Detección de wedgies en BGP

Lara Carrión Ovejero

Grado en Ingeniería Informática
Facultad de Informática
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo de Fin de Grado

Departamento de Arquitectura de Computadores y Automática

Septiembre de 2015

Director:
Juan Carlos Fabero Jiménez

Detección de wedgies en BGP

Memoria Trabajo Fin de Grado presentado por
Lara Carrión Ovejero

Dirigido por
Dr. Juan Carlos Fabero Jiménez

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2015

Autorizo a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria como el código, la documentación, contenidos audiovisuales y/o prototipo desarrollado.

Fdo. Lara Carrión Ovejero

Agradecimientos

En primer lugar, me gustaría agradecer todo el apoyo incondicional y ánimo para seguir adelante recibido durante estos años por parte de mis padres y mi hermana, sin los cuales no hubiera sido posible llegar hasta aquí.

También agradecer a todos los amigos y compañeros de la facultad que han formado parte de esto en algún momento durante todo este tiempo. Gracias por el compañerismo, las risas en los buenos momentos y el apoyo en los no tan buenos, las horas en la cafetería, las horas en la biblioteca y las interminables tardes en los laboratorios.

Tampoco olvidarme de todos los amigos y compañeros provenientes de todas partes del mundo que he conocido durante mi año académico en la *University of Calgary*, en Canadá. Gracias a todos los que hicieron que 8 meses a casi 8000 kilómetros de casa merecieran la pena.

Y especialmente a mi tutor, Juan Carlos, por el apoyo prestado en la elaboración de este proyecto y su ayuda en todo momento.

Resumen

Hoy en día, la comunicación a través de Internet ocupa un lugar primordial en nuestra vida diaria. Con el propósito de que las comunicaciones lleguen a todas las partes del mundo, es necesario disponer de protocolos de encaminamiento que permitan guiar el tráfico en Internet. El protocolo de encaminamiento exterior más utilizado es *Border Gateway Protocol* (BGP), creado para que distintos dominios interconectados entre sí puedan intercambiar información de encaminamiento. Esta información es suficiente para construir el grafo de conectividad de la red, de manera que puedan aplicarse las políticas de encaminamiento adecuadas a cada situación. Los resultados obtenidos con BGP no siempre llevan a estados deterministas tras aplicar una política de encaminamiento determinada, sino que es posible que aparezcan estados no deseados y no deterministas, denominados *BGP-Wedgies*, que son igualmente estables pero no llevan al resultado primeramente esperado. Este es el objetivo de estudio de este proyecto, en el que se hace un estudio a fondo del protocolo BGP, así como una simulación y análisis de un escenario propenso a sufrir una *wedgie*, utilizando una herramienta de modelado llamada OMNeT++ y el paquete INET. En este último, ha sido necesario añadir algunas funciones imprescindibles para conseguir los objetivos buscados y que la implementación soportada del protocolo BGP sea más completa.

Palabras clave: BGP, Wedgies, encaminamiento, protocolo, OMNeT++,INET, simulación.

Abstract

Nowadays, communications over the Internet have a significant place in our everyday lives. For the purpose of carrying communications worldwide, it is necessary to have routing protocols that allow to guide traffic through the Internet. The most used exterior gateway protocol is *Border Gateway Protocol* (BGP), created for different interconnected domains to exchange routing information. This information is sufficient to build the connectivity network graph, in order to apply the suitable routing policies to each situation. BGP obtained results not always result in a deterministic state after applying a certain routing policy, but it is possible that unintended and non-deterministic states, called BGP-Wedgies, stables but unintended, appear. That is the purpose of this project, where a BGP protocol thorough study has been done, as well as a simulation and analysis based on a scenario prone to suffer a wedgie, using a modelling tool named OMNeT++ and the INET framework. The latter needed some implementation changes in order to get the intended objectives and to provide a more completed implementation of BGP .

Keywords: BGP, Wedgies, routing, protocol, OMNeT++, simulation.

Índice

Índice de figuras

Índice de tablas

Índice de abreviaturas

ABR – Area Border Router
AS – Autonomous System, Sistema Autónomo
ASBR – Autonomous System Border Router
BGP – Border Gateway Protocol
EBGP – Exterior Border Gateway Protocol
EGP – Exterior Gateway Protocol
IBGP – Interior Border Gateway Protocol
IDE – Integrated Development Environment
IGP – Interior Gateway Protocol
IP – Internet Protocol
ISP – Internet Service Provider
JRE – Java Runtime Environment
NED – Network Description
OMNET++ – Objective Modular Network Testbed in C++
OSPF – Open Shortest Path First
RFC – Request For Comments
RIP – Routing Information Protocol
XML – Extensible Markup Language

Capítulo 1

Introducción

1.1. Motivación

Internet crece cada vez más deprisa y se ha convertido en un elemento indispensable de nuestro día a día en los últimos 15 años. Es fácil pensar que este crecimiento continuará en los años venideros, sin embargo, los recursos con los que cuenta son limitados y, debido a la creciente demanda de servicios por parte del usuario, es necesario aprender a gestionar de una forma adecuada estos recursos.

Actualmente, Internet está conformada por miles de Sistemas Autónomos interconectados entre sí, gestionados por las principales operadoras de telecomunicaciones. Uno de los principales problemas que presenta el hecho de que compañías comerciales se encarguen de la distribución de tráfico en Internet es que este sistema va a carecer de algo imprescindible para un correcto funcionamiento: la cooperación, puesto que cada compañía suele anteponer su propio interés al bien común de la red. La comunicación entre estos Sistemas Autónomos se lleva a cabo mediante el protocolo Border Gateway Protocol (BGP), el protocolo de encaminamiento exterior más utilizado a nivel global actualmente. Una de las limitaciones que presenta BGP debido a la ya mencionada gestión de distintas compañías comerciales, es la posibilidad de conflicto debido a que cada Sistema Autónomo posee políticas de encaminamiento locales

elegidas por su administrador, lo que puede generar incongruencias a la hora de encaminar tráfico en Internet. Generalmente, en la práctica se utilizan otros protocolos para supervisar el funcionamiento de BGP.

Para poder gestionar correctamente los recursos y ofrecer un buen servicio es necesario estudiar a fondo el funcionamiento de este protocolo, así como los problemas que presenta, principalmente, las *BGP-Wedgies*, estados no deterministas pero estables que se presentan ante ciertas topologías de red y que necesitan de intervención del administrador para volver al estado esperado.

1.2. Objetivos

El objetivo más importante perseguido con este proyecto es principalmente el estudio a fondo del protocolo BGP, para conocer de primera mano cuál es su funcionamiento, en qué escenarios se aplica, cuáles son sus capacidades y cómo actuar en caso de tener una topología propensa a presentar *wedgies*. Para ello, ha sido necesario, además de estudiar la documentación, realizar una implementación de una red BGP con la ayuda de una herramienta de modelado, OMNeT++, la cual contiene librerías adecuadas para simular una red BGP, pero carente de otras funcionalidades básicas que ha sido necesario implementar para este estudio.

1.3. Estructura de la memoria

La memoria final de este proyecto está estructurada en capítulos, de la siguiente manera:

- **Capítulo 1:** capítulo actual en el que se hace una breve introducción en forma de motivación, se definen los objetivos del

proyecto y se hace una breve descripción de cómo está organizado el documento actual.

- **Capítulo 2:** capítulo 1 en Inglés.
- **Capítulo 3:** en este capítulo se aportan unas nociones básicas sobre encaminamiento en Internet, necesarias para comprender el tema que aborda este proyecto, el encaminamiento a través del protocolo exterior BGP.
- **Capítulo 4:** en este capítulo se introduce el protocolo BGP y se hace un estudio a fondo de su funcionamiento y características.
- **Capítulo 5:** se presenta y se explican las características y el funcionamiento del software utilizado para llevar a cabo la simulación y posterior análisis, además de las nuevas funcionalidades añadidas para poder llevar a cabo satisfactoriamente este proyecto.
- **Capítulo 6:** se presenta el escenario BGP utilizado en la simulación, así como ejecución y resultados de la misma.
- **Capítulo 7:** expone las conclusiones resultantes de este estudio.
- **Apéndice A:** incluye información paso a paso para realizar la instalación del framework de simulación utilizado.
- **Apéndice B:** contiene información para realizar la instalación del paquete de librerías necesario para el correcto funcionamiento del protocolo.
- **Apéndice C:** incluye todos los archivos de configuración desarrollados para llevar a cabo la simulación.

Capítulo 2

Introduction

2.1. Motivation

The Internet is growing faster and it has become an essential element in our everyday for 15 years. It is easy to think that this growth will increase in the coming years, however, the available resources are limited and, due to a growing demand of services it is necessary to learn how to manage these resources wisely.

Currently, the Internet consists of thousands of interconnected Autonomous Systems, managed by the main telecommunication companies. One of the main issues to face due to these big companies are distributing traffic is the fact that this method is lack of something essential for it to work properly: cooperation, since they only look for their own profit.

The routing protocol that makes possible the communication between these Autonomous Systems is the Border Gateway Protocol (BGP), the most used external gateway protocol globally nowadays. One of the main limitations of BGP due to the previously mentioned big commercial companies' management, is the possibility of conflict that may appear considering that each Autonomous System uses different and own policies chosen by its manager, what may result in contradiction when it concerns routing information. Generally, another protocols are used to supervise the proper functioning of BGP in practice.

