability_cross (parámetro del algoritmo). Este operador intercambia todos los bits de un individuo a otro entre dos puntos sorteados.

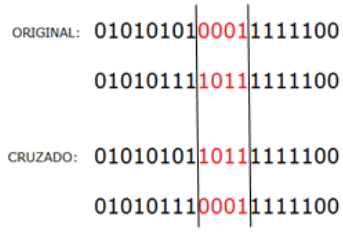


Figura 10.9: Efecto del operador de cruzamiento sobre dos tiras de bits

Los individuos son recombinados como sigue.

```
Recombine_Parents(P', size_offspring, probability_mutation, probability_cross)
        Cross_Operator(P', size_offspring, probability_cross);
        Mutation_Operator(P', size_offspring, probability_mutation);
        return P';
End_Recombine_Parents;


Cross_Operator(P', size_offspring, probability_cross)
        For i=1 to size_offspring - 1 do
                If Random() < probability_cross  Then
                        Cross(P'(i), P'(i+1), );
                End_If;
        End_For;
End_Cross_Operator;


Mutation_Operator(P', size_offspring, probability_cross)
        For i=1 to size_offspring do
                Mutation(P'(i), probability_cross);
        End_For;
End_Cross_Operator;
```

F. Reemplazo o recambio generacional

Dado una población resultado de la etapa de recombinación elegimos como remplazo generacional una selección aleatoria. Esto quitará presión en la selección de los futuros padres dando la posibilidad de que individuos menos adatados se conviertan en futuros padres, lo que permite mantener la diversidad en la población.

El remplazo se realiza como sigue.

```
Replace_Generation(P'', size_population, size_offspring)
        P = EMPTY;
        For i=1 to size_population do
                ran = Random(size_offspring)
```

```
                P(i) = P''(ran);
        End_For;
        retrun P;
End_Replace_Generation;
```

Greedy Randomized Adaptive Search Procedure (GRASP)

Este enfoque es clasificado como de exploración local, es un proceso iterativo sobre dos etapas principales:

- Construcción de la solución factible
- Mejora de la misma (búsqueda local)

La etapa de construcción se caracteriza por ser greedy aleatorizada. Se comienza con una solución vacía y se va construyendo iterativamente como resultado de toma de decisiones locales. Luego de este proceso no se puede asegurar que la solución construida sea un óptimo local (mucho menos global) por lo que resulta beneficioso aplicar una búsqueda local para intentar mejorar cada solución construida. La etapa de mejora de la solución se caracteriza por remplazos sucesivos en la solución actual. Esta etapa depende fuertemente de la calidad de las soluciones que retorna la etapa de construcción. Por más información acerca de GRASP consultar [16].

Solución

Se construirá un pseudo-bisector (la cantidad de componentes en el subgrafo G/S es al menos 2) que no tomará en cuenta la ubicación de los routers y además dará menos importancia al balanceo (más adelante se justificará porqué).

Para la representación de la solución se utilizará una terna $(S, C1, C2)$ donde S es el grafo separador y $C1, C2$ componentes (o conjuntos de nodos).

Definimos balanceo del pseudo-bisector como:

$$balanced\big((S, C1, C2)\big) = \sum_{i=1..2} |\boldsymbol{SIZE}(C_i) - (\sum_{i=1..2} \boldsymbol{SIZE}(C_i)) / 2|$$

Nos basaremos en el esquema clásico donde se destacan las dos etapas antes mencionadas.

```
Greedy_Randomized_Adaptive_Search_Procedure(G=(V, E), max_iter, alfa, beta, gamma)
        (S, C1, C2) = (EMPTY, EMPTY, EMPTY);

        For i=0 to max_iter do
                (S', C1', C2') = Construct_Greedy_Randomized_Solution(G);
                (S', C1', C2') = Local_Search(G, (S', C1', C2'), gamma);
                If Is_Better(alfa, beta, (S', C1', C2'), (S, C1, C2)) Then
                        (S, C1, C2) = (S', C1', C2');
                End_If;
        End_For;

        return (S, C1, C2);
End_Greedy_Randomized_Adaptive_Search_Procedure;
```

```
Is_Better(alfa, beta, (S', C1', C2'), (S, C1, C2))
        S_size = SIZE(S);
        S_average = (SIZE(C1) + SIZE(C2)) / 2;
        S_balanced = |SIZE(C1) - S_average| + |SIZE(C2) - S_average|;

        S'_size = SIZE(S')
        S'_average = (SIZE(C1') + SIZE(C2')) / 2;
        S'_balanced = |SIZE(C1') - S'_average| + |SIZE(C2') - S'_average|;


        If S'_size < S_size Then
                If S'_balanced < S_balanced Then
                        return TRUE;
                Else If S'_balanced <= ((1+alfa)×S_balanced) Then
                        return TRUE;
                End_If;
        Else If S'_size <= ((1+beta)×S_size) Then
                If S'_balanced < S_balanced Then
                        return TRUE;
                End_if;
        End_If;

        return FALSE;
End_Is_Better;
```

La función **Is_Better**(…) determina si $(S', C1', C2')$ es mejor que $(S, C1, C2)$ según $alfa$ y $beta$. Dado que el problema de encontrar un grafo separador es un compromiso entre el balance de G/S y el tamaño de S (define un frente de soluciones) es necesario definir un criterio de elección entre dos soluciones.

A. Construcción de la solución factible

Se toman dos nodos $u, v \in V$ tal que no exista $(u, v) \in E$, entonces definimos como grafo separador a $S = V/\{u, v\}$. Es fácil ver que S se convierte en una solución factible que tiene balance perfecto, no es para nada pequeña y donde las componentes resultantes de G/S son dos: una que contiene a sólo u y otra que contiene sólo a v. Se pueden ver ejemplos de soluciones factibles en la Figura 10.10.

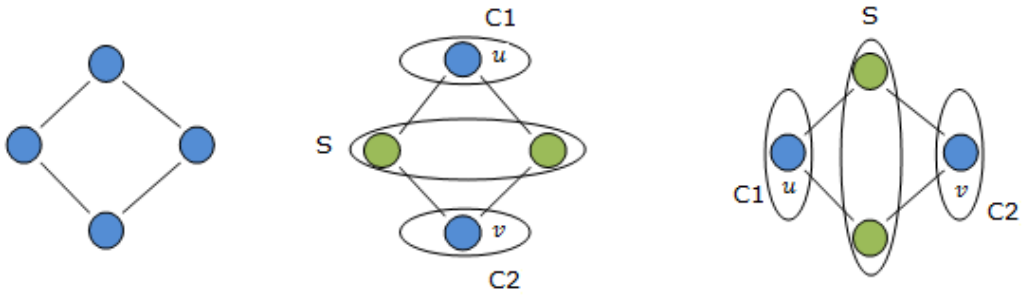


Figura 10.10: Ejemplos de soluciones factibles construidas en la fase de construcción.

B. Mejora de la solución factible

Como se observa en la etapa de construcción, el tamaño del grafo separador S en las construcciones resultantes no es para nada pequeño. La mejora local tratará de pasar nodos pertenecientes a S a las componentes donde se encuentra los nodos u y v conservado la factibilidad de la solución (debe ser un grafo separador).

Definición: Dado $G = (V, E)$ y tres conjuntos $S, C_1, C_2 \subset V$ tal que $S \cap C_1 = \emptyset$, $S \cap C_2 = \emptyset$ y $C_1 \cap C_2 = \emptyset$ definimos nodos candidatos de C_1 según S y C_2 y lo denotamos como $C_1^{(C_2,S)}$ al conjunto

$$C_1^{(C_2,S)} = \{u \in S \mid \exists (w,u) \in E \; y \nexists (u,v) \in E \; con \; v \in C_2 \; y \; w \in C_1\}$$

Basados en esta definición y aplicando la misma en el pseudo-bisector, podemos ver que todo nodo candidato a C_1 según S y C_2 puede ser quitado de S y agregado en C_1 (análogo para C_2), esto nos permite minimizar el tamaño de S y conservar la factibilidad. La aplicación de esta propiedad en la mejora local se efectúa de la siguiente manera.

```
Local_Search(G, (S', C1', C2'), gamma)
        follow = TRUE;

        While follow do
                follow = FALSE;
                While C1'(C2', S') != EMPTY and SIZE(C1') < SIZE(C2') ×gamma do
                        follow = TRUE;

                        u = Select(C1'(C2', S'));
                        S = S/u;
                        C1' = C1' ∪ u;
                End_While;

                While C2'(C1', S') != EMPTY and SIZE(C2') < SIZE(C1') ×gamma do
                        follow = TRUE;

                        u = Select(C2'(C1', S'));
                        S = S/u;
                        C2' = C2' ∪ u;
                End_While;
        End_While;
        return (S', C1', C2');
End_Local_Search;
```

Como se puede ver se quitarán nodos candidatos de ambas componentes (o conjuntos de nodos) de forma que $SIZE(C_1') \leq SIZE(C_2') \times gama$ ó $SIZE(C_2') \leq SIZE(C_1') \times gama$. Este proceso de intercambio de nodos candidatos desde S' a las componente (o conjuntos de nodos) genera un pseudo-bisector debido a que el subgrafo G/S' puede llegar a tener más de dos componentes conexas, lo que convierte a C_1' y C_2' en simples conjuntos de nodos.

A continuación veamos el siguiente ejemplo, dado el grafo mostrado en la Figura 10.11:

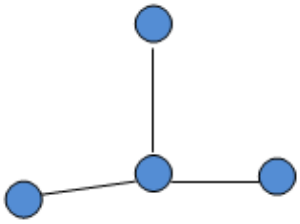


Figura 10.11: Grafo original.

Podríamos tener como resultado de la construcción factible el grafo mostrado en la Figura 10.12:

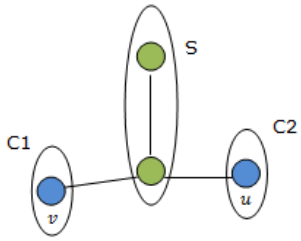


Figura 10.12: Construcción factible.

Y como resultado luego de aplicar el procedimiento de búsqueda local el grafo de la Figura 10.13:

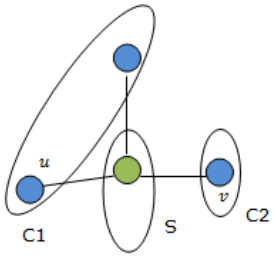


Figura 10.13: Resultado de aplicar el procedimiento Local_Search(…).

Tenemos entonces que el procedimiento de búsqueda local construye un pseudo-bisector ya que en realidad resultan más de dos componentes (son tres componente conexas). Esto además provoca que el balance definido anteriormente pierda el sentido ocasionando posiblemente balanceos muy grandes o lo que es lo mismo “desbalanceo”.

10.2 Calibración de los algoritmos de grafo separador utilizados por BGPSep

Como ya se mencionó, BGPSep utiliza la noción de un grafo separador para elegir reflectores de rutas y sesiones iBGP de manera de garantizar la correctitud. El principal aspecto que determina la topología resultado en la ejecución de un algoritmo de la familia BGPSep es el grafo separador que se elige en cada paso de la ejecución. Luego que se han determinado los grafos separadores utilizados en cada paso de la ejecución la topología resultado estará determinada. Nótese entonces la importancia en la selección del grafo separador para obtener buenos resultados.

Recordemos que, a grandes rasgos, un separador de grafo es un conjunto de vértices cuya eliminación separa a un grafo en dos o más componentes conexas. Recordemos también que encontrar soluciones para este problema en su caso general es un problema NP-Hard. Por esto, como ya hemos visto en la sección anterior, hemos utilizado metaheurísticas que consisten de métodos no exactos para obtener soluciones cercanas al óptimo.