With the aim of managing the provided resources properly and offer a good service is necessary to do a deep study about how this protocol works, as well as the problems it presents, principally, the *BGP-Wedgies*, non-deterministic but stable states that can appear in some network topologies and that need the manager intervention to go back to an intended state.

2.2. Objectives

The most important objective pursued with this project is to make a thorough study of the BGP protocol, in order to know first-hand how it works, what kind of scenarios it is applied to, what its capabilities are and how to act in case we have a network topology prone to present wedgies. In order to achieve that, besides studying the documentation, it has been necessary to make an implementation of a BGP network using a modelling tool, OMNeT++, that contains proper libraries to make a BGP simulation, although it does not provide some other basic features necessary for this study.

2.3. Structure of this report

This final report is organized into chapters, as follows:

- **Chapter 1:** first chapter, where a brief introduction in the form of a motivation and and a short description about how this report is structured are made and the project objectives are defined.
- **Chapter 2:** current chapter, where everything explained in the previous chapter is explained again, this time in English.
- **Chapter 3:** in this chapter, basic notions about Internet routing are provided, necesarry to understand the topic of this project, routing using the external gateway protocol BGP.
- **Chapter 4:** in this chapter, the BGP protocol is introduced and a deep study about its functionality and characteristics is made.

- **Chapter 5:** the software used for this simulation ana later analysis, as well as its functionality and characteristics are introduced and explained. Also, the new added features necessary to develop this project successfully.
- **Chapter 6:** the used BGP scenario is introduced, as well as its exeution and results.
- **Chapter 7:** this chapter contains the resulting conclusions of this project.
- **Appendix A:** this appendix includes step-by-step information about how to install the used simulation framework.
- **Appendix B:** it contains useful information to intall the necessary libraries to use the BGP protocol and others properly.
- **Appendix C:** includes all configuration files developed to run the simulation.

Capítulo 3

Nociones básicas sobre encaminamiento en Internet

3.1. La estructura actual de Internet

De una forma abstracta, podemos definir el Internet que conocemos hoy en día como un gran conjunto en el que cualquier dispositivo utilizado por un usuario final está conectado a un encaminador, que a su vez está conectado a otros encaminadores, los cuales forman un grafo conexo de encaminadores que cooperan entre sí utilizando protocolos de encaminamiento para intercambiar información y proveer conectividad a nivel global.

Esta visión abstracta de Internet supone que dicho grafo de conectividad contenga una gran cantidad de redundancias, las cuales serían soportadas por los protocolos de encaminamiento, diseñados para detectar cualquier fallo o problema en la red, con su consiguiente subsanación.

Sin embargo, esta idea no es del todo cierta. Lo que de verdad sucede es que el servicio de Internet es ofrecido por numerosas empresas de carácter comercial, que, generalmente, son competencia las unas de las otras. Lo que genera esta competencia entre empresas es que haya una carencia de cooperación, algo totalmente necesario a la hora de hablar de conectividad a nivel mundial en lo que a comunicaciones se refiere.

Una visión más cercana a la realidad es concebir la idea de Internet como un conjunto de proveedores de Internet, *ISP* (Internet Service Provider) de alguna manera cooperando entre sí para ofrecer una conectividad global al usuario final. El inconveniente que se presenta ante esto es que no todos los ISP son creados de la misma forma. Por ejemplo, unos son más grandes y están "más conectados" que otros, mientras que otros contienen rutas globales en sus tablas de encaminamiento. Los distintos ISP pueden clasificarse en los siguientes tipos:

- **Tier-3:** son aquellos proveedores que dan servicio de conexión a Internet a viviendas y numerosas empresas y no tienen un número demasiado elevado de usuarios finales. En España, pertenecen a esta clase ISP como *Movistar*, *ONO* o *Vodafone*, o la red académica y de investigación *RedIRIS*.

- **Tier-2:** generalmente tienen ámbito regional y necesitan "comprar tránsito" para alcanzar un destino dentro de la región a la que pertenecen. Normalmente se conectan a ISP del tipo Tier-1. Su función principal es ofrecer servicios de conexión a los ISP del tipo Tier-3. Por ejemplo, el AS3209, del tipo Tier-2 y perteneciente a Vodafone Alemania, compra tránsito a otros AS como el AS8928, perteneciente a la multinacional europea *Interoute*, quien a su vez compra tráfico al AS2914, propiedad de NTT Communications Corp., que tiene su sede en Japón. Cabe además destacar *Géant2*, que es la red que sirve a la comunidad educativa y de investigación en Europa, y *RedCLARA*, que ofrece el mismo servicio en Latinoamérica.

- **Tier-1:** pertenecen a este tipo los grandes proveedores globales (*Global Carriers*). Cualquier punto de Internet es accesible desde una red Tier-1 ya que todas las de este tipo han de estar conectadas entre sí. El conjunto de estas forman el *backbone* o troncal de Internet. Sus tablas de rutas contienen todos los prefijos de red alcanzables mundialmente. Algunos proveedores a nivel mundial que pertenecen a esta clase son *Cogent*, *Hurricane Electric*, *Seabone, TeliaSonera y Telefonica International Wholesale Services*.

Un esquema detallado de cómo se interconectan los diferentes *tiers* puede verse en la Figura 3.1.

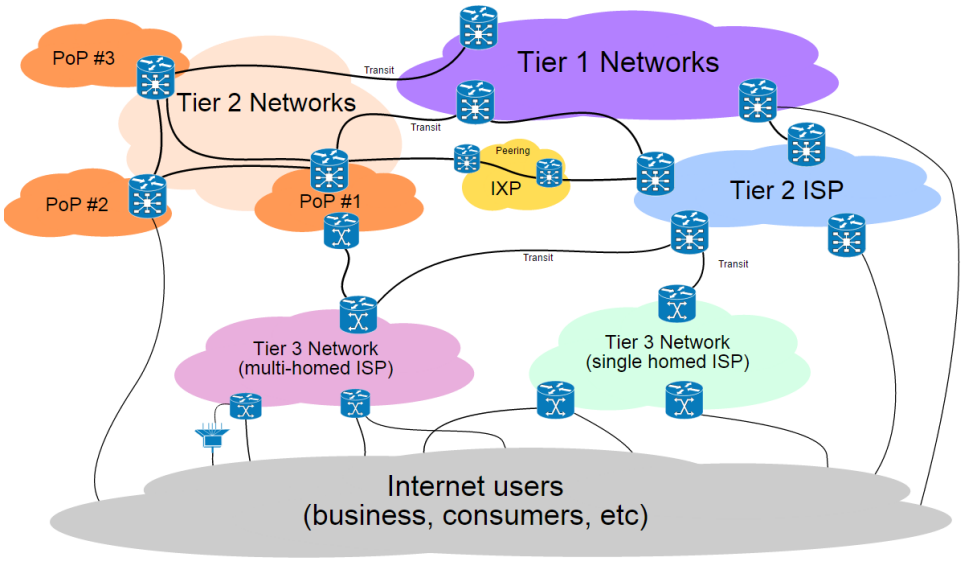


Figura 3.1. Relación entre los diferentes *tiers* de Internet

El protocolo de encaminamiento actualmente utilizado para intercambiar información de alcanzabilidad entre encaminadores en la frontera de estos proveedores de Internet, es BGP (*Border Gateway Protocol*), protocolo al que cuyo estudio está dedicado este proyecto. Más exactamente, la arquitectura de encaminamiento en Internet está dividida en Sistemas Autónomos, que intercambian esta información de alcanzabilidad.

Un **Sistema Autónomo** (AS) es un conjunto de redes IP bajo la autoridad de una o varias organizaciones con políticas de encaminamiento común, es decir, cada AS utilizará interiormente los protocolos de encaminamiento que considere necesarios y más apropiados para dirigir su tráfico. Cada AS está identificado por un Número de Sistema Autónomo (ASN) único, compuesto tradicionalmente por 16 bits, lo que ha permitido tener 65.536

asignaciones de ASN diferentes. A partir del año 2007, ante la posibilidad de agotamiento, se decidió extender los ASN a 32 bits.

Dentro de un AS, la información se intercambia a través de un protocolo de encaminamiento interno (IGP), entre los que destacan RIP (*Routing Information Protocol*), OSPF (*Open Shortest Path First*) o ISIS (*Intermediate System-Intermediate System*). La diferencia entre estos y los protocolos de encaminamiento externo (EGP) como BGP, es que mientras que los primeros están diseñados para optimizar la métrica de las rutas, los segundos están orientados a ofrecer información de alcanzabilidad y una implementación de políticas de encaminamiento de manera escalable.