Se han implementados dos soluciones al problema: una utilizando un Algoritmo Evolutivo y otra utilizando la metaheurística GRASP. GRASP es una metodología potente que ha demostrado ser altamente eficiente al ser aplicada en otros problemas de optimización combinatoria y optamos por la implementación de un Algoritmo Evolutivo por tratarse de una de las técnicas más tradicionales.

Ambas aproximaciones utilizan varios parámetros que determinan elecciones de compromiso (como seleccionar un separador que consiste en menos nodos pero las componentes conexas resultado están desbalanceadas frente a otro con más nodos pero que las componentes conexas resultado están balanceadas) o su valor depende fuertemente del tipo de topologías a resolver (como la cantidad de nodos, por ejemplo).

Por otra parte, otros parámetros son inherentes a la propia metaheurística utilizada. Los Algoritmos Evolutivos basan su funcionamiento en procedimientos conceptualmente simples que encuentran soluciones de buena calidad (no necesariamente hallan la solución óptima) a problemas difíciles, de un modo sencillo y eficiente. Los procedimientos de búsqueda a menudo están basados en el sentido común o emulación de fenómenos bien conocidos. Las metaheurísticas consisten en estrategias que guían el proceso de búsqueda y son de uso genérico (no específicas para una cierta clase de problemas). Por esto es necesario calibrar los parámetros que utiliza como entrada: la probabilidad de cruzamiento y la probabilidad de mutación.

Mencionaremos ahora los parámetros que utilizan cada uno de los acercamientos implementados. Para el caso de la solución que utiliza la técnica GRASP los parámetros son:

1. max_iter: La cantidad máximas de iteraciones utilizadas.
2. alpha: Determina la decisión de compromiso que existe entre el balanceo de la cantidad de nodos en las componentes conexas resultado de eliminar el separador del grafo y la cantidad de nodos en el separador. Al aumentar el valor de alpha se prefieren soluciones desbalanceadas si el tamaño del grafo separador es menor. Es un valor entre 0 y 1.
3. beta: Determina la decisión de compromiso que existe entre el balanceo de la cantidad de nodos en las componentes conexas resultado de eliminar el separador del grafo y la cantidad de nodos en el separador. Al aumentar el valor de beta se prefieren soluciones balanceadas a pesar de que el tamaño del grafo separador sea mayor. Es una valor entre 0 y 1.
4. gamma: Al igual que beta determina la decisión de compromiso que existe entre el balanceo de la cantidad de nodos en las componentes conexas resultado de eliminar el separador del grafo y la cantidad de nodos en el separador. En la etapa de búsqueda local al aumentar el valor de gamma se considera que una solución "se acerca a un óptimo local" a pesar de que el tamaño del grafo separador sea mayor. Es una valor entre 0 y 1.

Para el caso de la solución que utiliza un Algoritmo Evolutivo los parámetros son:

1. nb_run: Cantidad de ejecuciones independientes del algoritmo. El resultado será la mejor solución obtenida en todas las ejecuciones.
2. n_gen: Cantidad de iteraciones utilizadas.
3. sizeP: Tamaño de la población utilizada.
4. sizeOf: Cantidad de individuos de la población que son sustituidos por sus hijos.
5. pmut: Probabilidad de mutación de un individuo.
6. pcross: Probabilidad de cruzamiento de un individuo.

Para realizar la calibración de ambos algoritmos se utilizó un conjunto de 30 topologías. Este conjunto de topologías fue generado utilizando IGen [22], un generador de topologías de código abierto.

A continuación presentamos una tabla que intenta caracterizar las topologías. Para cada topología se muestra la cantidad de nodos, la cantidad de enlaces, el grado promedio de los nodos, el grado máximo, el grado mínimo y por último la varianza del grado.

Topología	# Nodos	# Aristas	Grado (Avg)	Grado (Máx)	Grado (Mín)	Grado (Var)
test0.xml	12	60	5	8	2	2,5454545455
test1.xml	25	128	5,12	8	3	1,9433333333
test2.xml	50	204	4,08	18	2	21,911836735
test3.xml	25	130	5,2	8	3	1,8333333333
test4.xml	25	128	5,12	8	3	1,61
test5.xml	30	58	1,9333333333	4	1	0,5471264368
test6.xml	45	90	2	2	2	0
test7.xml	50	278	5,56	8	4	1,4759183673
test8.xml	55	266	4,8363636364	11	1	6,6949494949
test9.xml	60	244	4,0666666667	22	2	25,114124294
test10.xml	65	256	3,9384615385	6	2	1,1836538462
test11.xml	5	24	1,0909090909	2	1	0,0865800866
test12.xml	70	394	5,6285714286	10	3	1,7151138716
test13.xml	75	296	3,9466666667	6	2	1,1052252252
test14.xml	80	456	5,7	9	3	1,9088607595
test15.xml	90	376	4,1777777778	16	2	18,732084894
test16.xml	95	544	5,7263157895	9	4	1,8817469205
test17.xml	112	448	4	30	2	43,315315315
test18.xml	100	568	5,68	9	3	1,8157575758
test19.xml	100	406	4,06	36	2	46,40040404
test20.xml	100	396	3,96	6	2	1,0286868687
test21.xml	100	208	2,08	3	2	0,0743434343
test22.xml	6	20	3,3333333333	6	2	4,2666666667
test23.xml	9	36	4	8	2	9
test24.xml	18	127	7,1111111111	14	4	17,045751634
test25.xml	23	124	5,3913043478	26	2	21,612648221
test26.xml	26	192	7,3846153846	26	4	33,206153846
test27.xml	45	336	7,1489361702	14	1	12,607770583
test28.xml	18	108	5,6842105263	11	1	6,1169590643
test29.xml	15	54	3,6	10	2	8,2571428571

Figura 10.14: Caracterización de las topologías

Calibración del Algoritmo Evolutivo

La calibración de parámetros que hemos utilizado para el caso de la solución que consiste en un Algoritmo Evolutivo utiliza el conjunto de parámetros que está determinado por todas las combinaciones de sizeP, pmut, pcross variando cada uno de ellos en los siguientes conjuntos:

1. sizeP: Variando en (100, 200) de forma de tomar 100 valores tal que estén equiespaciados.
2. pmut: Variando en (0.01, 0.04) de forma de tomar 5 valores tal que estén equiespaciados.
3. pcross: Variando en (0.1, 0.4) de forma de tomar 5 valores tal que estén equiespaciados.

El resto de los parámetros necesarios para la ejecución del algoritmo serán fijados utilizando conocimiento generado en base a la experimentación. Los valores que toman los parámetros restantes son los siguientes:

1. nb_run: 10.
2. n_gen: 25.
3. sizeOf: Este parámetro tomará el valor del 60% de sizeP.

Topología	AE							
	Separador		Componentes					
	Tamaño	#S / #G	Cantidad	Tamaño Medio	Varianza	Máximo	Mínimo	Balanceo
test0.xml	4	0,3333333333	2	4	0	4	4	0
test1.xml	1	0,2	2	2	0	2	2	0
test2.xml	2	0,3333333333	4	1	0	1	1	0
test3.xml	3	0,3333333333	6	1	0	1	1	0
test4.xml	7	0,3888888889	11	1	0	1	1	0
test5.xml	14	0,6086956522	9	1	0	1	1	0
test6.xml	16	0,6153846154	10	1	0	1	1	0
test7.xml	31	0,6888888889	2	7	0	7	7	0
test8.xml	8	0,4444444444	2	5	0	5	5	0
test9.xml	5	0,3333333333	10	1	0	1	1	0
test10.xml	11	0,44	2	7	0	7	7	0
test11.xml	29	0,58	21	1	0	1	1	0
test12.xml	13	0,52	2	6	0	6	6	0
test13.xml	15	0,6	2	5	0	5	5	0
test14.xml	1	0,0333333333	2	14,5	24,5	18	11	7
test15.xml	3	0,0666666667	2	21	0	21	21	0
test16.xml	14	0,28	2	18	0	18	18	0
test17.xml	44	0,8	11	1	0	1	1	0
test18.xml	39	0,65	21	1	0	1	1	0
test19.xml	41	0,6307692308	2	12	0	12	12	0
test20.xml	28	0,4	2	21	0	21	21	0
test21.xml	9	0,12	2	33	0	33	33	0
test22.xml	40	0,5	2	20	0	20	20	0
test23.xml	64	0,7111111111	26	1	0	1	1	0
test24.xml	39	0,4105263158	2	28	0	28	28	0
test25.xml	69	0,6160714286	43	1	0	1	1	0
test26.xml	36	0,36	2	32	0	32	32	0
test27.xml	48	0,48	52	1	0	1	1	0
test28.xml	12	0,12	2	44	0	44	44	0
test29.xml	2	0,02	2	49	18	52	46	6

Figura 10.15: Estadísticas resultados

Realizando ejecuciones independientes para cada una de las combinaciones de parámetros que pueden generarse utilizando estos valores sobre las topologías mencionadas anteriormente construimos la tabla de la Figura 10.15 con los resultados obtenidos. Para cada una de las topologías se muestran datos para el mejor grafo separador encontrado utilizando todas las posibles combinaciones de parámetros dentro del conjunto de prueba. Además se muestra para cada una de las topologías datos de las componentes conexas

resultados de eliminar del grafo el mejor separador encontrado utilizando todas las posibles combinaciones de parámetros dentro del conjunto de prueba.

También se muestra la cantidad de nodos pertenecientes al mejor separador encontrado y la proporción de nodos del grafo que pertenecen a al separador. Luego se muestra la cantidad de componentes conexas, su tamaño promedio, la varianza de sus tamaños, el tamaño máximo, el tamaño mínimo y por último el balanceo. El balanceo se calcula como la suma para todas las componentes de la diferencia de la cantidad de nodos de la componente menos la cantidad media de nodos de las componentes.

Calibración de la metodología GRASP

La calibración de parámetros que hemos utilizado para el caso de la solución que consiste de una metodología GRASP utiliza el conjunto de parámetros que está determinado por todas las combinaciones de max_iter, alpha, beta y gamma variando cada uno de ellos en los siguientes conjuntos:

1. max_iter: Variando en {10000, 20000, 30000}.
2. alpha: Variando en (0.04, 0.05) de forma de tomar 5 valores tal que estén equiespaciados.
3. beta: Variando en (0.01, 0.02) de forma de tomar 5 valores tal que estén equiespaciados.
4. gamma: Variando en (0.10, 0.15) de forma de tomar 5 valores tal que estén equiespaciados.

Realizando ejecuciones independientes para cada una de las combinaciones de parámetros que pueden generarse utilizando estos valores sobre las topologías mencionadas anteriormente construimos la tabla presentada en la Figura 10.16 con los resultados obtenidos. Al igual que el caso de la calibración de la solución que utiliza un Algoritmo Evolutivo, para cada una de las topologías se muestran datos para el mejor grafo separador encontrado utilizando todas las posibles combinaciones de parámetros dentro del conjunto de prueba. Además se muestra para cada una de las topologías datos de las componentes conexas resultados de eliminar del grafo el mejor separador encontrado utilizando todas las posibles combinaciones de parámetros dentro del conjunto de prueba.

Nuevamente, al igual que el caso de la calibración de la solución que utiliza un Algoritmo Evolutivo, se muestra la cantidad de nodos pertenecientes al mejor separador encontrado y la proporción de nodos del grafo que pertenecen a al separador. Luego se muestra la cantidad de componentes conexas, su tamaño promedio, la varianza de sus tamaños, el tamaño máximo, el tamaño mínimo y por último el balanceo. El balanceo se calcula de forma análoga al caso anterior.