Las direcciones IP necesarias para el encaminamiento de información en Internet, así como el número identificador de cada AS (ASN), son asignadas a los distintos proveedores por IANA (Internet Assigned Numbers Authority), quien es responsable además de DNS y otros recursos relacionados con el protocolo IP a nivel global.
Actualmente, existen dos tipos de direcciones IP en uso: IP versión 4 (IPv4) y IP versión 6 (IPv6). IPv4 se empezó a utilizar el 1 de Enero de 1983 y hoy en día es la versión más utilizada. Las direcciones IPv4 son números de 32 bits expresados como 4 octetos en notación de punto (por ejemplo 192.168.1.1). El lanzamiento de IPv6 se produjo en 1999, con direcciones de 128 bits convencionalmente expresados utilizando cadenas hexadecimales (por ejemplo, 2001:db8:ac10:fe11:1ab:ffe2::12ae). La aparición de IPv6 se debió al futuro agotamiento de las direcciones IPv4 disponibles, debido principalmente a la aparición de nuevos dispositivos que se conectan a la red y la rápida expansión de Internet. IPv6 supone un cambio no solo a nivel de tamaño de la dirección, sino que también se intenta mejorar la autenticación, seguridad, integridad y confidencialidad de la información transmitida, además de proveer una mayor facilidad a los encaminadores a la hora de leer su cabecera.

Estas direcciones IP (tanto IPv4 como IPv6) son generalmente asignadas de manera jerárquica. Un usuario obtiene direcciones IP de su proveedor de Internet (ISP) y este último las consigue a través del Registro de Internet Regional (RIR) que le corresponda. Cuando uno de ellos necesita más direcciones IP en su región, IANA hace una asignación adicional de direcciones para ser utilizadas por ese RIR en concreto. Estos RIR son divisiones de IANA a nivel continental y son los siguientes (Figura 3.2):

- **AFRINIC:** Región de África
- **APNIC:** Región de Asia – Pacífico
- **ARIN:** Región de Norte América
- **LACNIC:** Región de Latino América y Caribe
- **RIPE NCC:** Región de Europa, Oriente Medio y Asia Central

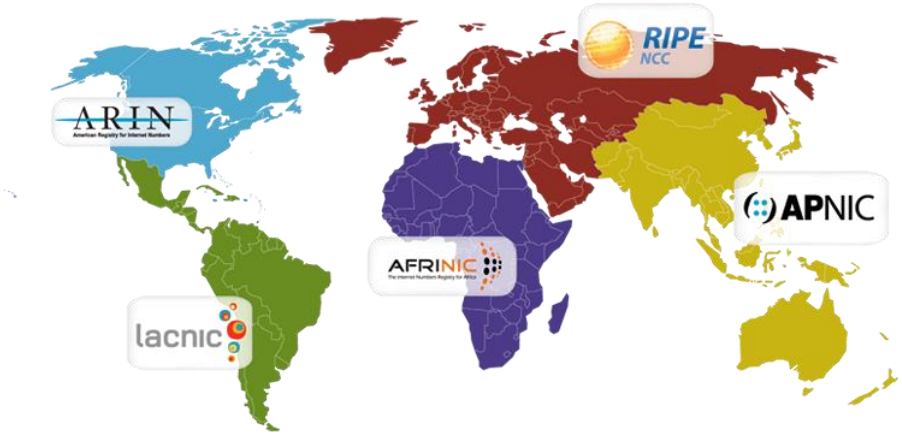


Figura 3.2. Registros Regionales de Internet a nivel global

Las bases de datos pertenecientes a cada RIR contienen detalles de registro de todas las direcciones IP y números de sistemas autónomos (ASN) asignadas a sus miembros. Esta información muestra qué recursos de Internet tienen asignados las organizaciones y particulares pertenecientes a un RIR en cuestión y detalles de contacto. Cada "participante" es responsable de mantener su propia información actualizada. Para obtener información sobre cualquier dominio del que es responsable uno de los RIR, podemos dirigirnos directamente a la base de datos pública de cualquiera de ellos y hacer una búsqueda, por ejemplo, de un ASN determinado (Figura 3.3). En caso de que el dominio introducido no pertenezca a ese RIR,

igualmente se mostrará la información asociada y se especificará a qué RIR pertenece (Figura 3.4).

3.2. WHOIS

WHOIS es una base de datos pública que mantiene información sobre los dominios que existen en Internet, tal como quién ha registrado ese dominio, direcciones IP, números de sistema autónomo, información de contacto y datos relevantes acerca del dominio. Esta base de datos está formada a partir de cada una de las bases de datos pertenecientes a los 5 RIR existentes.

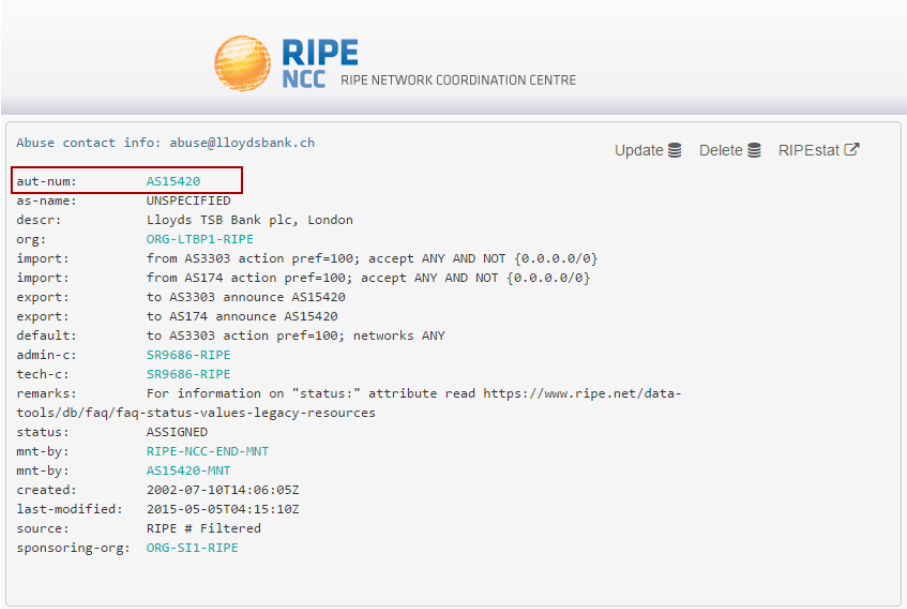


Figura 3.3. Resultados de la búsqueda del ASN15420 (perteneciente a la zona geográfica de RIPE) en la base de datos de RIPE NCC

El protocolo utilizado para realizar consultas sobre esta base de datos (que recibe el mismo nombre) opera en el puerto 43 y está definido en el **RFC 3912**.

Según el ICANN (*Internet Corporation for Assigned Names and Numbers*), WHOIS es utilizado para los siguientes propósitos:

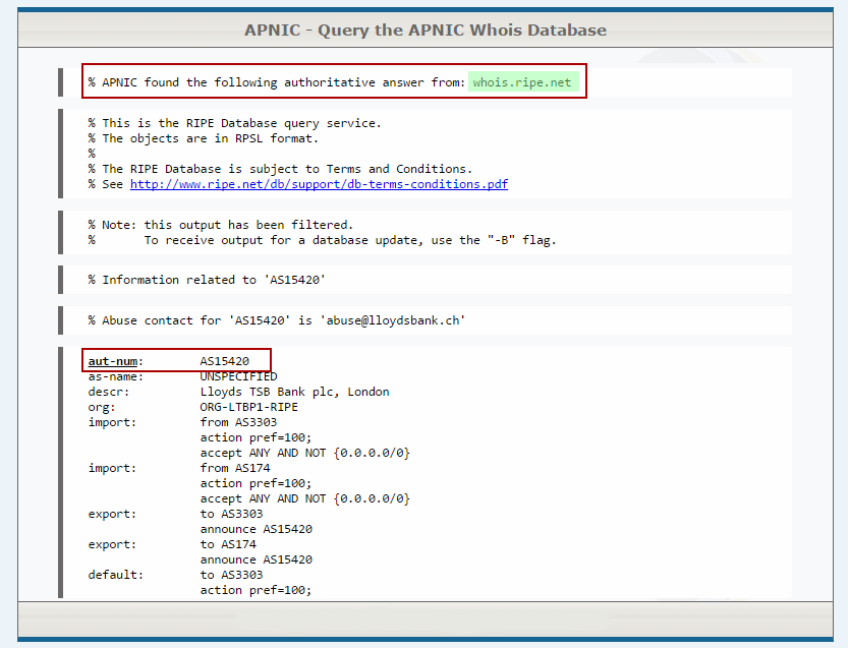


Figura 3.4. Resultados de la búsqueda del ASN15420 (perteneciente a la zona geográfica de RIPE) en la base de datos de APNIC

- Determinar si un dominio está disponible
- Contactar al administrador o administradores de una red para resolver aspectos técnicos relacionados con redes asociadas a un nombre de dominio (por ejemplo, DNS o aspectos de encaminamiento, origen y ruta de DoS y otros ataques...)
- Diagnosticar dificultades en el registro. Las consultas WHOIS ofrecen información útil para resolver conflictos como fechas de creación y expiración o la identidad del registro.
- Contactar con administradores web para resolver asuntos técnicos con un nombre de dominio
- Obtener una identidad real, localización e información de contacto de alguna persona u organización que tiene una presencia online
- Asociar una compañía o particular a un nombre de dominio

- Contactar al responsable del registro con el propósito de discutir y negociar alguna transacción relacionada con un nombre de dominio
- Notificar a un responsable de dominio que es su obligación mantener la información que aparece en la base de datos actualizada
- Contactar a un responsable para hablar temas relacionados con la protección o refuerzo de los derechos de propiedad intelectual
- Investigar una identidad en el ciberespacio, como respuesta a un ataque en Internet (los profesionales en seguridad utilizan WHOIS para identificar localizaciones u obtener información de contacto acerca de un nombre de dominio)
- Recoger pistas en una investigación. Por ejemplo, identificar implicados de los cuales se pueda obtener información relevante. Las agencias de seguridad utilizan WHOIS para encontrar direcciones de e-mail e intentar identificar la localización de presuntos delincuentes envueltos en casos de fraude.
- Investigar casos de SPAM recopilando información acerca de la web anunciada en los mensajes.