Topología	GRASP							
	Separador		Componentes					
	Tamaño	#S / #G	Cantidad	Tamaño Medio	Varianza	Máximo	Mínimo	Balanceo
test0.xml	2	0,166667	2	5	0	5	5	0
test1.xml	1	0,2	2	2	0	2	2	0
test2.xml	2	0,333333	4	1	0	1	1	0
test3.xml	2	0,222222	5	1,4	0,8	3	1	3,2
test4.xml	3	0,166667	5	3	7,5	6	1	12,0
test5.xml	2	0,086957	5	4,2	10,7	8	1	13,2
test6.xml	3	0,115385	3	7,6666666667	20,3333333333	12	3	9,3333333333
test7.xml	6	0,133333	5	7,8	38,7	18	1	20,4
test8.xml	2	0,111111	3	5,3333333333	6,3333333333	8	3	5,3333333333
test9.xml	2	0,133333	8	1,625	3,125	6	1	8,75
test10.xml	5	0,2	2	10	0	10	10	0
test11.xml	5	0,1	26	1,7307692308	6,8446153846	12	1	35,0769230769
test12.xml	5	0,2	2	10	0	10	10	0
test13.xml	5	0,2	2	10	0	10	10	0
test14.xml	2	0,066667	5	5,6	7,3	10	3	9,6
test15.xml	2	0,044444	2	21,5	0,5	22	21	1
test16.xml	8	0,16	2	21	0	21	21	0
test17.xml	11	0,2	2	22	0	22	22	0
test18.xml	4	0,066667	19	2,9473684211	34,2748538012	21	1	66,2105263158
test19.xml	5	0,076923	2	30	0	30	30	0
test20.xml	8	0,114286	2	31	0	31	31	0
test21.xml	3	0,04	2	36	0	36	36	0
test22.xml	8	0,1	2	36	0	36	36	0
test23.xml	8	0,088889	37	2,2162162162	26,6186186186	24	1	85,1351351351
test24.xml	9	0,094737	2	43	0	43	43	0
test25.xml	6	0,053571	72	1,4722222222	8,0273865415	20	1	66,1111111111
test26.xml	8	0,08	2	46	0	46	46	0
test27.xml	6	0,06	74	1,2702702703	5,4054054054	21	1	39,4594594595
test28.xml	6	0,06	2	47	0	47	47	0
test29.xml	8	0,08	7	13,1428571429	36,1428571429	22	5	33,1428571429

Figura 10.16: Estadísticas resultados

Comparación del Algoritmo Evolutivo y GRASP

Una vez realizadas las calibraciones de parámetros tanto para el enfoque que utiliza un Algoritmo Evolutivo como para el enfoque que utiliza una técnica GRASP se plantea la siguiente duda: ¿es conveniente utilizar la técnica que utiliza un Algoritmo Evolutivo o el enfoque que utiliza una técnica GRASP?. Para ayudar a responder esta pregunta hemos diseñado un conjunto de prueba en el que ejecutamos ambos algoritmos con el fin de poder comparar los resultados obtenidos por cada uno de ellos.

Para realizar la prueba de ambos algoritmos se utilizó un conjunto de 15 topologías. En la Figura 10.17 se presenta una tabla que intenta caracterizar las topologías. Para cada topología se muestra la cantidad de nodos, la cantidad de enlaces, el grado promedio de los nodos, el grado máximo, el grado mínimo y por último la varianza del grado.

Topología	# Nodos	# Aristas	Grado (Avg)	Grado (Máx)	Grado (Mín)	Grado (Var)
test0.xml	25	128	5,12	8	3	1,61
test1.xml	90	356	3,9555555556	6	2	1,0541822722
test2.xml	90	180	2	2	2	0
test3.xml	90	386	4,2888888889	15	2	16,589762797
test4.xml	100	576	5,76	10	3	1,861010101
test5.xml	100	396	3,96	6	2	0,9074747475
test6.xml	25	96	3,84	6	2	0,9733333333
test7.xml	50	274	5,48	9	3	1,5608163265
test8.xml	50	196	3,92	6	2	1,2995918367
test9.xml	50	202	4,04	23	2	25,467755102
test10.xml	75	422	5,6266666667	8	3	1,3181981982
test11.xml	75	296	3,9466666667	7	2	1,3214414414
test12.xml	75	150	2	2	2	0
test13.xml	75	310	4,1333333333	20	2	20,765765766
test14.xml	90	506	5,6222222222	9	3	2,4848938826

Figura 10.17: Caracterización de las topologías

En la Figura 10.18 se presenta una tabla que contiene los resultados obtenidos para el acercamiento que utiliza un Algoritmo Evolutivo y en la Figura 10.19 otra que contiene los resultados obtenidos para el acercamiento que utiliza GRASP. Ambas tablas son muy similares a las ya presentadas anteriormente por lo que no entraremos en detalle en estos casos.

Topología	AE							
	Separador		Componentes					
	Tamaño	#S / #G	Cantidad	Tamaño Medio	Varianza	Máximo	Mínimo	Balanceo
test0.xml	11	0,44	2	7	0	7	7	0
test1.xml	3	0,12	2	11	0	11	11	0
test2.xml	12	0,24	2	19	0	19	19	0
test3.xml	6	0,12	2	22	0	22	22	0
test4.xml	16	0,32	34	1	0	1	1	0
test5.xml	27	0,36	2	24	0	24	24	0
test6.xml	3	0,04	2	36	0	36	36	0
test7.xml	3	0,04	2	36	0	36	36	0
test8.xml	38	0,5066666667	37	1	0	1	1	0
test9.xml	36	0,4	2	27	0	27	27	0
test10.xml	12	0,1333333333	2	39	0	39	39	0
test11.xml	2	0,0222222222	2	44	0	44	44	0
test12.xml	52	0,5777777778	38	1	0	1	1	0
test13.xml	22	0,22	2	39	0	39	39	0
test14.xml	8	0,08	2	46	0	46	46	0