3.3. Tipos de protocolos de encaminamiento

Debido a que Internet se basa en una arquitectura formada por Sistemas Autónomos, es necesario diferenciar entre distintos tipos de protocolos de encaminamiento para establecer cómo se comunican entre sí y cómo se encamina el tráfico tanto en su interior como hacia el exterior. Existen 2 tipos de protocolos: de encaminamiento interior y de encaminamiento exterior.

3.3.1. Protocolos de Encaminamiento Interior

Conocidos comúnmente como *IGP* (Interior Gateway Protocol) o Protocolos de Pasarela Interna, son aquellos protocolos utilizados para la comunicación entre encaminadores y para resolver

el encaminamiento dentro de un mismo sistema autónomo. Este tipo de protocolos pueden dividirse en dos familias bien diferenciadas: vector-distancia y estado del enlace. En la Tabla 3.1 se hace un breve resumen acerca de sus diferencias.

- **Vector-distancia**: Este método se basa en la existencia de varios encaminadores conectados a distintas redes, los cuales van a intercambiar información entre ellos sobre destinos alcanzables dentro del sistema y a qué distancia se encuentran. Este tipo de encaminamiento está basado en el algoritmo de Bellman-Ford.

  Los protocolos pertenecientes a esta familia están basados en el intercambio de pequeñas cantidades de información. Cada entidad (encaminador o host) dentro de la red debe guardar información sobre todos los destinos del sistema.

  El nombre de vector-distancia viene de la posibilidad de encontrar rutas óptimas simplemente con el intercambio de distancias entre entidades.

  Algunos de los problemas que presentan este tipo de protocolos son la aparición de bucles, normalmente a causa de cambios en la topología (por ejemplo, un enlace desaparece y vuelve a reaparecer instantes después) y la escalabilidad limitada, ya que en redes muy grandes se invierte demasiado tiempo en la propagación de estas modificaciones. Dichos cambios no son detectados inmediatamente por los encaminadores vecinos, lo que provoca que en las tablas de encaminamiento pueda aparecer información incongruente. Por ejemplo, imaginemos que, un encaminador R1 que está funcionando y anunciando sus tablas de encaminamiento a los encaminadores vecinos, se apaga. Estos protocolos tienen configurado un temporizador que indica a los encaminadores que si, pasados x segundos (normalmente 30 segundos), no reciben anuncios de un encaminador se da por hecho que el enlace ha dejado de estar activo y hay que eliminarlo de la tabla de rutas. De esta manera, un encaminador conectado directamente a R1 (denominado R2) borra la entrada al haber desaparecido la conexión con R1, pero otro vecino (suponemos R3) puede que no haya borrado la entrada con la información del encaminador apagado y vuelva a anunciársela a R2, que la añadirá nuevamente al ser información que proviene de un vecino al que se encuentra conectado directamente. De esta forma,

R2 volverá a incluir la entrada en su tabla de encaminamiento y estará contenida en sus anuncios con una distancia mayor (puesto que aprende la ruta de uno de sus vecinos y no directamente como hacía antes). A su vez, R3 recibiría el anuncio proveniente de R2 y modificaría la entrada R1 con un valor igual a la distancia que le anuncia R2 más 1, siguiendo el mismo proceso, y así sucesivamente, incrementándose cada vez más el valor de la distancia. Este problema es conocido como *convergencia lenta* o *cuenta a infinito*. Para evitar alcanzar el "infinito" demasiado pronto, existen varios métodos aplicables a estos protocolos:

-**Valor máximo de "infinito":** se establece el infinito en 16 saltos, de manera que si aparece un valor igual o mayor para una distancia en una entrada, la red se consideraría como inaccesible.

-**Envenenamiento de rutas (*route poisoning*):** se utiliza para marcar como inalcanzable una red que se va a anunciar. Para ello, dicha red se pone a distancia máxima, 16, de manera que ningún encaminador la añadirá a su tabla de encaminamiento.

-**Horizonte dividido (*split-horizon*):** consiste en no anunciar las redes alcanzables aprendidas por un enlace a través de ese mismo enlace. Existe una variante, denominada *horizonte dividido con envenenamiento en reversa,* que consiste en anunciar las redes accesibles aprendidas mediante un enlace por ese mismo enlace, pero marcándolas con distancia infinita, es decir, con un valor de 16.

-**Actualizaciones provocadas (*triggered updates*)**: para acelerar la convergencia cuando se produce un cambio en la topología de la red, se utiliza el método de las actualizaciones provocadas, que consiste en transmitir un cambio en la topología en el momento en que sucede, sin esperar a que venza un temporizador.

El protocolo **RIP** (Routing Information Protocol) es uno de los protocolos más importantes dentro del grupo vector-distancia.

- **Estado del enlace**: En este método, cada encaminador mantiene una base de datos con la topología del AS al que pertenece,

es decir, contiene un mapa de conectividad de la red en forma de grafo, mostrando qué nodos están conectados con otros nodos. Esta base de datos contiene exactamente la misma información para todos los encaminadores. Independientemente, cada encaminador calcula las mejores rutas, y de ahí construye su propia tabla de rutas. No se comparten tablas de encaminamiento. Esta familia de protocolos presenta varias ventajas frente a los protocolos del tipo vector-distancia.

El procedimiento que siguen los encaminadores que están configurados bajo este tipo de procolos, logran cierta convergencia de la siguiente manera:

- Cada encaminador conoce las redes a las que puede acceder directamente.

- Se intercambian mensajes de "saludo" para saber qué encaminadores son sus vecinos.

- Cada encaminador crea un paquete con información sobre el estado del enlace (denominado LSP, Link State Packet), que incluye datos relevantes como información sobre los vecinos, ancho de banda disponible, tipo de enlace, etc.

-Este LSP, es enviado a todos los vecinos, que no solo almacenan la información contenida en él, sino que además lo reenvían al resto de vecinos, de manera que cuando finalice la transmisión de paquetes de este tipo, todos los encaminadores contengan exactamente la misma información de encaminamiento.

- Una vez repartida la información, cada encaminador crea un mapa con la topología de la red, a partir del cual cada uno decidirá cuál es la mejor ruta para encaminar su tráfico.

En cuanto a las desventajas de este tipo de protocolos, cabe destacar que requieren de más memoria para ejecutarse, así como más procesamiento de CPU y una alta utilización del ancho de banda disponible.

El protocolo de encaminamiento interior más utilizado junto a protocolos EGP, llamado **OSPF** (Open Shortest Path First), pertenece a esta familia.

	Vector-distancia	**Estado del enlace**
Topología	Cada encaminador ve la topología de la red desde la prespectiva de sus vecinos	Cada encaminador conoce la topología completa de la red
Encaminamiento	Se pasan las tablas de encaminamiento a los vecinos	Se pasan las actualizaciones del estado del enlace a todos los encaminadores
Actualizaciones	Se producen actualizaciones periódicas, lo que lleva a una convergencia lenta	Las actualizaciones se activan por eventos, lo que hace que la convergencia sea más rápida
Rutas	Añade vectores de distancia de encaminador a encaminador	Calcula la mejor ruta hacia otros encaminadores

Tabla 3.1. Diferencias entre procolos Vector-Distancia y Estado del enlace

3.3.1.1. RIP (Routing Information Protocol)

RIP es el protocolo de encaminamiento interior más antiguo y más sencillo y estable utilizado sobre todo en redes de tamaño reducido. Actualmente cuenta con dos versiones, definidas en los RFC 1058 (Versión 1) y RFC 2453 (Versión 2), además de una versión para IPv6 denominada RIPng, definida en el RFC 2080. Pertenece a la familia **vector-distancia.**

El funcionamiento de RIP se basa en el intercambio periódico de información sobre rutas entre encaminadores vecinos. Algunos encminadores envían estos mensajes cada 30 segundos, de manera que la red tenga todo momento información actualizada para poder adaptarse a los cambios rápidamente en caso de que se produzcan (encaminadores y hosts apareciendo y desapareciendo de la red). Los vectores de distancia incluyen la lista de destinos alcanzables desde

cada encaminador y la distancia a la que estos se encuentran (número de saltos). El mayor número de saltos permitido es 15, considerándose 16 o más como infinito y declarándose una ruta con tal número de saltos como inalcanzable.

Cuando un encaminador recibe información de un encaminador vecino con un camino más corto para alcanzar un tercer encaminador, el primero actualiza su tabla de encaminamiento con esa nueva distancia y ese encaminador como siguiente salto. Cualquier camino más rápido propagado entre encaminadores vecinos a través de este proceso tiene que llegar a toda la red RIP.

3.3.1.2. OSPF (Open Shortest Path First)

OSPF es un protocolo de encaminamiento interior basado en el Algoritmo de Dijkstra. Es el más utilizado debido a que no presenta problemas de convergencia lenta. Sus dos versiones están definidas en los **RFC 2328** (Versión 2, para IPv4) **y RFC 5340** (Versión 3, para IPv6). Pertenece a la familia **estado del enlace**.

El funcionamiento de OSPF se basa en calcular las rutas mínimas a partir de la topología completa de la red. Esto supone un coste muy elevado, incrementándose aún más si se suceden cambios en la topología. Para reducir este coste, se divide la topología en distintas áreas, que son básicamente un conjunto lógico formado por varios encaminadores. Existe un área "privilegiada", llamada área troncal o área 0, a la que el resto de áreas han de estar conectadas directamente y a través de la cual han de dirigir el tráfico obligatoriamente antes de llegar a su destino final. Si la topología no es dividida en áreas, todo el conjunto formará el área troncal.

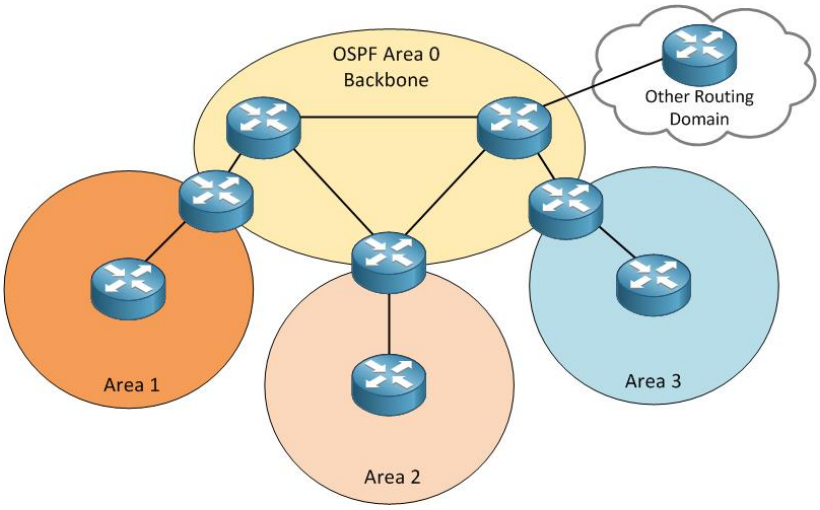


Figura 3.5. División en áreas de un Sistema Autónomo mediante OSPF

Dentro de este esquema, existen varios tipos de encaminadores, cada uno con una función determinada (Figura 3.6):

- **Encaminadores internos:** son aquellos encaminadores que pertenecen a un solo área. En caso de tener una topología no dividida en áreas, todos los encaminadores serán de este tipo.

- **Encaminadores troncales:** son aquellos encaminadores que pertenecen al área troncal o área 0.

- **Encaminadores frontera de área:** son los encaminadores que pertenecen a dos áreas: un área determinada y el área troncal. Son los encargados de establecer la conexión entre ambas áreas. Reciben el nombre de ABR (Area Borer Router).

- **Encaminadores frontera de sistema autónomo:** estos encaminadores reciben el nombre de ASBR (Autonomous System Border Router) y pertenece a esta clase cualquier encaminador que indica rutas no aprendidas mediante OSPF (por ejemplo, aprendidas por BGP).

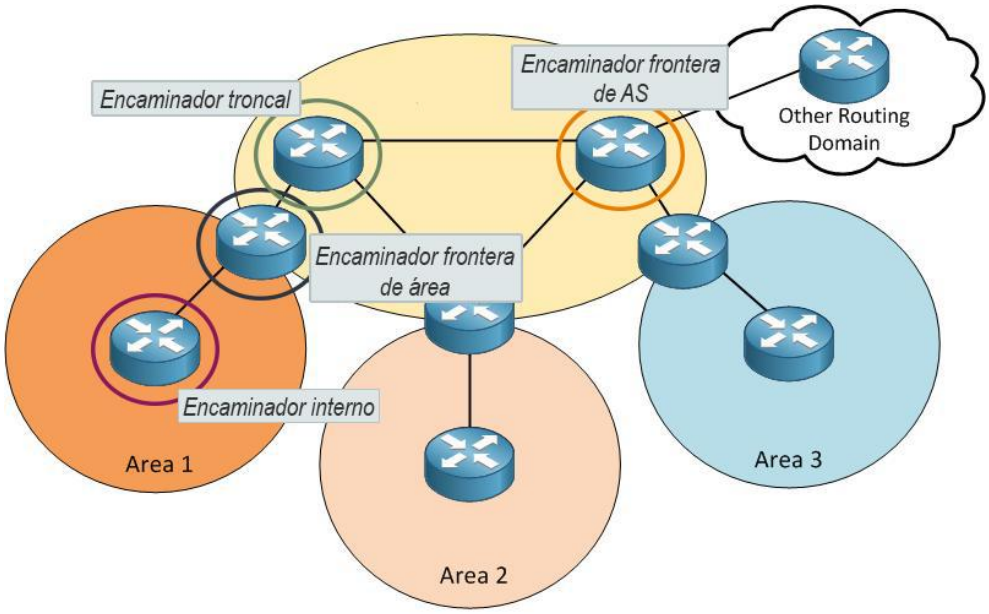


Figura 3.6. Tipos de encaminadores dentro del esquema de áreas

3.3.2. Protocolos de Encaminamiento Exterior

Los protocolos de encaminamiento exterior son utilizados para intercambiar información de encaminamiento entre distintos sistemas autónomos. Son conocidos como EGP (External Gateway Protocol) o Protocolos de Pasarela Externa.

El propósito de este tipo de protocolos es permitir que uno o varios sistemas autónomos sean utilizados como medio de transporte de tráfico originado en un sistema autónomo diferente y destinado a otro también diferente, de forma que todo el conjunto de sistemas autónomos formen un solo internet a vista del usuario final, quien podrá conocer que ruta sigue un datagrama enviado y el número de AS que este atraviesa para llegar a su destino (haciendo uso de la opción IP *source route*).

El protocolo EGP más conocido y utilizado en las comunicaciones hoy en día es BGP, objeto de estudio de este proyecto y del que se hace un estudio a fondo en el Capítulo 4.

3.4. Clasificación de métodos de encaminamiento

Los protocolos de encaminamiento proporcionan distintas formas para mantener las tablas de encaminamiento de una red, además de escoger la mejor ruta para alcanzar un destino. Para ello, existe el encaminamiento estático y el encaminamiento dinámico, cuyas diferencias aparecen detalladas en la Tabla 3.2.

3.4.1. Encaminamiento estático

El administrador de la red fija de forma estática las rutas por las que debe fluir el tráfico y las tablas de encaminamiento permanecen inalterables hasta que se vuelva a realizar un cambio manualmente.

Este tipo de encaminamiento suele presentar bastantes problemas por diferentes causas:

- **Errores humanos:** en la mayoría de los casos, al introducirse las rutas de forma manual, es muy fácil cometer un error en la configuración o en la información de la red.
- **Intolerancia a fallos:** el encaminamiento estático no es tolerante a fallos. Esto quiere decir que si se produce un fallo en la red por la causa que sea, el tráfico no es reenviado. A consecuencia de esto, hasta que el error no es reparado por el administrador, la red queda inútil.
- **Complejidad en la administración:** cuando se trata de configuración de rutas estáticas, es necesario hacerla en cada uno de los encaminadores que conforman la red. Esto puede llevar mucho tiempo y volverse un trabajo tedioso si la cantidad de encaminadores es elevada. Una configuración dinámica, sin embargo,

propagaría los cambios de rutas inmediatamente y sin necesidad de intervenir manualmente.

- **Prioridad ante rutas configuradas dinámicamente:** normalmente las rutas estáticas prevalecen sobre las rutas que han sido configuradas con un protocolo de encaminamiento dinámico, de manera que puede darse el caso de que estos protocolos no funcionen de la manera esperada debido a que se toma prioridad en las rutas establecidas manualmente.

3.4.2. Encaminamiento dinámico

En este tipo de encaminamiento, se utilizan diferentes protocolos para facilitar el intercambio de información entre los encaminadores de la red, tomando decisiones de manera dinámica y adaptándose a las condiciones de la red. Cuando un encaminador recibe información proveniente de otro, incorpora esa información a su propia tabla de encaminamiento.

Entre los propósitos que se buscan utilizando encaminamiento dinámico destacan los siguientes:

- Descubrir redes remotas
- Mantener la información de encaminamiento actualizada
- Elegir la mejor ruta hacia un destino
- Encontrar una ruta alternativa si la ruta actual deja de estar disponible

Una de las principales ventajas de este tipo de encaminamiento es que la información se intercambia cuando se produce un cambio en la topología de la red, lo que permite automáticamente al resto de encaminadores descubrir nuevas redes y encontrar rutas alternativas en caso de fallo.

En comparación con el encaminamiento estático, estos protocolos no son tan complejos en la parte de administración. En contraposición, utilizan muchos recursos, como la CPU y el ancho de banda.

Las familias de protocolos **vector-distancia** y **estado del enlace** pertenecen a este tipo de encaminamiento.