Figura 10.18: Estadísticas resultados AE

Topología	GRASP							
	Separador		Componentes					
	Tamaño	#S / #G	Cantidad	Tamaño Medio	Varianza	Máximo	Mínimo	Balanceo
test0.xml	5	0,2	2	10	0	10	10	0
test1.xml	3	0,12	2	11	0	11	11	0
test2.xml	8	0,16	3	14	127	21	1	26
test3.xml	4	0,08	2	23	0	23	23	0
test4.xml	4	0,08	31	1,4838709677	3,6580645161	10	1	28,06451612903227
test5.xml	11	0,1466666667	2	32	0	32	32	0
test6.xml	5	0,0666666667	3	23,3333333333	258,33333333	35	5	36,6666666667
test7.xml	6	0,08	6	11,5	46,7	20	3	35
test8.xml	5	0,0666666667	12	5,8333333333	127,60606061	31	1	96,6666666667
test9.xml	10	0,1111111111	2	40	0	40	40	0
test10.xml	10	0,1111111111	3	26,6666666667	457,33333333	40	2	49,3333333333
test11.xml	6	0,0666666667	6	14	58	26	5	36
test12.xml	8	0,0888888889	25	3,28	62,293333333	30	1	104,88
test13.xml	12	0,12	3	29,3333333333	522,33333333	44	3	52,6666666667
test14.xml	8	0,08	2	46	0	46	46	0

Figura 10.19: Estadísticas resultados GRASP

Se torna complicado leer los resultados de ambas tablas con el fin de comparar ambas técnicas, por lo que hemos decidido agregar a continuación un conjunto de imágenes que muestran la topología utilizada en la prueba a la izquierda, seguido del grafo separador obtenido por la técnica que utiliza un Algoritmo Evolutivo y por último el grafo separador obtenido por la técnica que utiliza el GRASP. Se presenta una figura para cada uno de los tests realizados.