	Encaminamiento estático	**Encaminamiento dinámico**
Complejidad en la configuración	Aumenta con el tamaño de la red	Independiente del tamaño de la red
Nivel de conocimientos del administrador	No se requieren conocimientos avanzados	Se requieren conocimientos avanzados
Escalabilidad	Adecuada para topologías simples	Adecuada para topologías simples y complejas
Seguridad	Más seguro	Menos seguro
Cambios en la topología	Es necesaria la intervención del administrador	Se adapta automáticamente a los cambios en la topología
Consumo de recursos	No consume recursos adicionales	CPU, memoria y ancho de banda
Facilidad de pronóstico	La ruta que lleva al destino es siempre la misma	La ruta que lleva al destino depende de la topología actual de la red

Tabla 3.2. Comparación entre encaminamiento estático y dinámico

Capítulo 4

BGP: Border Gateway Protocol

4.1. Qué es BGP

BGP (Border Gateway Protocol) es un protocolo de encaminamiento externo entre los distintos Sistemas Autónomos que conforman Internet. La principal función para la cual es utilizado BGP es el intercambio de información de encaminamiento entre diferentes sistemas que hablan un idioma común: BGP. Esta información básicamente define cómo estos Sistemas Autónomos pueden alcanzarse entre sí, es decir, las rutas que puede seguir la información que viaja por Internet. Esto es suficiente para crear un grafo de conectividad de los Sistemas Autónomos existentes, pudiendo además permitir la eliminación de bucles y la toma de decisiones en cuanto a rutas se refiere.

BGP actualmente se encuentra en su versión 4 y está definido en el **RFC 4271**. Hoy en día, es el protocolo de encaminamiento externo más utilizado por los grandes proveedores de Internet (ISP).

4.2. Aspecto y funcionamiento básico de una red BGP

Una red con topología BGP, como podemos ver en la Figura 4.1., está compuesta por un conjunto de encaminadores, distribuidos en distintos sistemas autónomos y los cuales pueden establecer sesiones BGP tanto externas como internas entre ellos.

Una sesión BGP se considera **interna** (iBGP) si es establecida entre encaminadores que pertenecen al mismo sistema autónomo. Asimismo, se considera **externa** (eBGP) si se establece entre encaminadores pertenecientes a diferentes Sistemas Autónomos.

El carácter de estas sesiones es virtual, lo que quiere decir que son independientes del enlace o enlaces físicos que unen cada par de encaminadores. Por lo tanto, si una sesión falla, esto no tiene por qué suponer una caída del enlace o enlaces físicos.

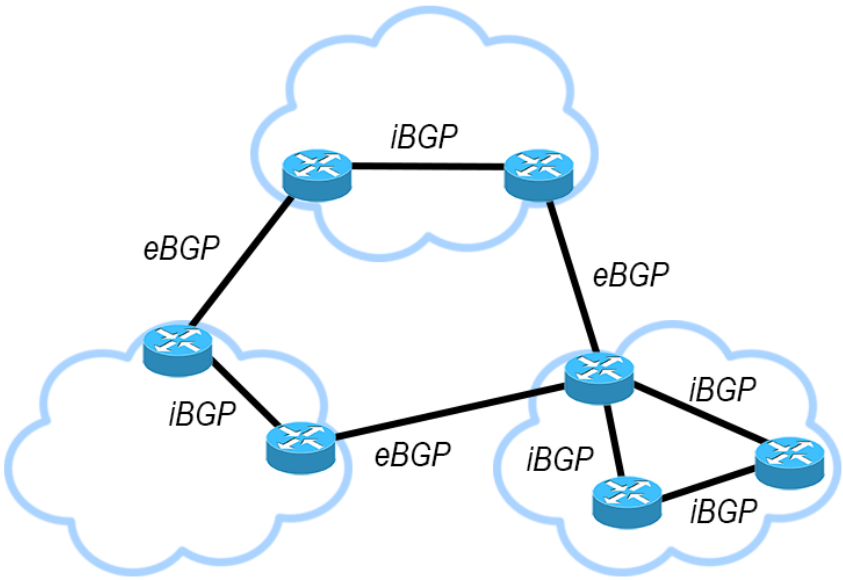


Figura 4.1. Red BGP

4.3. Objetivos de diseño

El diseño de BGP y su versión actual (BGP-4) surgió a raíz de tres necesidades esenciales:

- **Escalabilidad:** la división de Internet en Sistemas Autónomos bajo administración independiente se hizo mientras el *backbone* o troncal de Internet estaba bajo la administración de NSFNet (anteriormente había sido administrada por ARPANET). Un requisito imprescindible para BGP fue asegurar que la infraestructura de encaminamiento de Internet seguía siendo escalable según iba creciendo el número de redes interconectadas.

- **Políticas:** la habilidad de cada AS para implementar y reforzar políticas de encaminamiento fue uno de los objetivos clave a la hora de diseñar BGP. Algunas de las consecuencias directas de esto fueron la estructura de atributos para los anuncios y el filtrado de rutas.

- **Cooperación bajo circunstancias de competencia:** BGP fue diseñado en gran parte para manejar la transición de NSFNet a una situación en la que el *backbone* dejara de estar administrado por una sola entidad independiente. Esto derivó en permitir a los AS tomar decisiones de encaminamiento localmente para dirigir sus paquetes, de entre varias opciones.

4.4. Tipos de Sistemas Autónomos

Existen distintos tipos de Sistemas Autónomos, clasificados según la forma en la que se conectan con otros Sistemas Autónomos y cómo encaminan el tráfico (Figura 4.2):

- **Multihomed:** este tipo de AS está conectado directamente a dos o más AS. Esto les permite continuar conectados a Internet incluso si la conexión con uno de los AS falla.

- **Stub:** están conectados directamente a un solo AS. Pueden establecer conexiones privadas, pero a la vista de los demás solo mantienen una conexión con el resto de Internet.

- **Transit:** sirven de enlace entre otros dos AS y permiten el paso de tráfico a través de ellos. Por ejemplo, los ISP ofrecen a sus clientes el acceso a otras redes y a Internet a través de AS de tránsito.

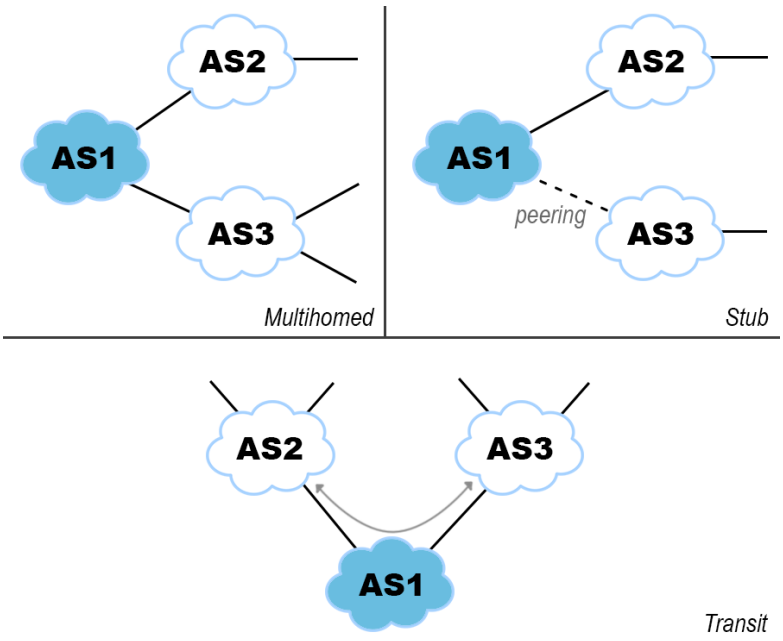


Figura 4.2. Tipos de Sistemas Autónomos

4.5. Relaciones entre Sistemas Autónomos

Internet está compuesto por diversos tipos de Sistemas Autónomos. Los más pequeños, como universidades o empresas, normalmente compran conectividad a otros ISP más grandes. Los

diferentes tipos de AS llevan a diferentes tipos de relaciones entre ellos, lo que a su vez deriva en diferentes políticas de encaminamiento, intercambio y selección de rutas. A la hora de configurar un Sistema Autónomo, es responsabilidad de la organización bajo la que este está administrado decidir con quién quiere establecer una conexión para poder intercambiar tráfico en Internet.

Existen dos tipos de relaciones que los AS pueden establecer entre sí: *peering* y tránsito (Figura 4.3):

- ***Peering***: es un tipo de conexión en el que dos AS se unen de forma voluntaria con el propósito de intercambiar tráfico de usuarios de las redes que conforman cada AS. Ambas partes se unen de mutuo acuerdo, con un fin común, sin pagar nada la una a la otra. Este tipo de relación requiere un conexión física de los dos AS implicados, además de un intercambio de información de encaminamiento a través de BGP.

  Las relaciones de *peering* suelen darse entre empresas de la competencia. Una de las principales razones por las que dos AS deciden mantener una relación de *peering* es evitar pagar a sus respectivos proveedores por el servicio y en su lugar, crean un enlace entre ellos para dirigir los paquetes de sus clientes directos. Es muy típico que los ISP de tipo Tier-1 establezcan este tipo de relaciones con otros ISP, generalmente del mismo tipo para obtener información de encaminamiento a nivel global.

- **Tránsito:** en este tipo de relación, un AS permite encaminar tráfico proveniente de otro AS y que va dirigido a una red de un tercer AS, es decir, actúa como un encaminador "de paso". El AS que contrata este tránsito, dispone de un servicio que le permite enviar y recibir una determinada cantidad de información (medida en Mbps) a un coste determinado. Si el AS contratante excede la cantidad de información acordada, esto implicará un cargo adicional en el coste.

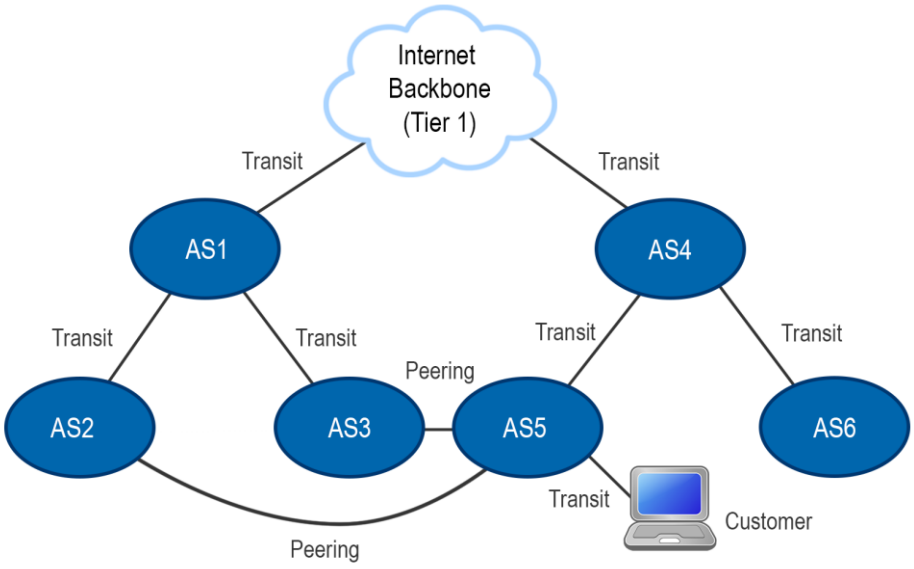


Figura 4.3. Relaciones típicas de *peering* y *transit* entre AS

4.6. Tipos de mensaje en BGP

Los encaminadores configurados bajo BGP, con el fin de implementar las diferentes funciones del protocolo, generan mensajes que son enviados en distintos instantes de tiempo. Generalmente, estos mensajes están controlados por temporizadores y algunos de ellos se envían como respuesta a un mensaje recibido de un vecino.

▪ **Mensaje OPEN: e**s el primer mensaje que se envía tras el establecimiento de la conexión TCP. Lo envía un encaminador frontera de Sistema Autónomo (ASBR) que quiere conectar con otro encaminador del mismo tipo. Se utiliza para intercambiar información de configuración y establecer unos parámetros comunes de encaminamiento. Este mensaje establece el comienzo de una sesión BGP entre dos encaminadores. Un mensaje OPEN cuenta con los siguientes campos:

- Versión del protocolo BGP (4 en el caso de la última versión)

- ASN al que pertenece el encaminador que envía el mensaje
- Hold Time, que indica el número de segundos que propone el encaminador que lo envía para expirar la sesión BGP
- Identificador BGP del encaminador que envía el mensaje
- Longitud de parámetros opcionales
- Parámetros opcionales

La longitud mínima de un mensaje OPEN es 29 Bytes (19 Bytes provenientes de la cabecera y 10 Bytes de los campos del mensaje). En la Figura 4.4. se muestra visualmente el formato de un mensaje OPEN.

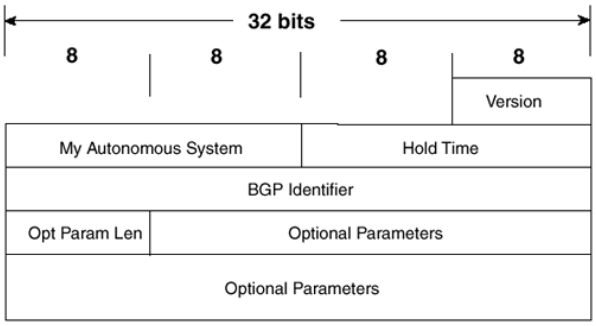


Figura 4.4. Formato de un mensaje OPEN

- **Mensaje UPDATE:** este mensaje se envía para difundir información de encaminamiento. Este tipo de mensaje puede ser utilizado para añadir nuevas rutas previamente anunciadas o para eliminar rutas ya existentes, de forma independiente o simultánea. Los campos de los que consta un mensaje UPDATE son:

- Longitud total del campo de *Rutas A Eliminar*, en Bytes.
- Lista de direcciones IP de las rutas que han de eliminarse de la tabla de encaminamiento correspondiente.
- Longitud total del campo *Atributos de ruta,* en Bytes.
- Secuencia de atributos de la ruta, de longitud variable, que ha de estar presente en todos los mensajes UPDATE, salvo en

aquellos que sólo llevan información sobre rutas a eliminar. Los atributos van codificados con un valor: Origin (1), AS_PATH(2), NEXT_HOP(3), MULTI_EXIT_DISC(4), LOCAL_PREF(5), ATOMIC_AGGREGATE(6) y AGGREGATOR(7).

La longitud mínima de un mensaje UPDATE es 23 Byes (19 Bytes de la cabecera y 4 de los campos del mensaje). Su formato viene mostrado en la Figura 4.5.

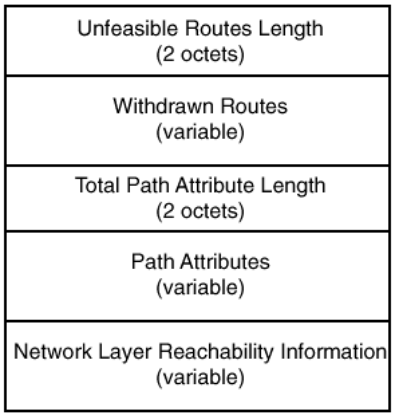
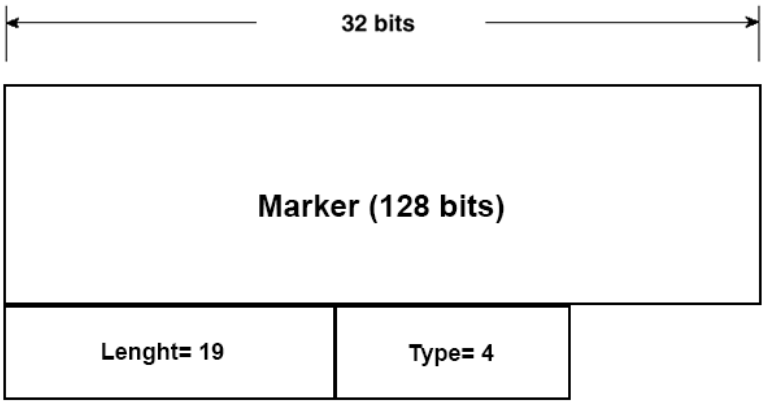


Figura 4.5. Formato de un mensaje UPDATE

- **Mensaje KEEPALIVE:** este tipo de mensaje se envía periódicamente para comprobar que el enlace punto a punto sigue activo. Estos mensajes son intercambiados para que no expire el *hold time*. Para ello, cada vez que se recibe un mensaje KEEPALIVE, *hold time* vuelve a inicializarse a 0. El valor para el temporizador KEEPALIVE se recomienda que sea 1/3 del valor del temporizador de *hold time*. Este mensaje no contiene datos, solo la cabecera, por lo que su longitud mínima es la misma que la de la cabecera: 19 Bytes. Su formato puede verse en la Figura 4.6.



4.6. Formato de un mensaje KEEPALIVE

- **Mensaje NOTIFICATION:** este mensaje se envía cuando BGP detecta que se ha producido un error. Después, se produce un cierre de la sesión BGP y de la conexión TCP y se eliminan todas las rutas de las tablas de encaminamiento. Los campos de este mensaje son:

  - Código de error, de longitud 1 Byte, que provee información más específica acerca del fallo que ha provocado el envío del mensaje (error de cabecera, error en el mensaje OPEN, error en el mensaje UPDATE, temporizador hold time expirado, error en la máquina de estados...)
  - Subcódigo de error, también de 1 Byte de longitud, que aporta más detalles acerca del propio error (conexión no sincronizada, longitud de mensaje no válida, tipo de mensaje no válido...)
  - Datos opcionales, de longitud variable, utilizados para diagnosticar la razón del envío de este mensaje. El contenido de este campo depende directamente de los campos anteriores.

La longitud mínima de un mensaje NOTIFICATION es 21 Bytes (19 Bytes de la cabecera y 2 de los campos del mensaje). Su formato puede verse en la Figura 4.7.