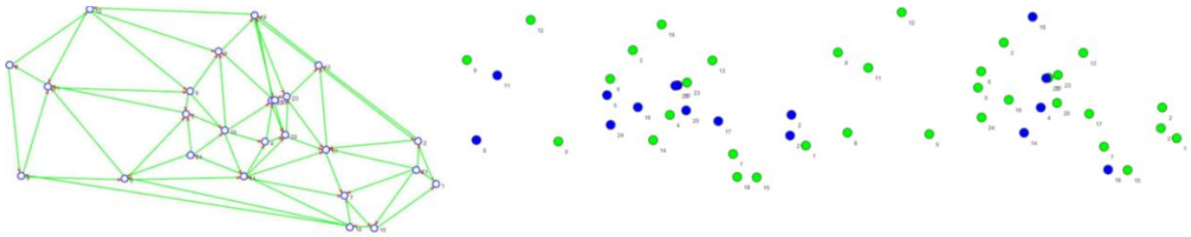


Figura 10.20: Test 1 - Resultados para el AE - Resultados para GRASP

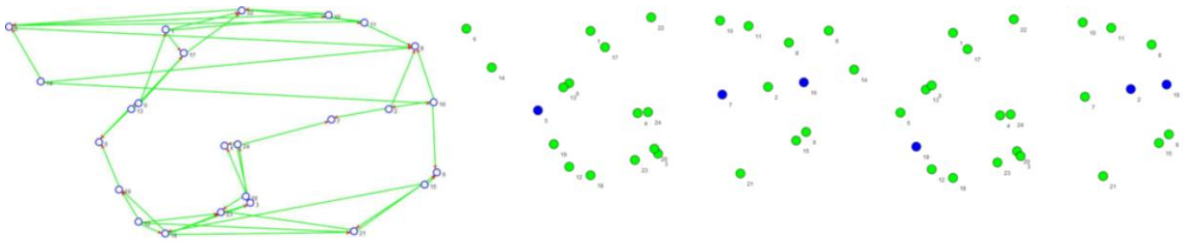


Figura 10.21: Test 2 - Resultados para el AE - Resultados para GRASP

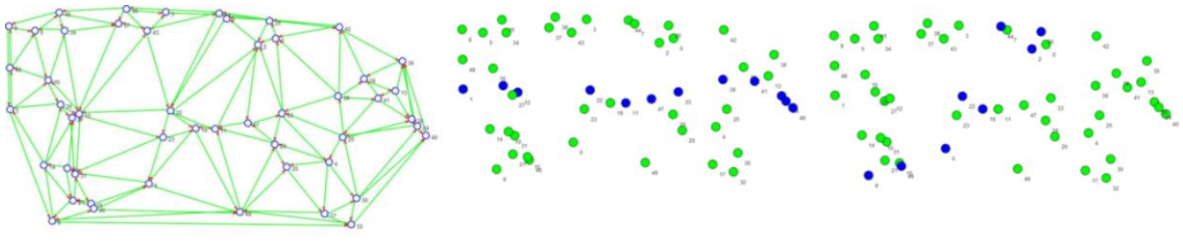


Figura 10.22: Test 3 - Resultados para el AE - Resultados para GRASP

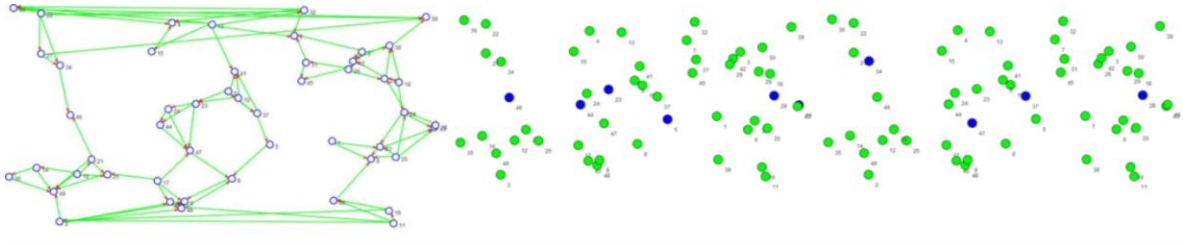


Figura 10.23: Test 4 - Resultados para el AE - Resultados para GRASP

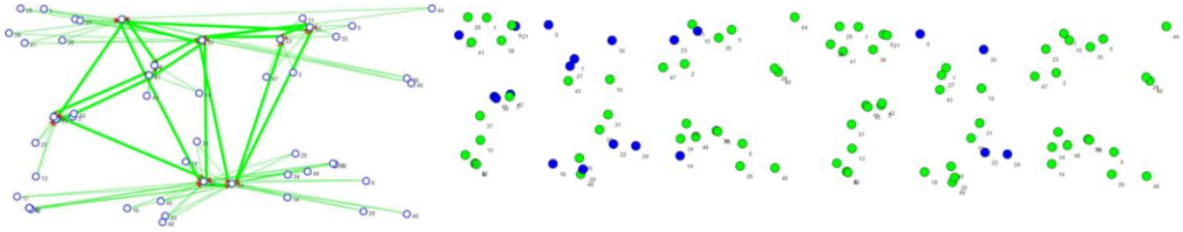


Figura 10.24: Test 5 - Resultados para el AE - Resultados para GRASP

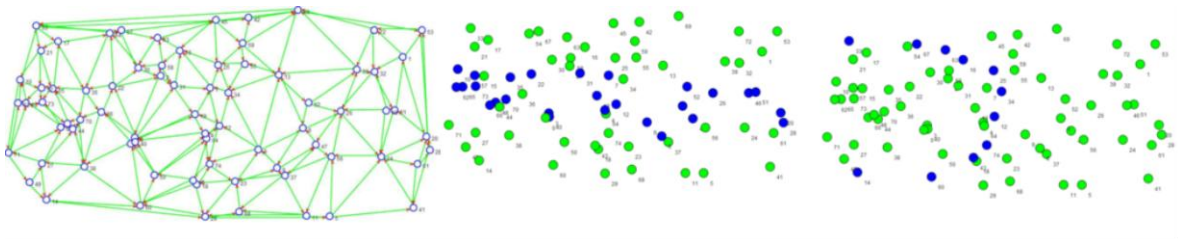


Figura 10.25: Test 6 - Resultados para el AE - Resultados para GRASP

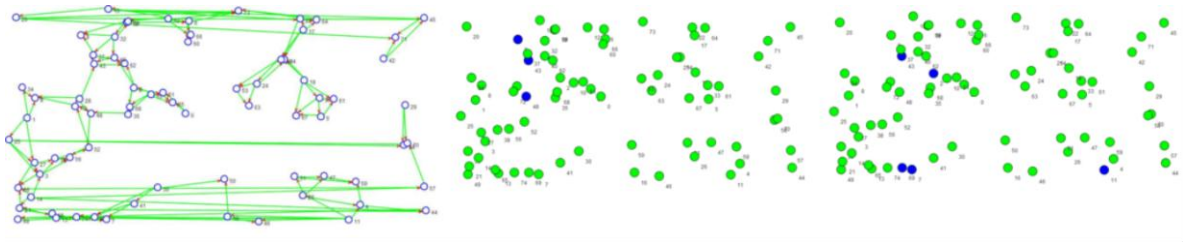


Figura 10.26: Test 7 - Resultados para el AE - Resultados para GRASP

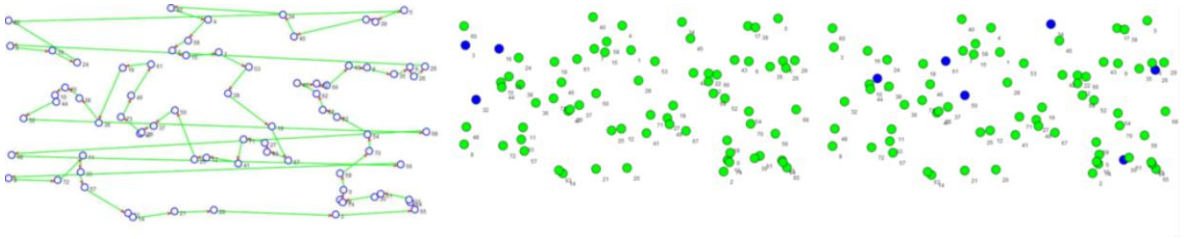


Figura 10.27: Test 8 - Resultados para el AE - Resultados para GRASP

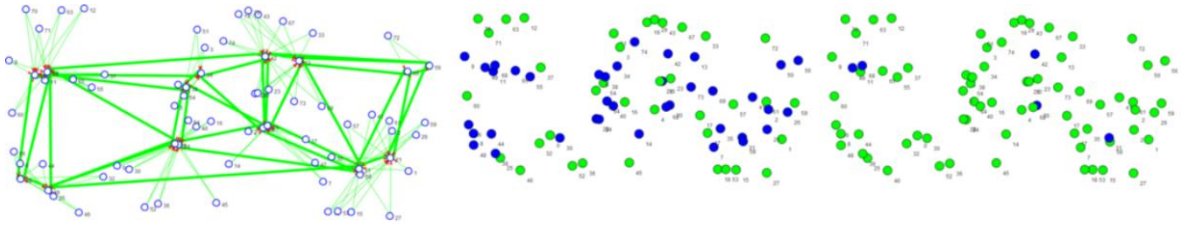


Figura 10.28: Test 9 - Resultados para el AE - Resultados para GRASP

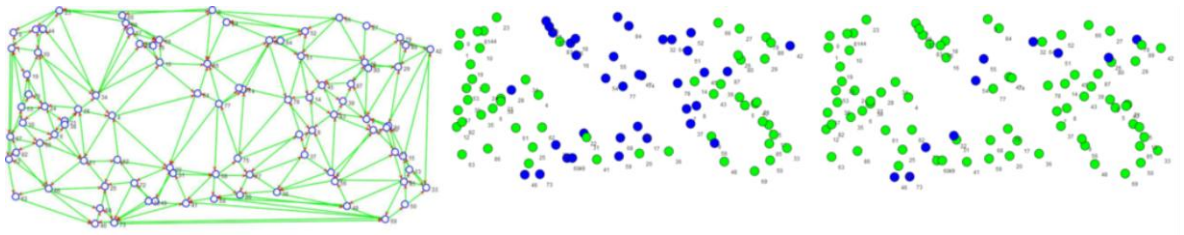


Figura 10.29: Test 10 - Resultados para el AE - Resultados para GRASP

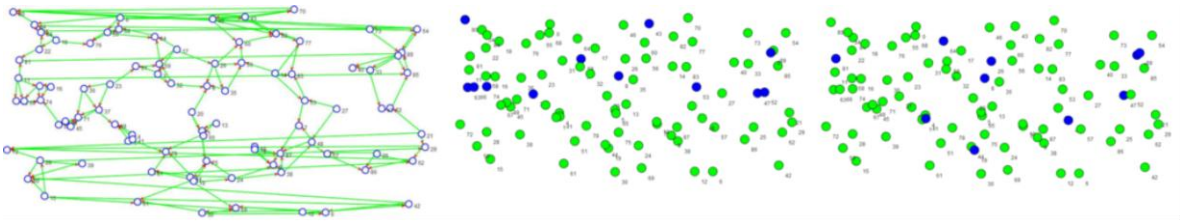


Figura 10.30: Test 11 - Resultados para el AE - Resultados para GRASP

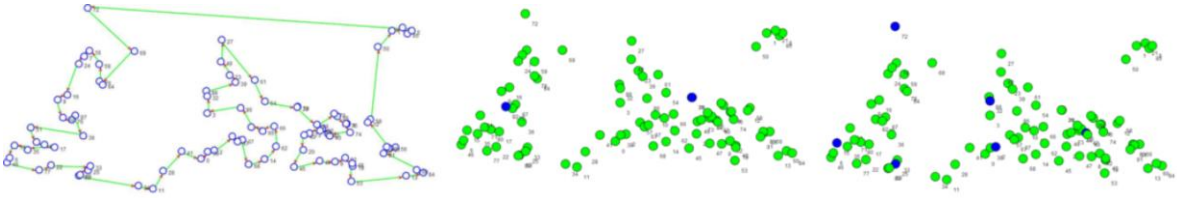


Figura 10.31: Test 12 - Resultados para el AE - Resultados para GRASP

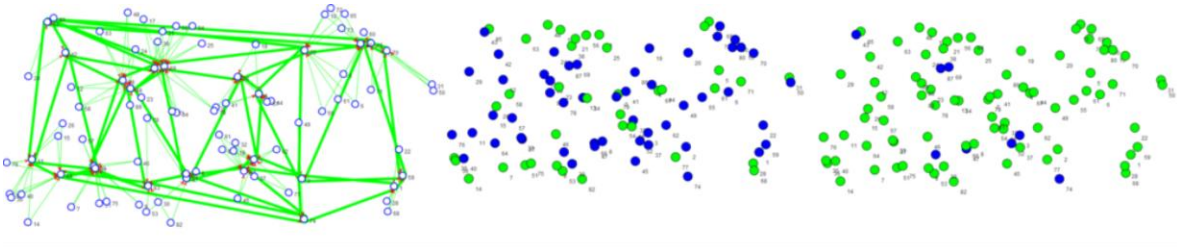


Figura 10.32: Test 13 - Resultados para el AE - Resultados para GRASP

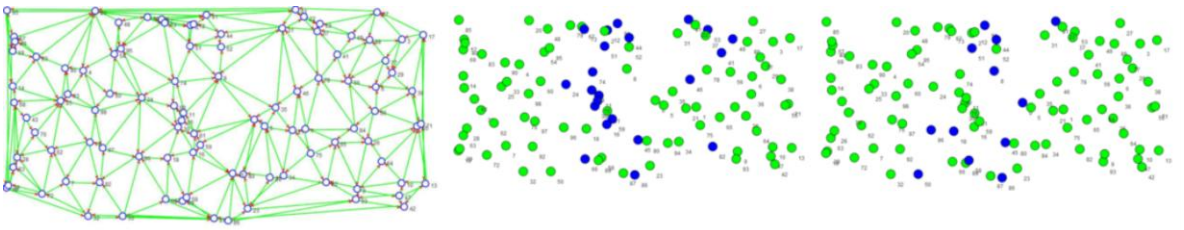


Figura 10.33: Test 14 - Resultados para el AE - Resultados para GRASP

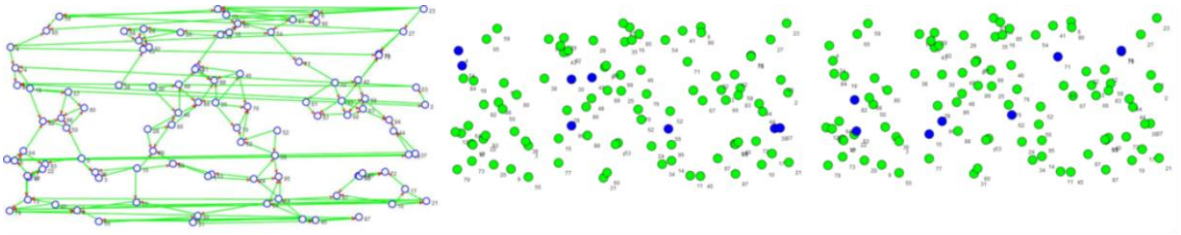


Figura 10.34: Test 15 - Resultados para el AE - Resultados para GRASP

Luego de analizar los resultados podemos decir que el acercamiento que utiliza una técnica GRASP obtiene, en términos generales, mejores resultados. De todas formas consideramos que no son despreciables los resultados que obtiene la técnica que utiliza un Algoritmo Evolutivo, nótese que ha obtenido en algunas de las pruebas resultados muy buenos; comparables o incluso mejores que los obtenidos por el acercamiento que utiliza el GRASP.