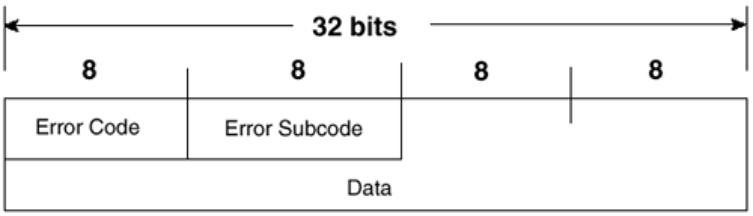


Figura 4.7. Formato de un mensaje NOTIFICATION

4.7. Decisiones de encaminamiento

BGP es un protocolo basado en políticas de encaminamiento (PBR, Policy Based Routing), es decir, proporciona al administrador del AS la capacidad de definir cómo fluirá el tráfico a través del AS mediante la configuración de una serie de atributos. El administrador del AS tiene total libertad para configurar estos parámetros, pudiendo así hacer ciertas rutas más o menos atractivas según sus intereses. Estos atributos son los que se encargan de dirigir el proceso de decisión cuando un encaminador recibe un anuncio de ruta a través de BGP. Existen 4 tipos de atributos:

- **Bien conocidos y obligatorios:** Son aquellos atributos que han de ser reconocidos por cualquier implementación BGP y aparecer en todos los mensajes UPDATE (o lo que es lo mismo, aparecer en cada entrada de información de encaminamiento). Si uno de estos atributos no se encuentra en el mensaje UPDATE, se debe enviar un mensaje NOTIFICATION al vecino BGP. A este tipo pertenecen los atributos *AS_PATH, NEXT_HOP y ORIGIN.*

- **Bien conocidos y discrecionales:** Estos atributos han de ser reconocidos por todas las implementaciones BGP pero no necesariamente han de aparecer en un mensaje UPDATE. A este tipo de atributos pertenecen *LOCAL_PREF y ATOMIC_AGGREGATE.*

- **Opcionales y transitivos:** Estos atributos no es necesario que sean reconocidos por todas las implementaciones BGP. Si aparecen en un mensaje UPDATE pero no son reconocidos por el encaminador que los

recibe, son reenviados al siguiente vecino BGP sin sufrir modificaciones. Los atributos AGREGATOR y COMMUNITY pertenecen a esta clase.

▪ **Opcionales y no transitivos:** Estos atributos no es necesario que sean reconocidos por todas las implementaciones BGP, pero tanto si son reconocidos como si no lo son, una vez recibidos, no es necesario que sean transmitidos al vecino. Simplemente, el encaminador que los recibe los descarta.

4.7.1. Atributos en BGP

Como ya se ha mencionado, BGP cuenta con unos atributos que son incorporados a los mensajes UPDATE y necesarios para tomar una decisión sobre el encaminamiento de tráfico. Los más importantes son los siguientes:

▪ **ORIGIN:** este atributo indica a todos los AS dónde se originó la ruta. Es obligatoria su aparición en todos los anuncios de ruta que haga un encaminador. El origen de una ruta puede ser:

- IGP: la ruta ha sido aprendida por un protocolo de encaminamiento interior, por lo tanto, su origen está en el propio AS.
- EGP: la ruta ha sido aprendida por encaminamiento externo, por lo que su origen está en un AS distinto.
- Incomplete: la ruta proviene de un protocolo que no es BGP.

▪ **AS_PATH:** este atributo contiene la secuencia de ASN que la información ha de atravesar para llegar a su destino, los AS por los que está formado una ruta, ordenados. Que una ruta esté formada por un número reducido de AS no quiere decir que sea la más corta, ya que no es posible saber qué número de encaminadores hay dentro de un AS.

Si un encaminador genera una ruta, el AS_PATH de la misma sólo contendrá 1 valor y este será el ASN del encaminador que la ha generado.

Si un encaminador anuncia una ruta a otro y esta no tiene origen en él mismo, añade su ASN al AS_PATH existente para esa ruta.

Si un encaminador recibe una ruta en cuyo AS_PATH aparece su ASN, se descarta, ya que es una evidencia de la aparición de un bucle.

- **AS4_PATH:** este atributo es opcional y transitivo y contiene el AS_PATH codificado con 32 bits. Tiene exactamente la misma semántica y la misma codificación que AS_PATH, salvo que es opcional y transitivo y que lleva números de 32 bits en lugar de 16 bits.

- **NEXT_HOP:** este atributo contiene la dirección IP del siguiente encaminador frontera de AS en la ruta. Si se recibe por eBGP, NEXT_HOP tendrá el valor del encaminador que lo manda. Si se recibe por iBGP, NEXT_HOP sigue manteniendo el valor de la última IP actualizada. Este atributo permite distinguir la topología BGP de la topología física, ya que no necesariamente el siguiente nodo BGP tiene que ser el siguiente nodo físico.

- **MULTI_EXIT_DISC:** este atributo se utiliza cuando dos AS están conectados por más de una ruta, con varios encaminadores ASBR conectados entre sí. La función de este atributo consiste en seleccionar una ruta cuando se reciben por eBGP varias rutas iguales provenientes del mismo AS por enlaces distintos. Por lo tanto, el administrador del AS que genera estas rutas, si quiere sugerir una ruta preferida, le asignará a un valor menor al MULTI_EXIT_DISC de esta, ya que un valor bajo para este atributo tiene prioridad a la hora de elegir una ruta.

- **LOCAL_PREFERENCE:** este atributo es utilizado para decirle a otros encaminadores dentro de un AS cómo salir de este, en caso de existir más de una ruta. Se prefiere un valor alto de este atributo frente a uno bajo. Solo se propaga mediante iBGP, por lo que únicamente es válido entre encaminadores BGP que pertenecen al mismo AS. Es el opuesto al atributo MULTI_EXIT_DISC.

- **ATOMIC_AGGREGATE:** es un atributo bien conocido utilizado cuando un encaminador agrega varias rutas con el propósito de anunciarlas a un encaminador en concreto. El AS_PATH del

agregado de rutas normalmente incluye un AS_SET formado a partir del conjunto de AS desde los que se formó el agregado. En algunos casos, el administrador de la red puede determinar si el agregado es anunciado sin el AS_SET, solo en el caso de ser seguro y no provocar bucles.

Cuando un encaminador BGP recibe un mensaje UPDATE que contiene ATOMIC_AGGREGATE no debe eliminar el atributo cuando propaga el anuncio.

- **AGGREGATOR:** es un atributo opcional transitivo, que puede ser incluido en los mensjes UPDATE que contienen rutas que se formaron por agregado. Contiene información sobre el encaminador que agregó la ruta, como su ID y ASN.

- **AS4_AGGREGATOR:** mismo atributo que Aggregator para ASN de 32 bits.

El proceso de decisión se basa en los atributos anteriormente descritos y se inicia una vez se han recibido anuncios de varias rutas para un mismo destino. Cuando esto sucede, los anuncios recibidos son listados en orden inverso a su llegada. Es decir, la ruta más nueva aparece en primer lugar, mientras que la más antigua aparece la última. Para seleccionar la mejor ruta para un destino determinado, BGP utiliza un método de comparación secuencial. Por ejemplo, si se reciben 3 rutas diferentes para un mismo destino (1, 2 y 3), primero se hará una comparación entre 3 (recibida en último lugar y más nueva) y 2. La mejor ruta resultante de estas dos será comparada con 1 (recibida en primer lugar y más antigua). La ruta preferida de esta última comparación será la anunciada e incluida en las tablas de rutas.

Este proceso de selección de ruta consta de 13 pasos, cada uno de los cuales es evaluado secuencialmente hasta que se encuentra una ruta preferente:

1.- **WEIGHT** es el primer parámetro considerado. Se prefiere la ruta con un valor mayor de WEIGHT. WEIGHT es un parámetro propiedad de Cisco y es local dentro del encaminador en el que este

es configurado. Por defecto, las rutas originadas localmente tienen un valor de WEIGHT de 32768 mientras el resto de rutas tienen un valor de 0.

2.- Se prefiere la ruta con un valor de **LOCAL_PREF** mayor. El valor por defecto para este atributo es 100.

3.- Las rutas son evaluadas basándose en su origen, teniendo preferencia aquellas que son **originadas localmente**.

4.- Se da prioridad a las rutas que tienen un **AS_PATH** más corto. Sin embargo, a la hora de configurarse, se puede ignorar este paso.

5.- El siguiente atributo que se evalúa es **ORIGIN**, siendo preferidas aquellas rutas que tengan el valor más bajo. IGP es menor que EGP, y EGP es menor que INCOMPLETE.

6.- Se evalúa el atributo **MED**, eligiendo la ruta con un valor de este menor.

7.- Las rutas aprendidas por **eBGP** son preferidas frente a las aprendidas mediante iBGP.

8.- BGP prefiere la ruta que tenga la métrica **IGP más baja** al salto siguiente. Esta comparación toma en consideración la topología.

9.-Si ***multipath*** está activado, con un valor n entre 2 y 6, y exiten varias rutas con el mismo coste(los valores obtenidos de las comparaciones 1-6 son idénticos y poseen exactamente el mismo AS_PATH), BGP inserta estas n rutas en la tabla de rutas. El valor por defecto de n, cuando *multipath* no está activado, es 1.

10.- Cuando ambas rutas son recibidas por eBGP, se prefiere aquella que se recibió primero (**la más antigua**).

11.- Se prefiere la ruta que venga del encaminador BGP con un **ROUTER_ID** menor.

12.- Si el encaminador es el mismo para varias rutas, se prefiere aquel que contenga una **CLUSTER_LIST** más corta.

13.- BGP se decide por el encaminador vecino que tenga una **dirección IP menor**.

4.8. Filtrado de rutas

El filtrado de rutas es la base sobre la que se asientan las políticas de BGP a la hora de encaminar tráfico. El hecho de aceptar o no tráfico con un determinado origen, evitar que el tráfico atraviese un determinado AS o "manipular" los atributos BGP para seguir cierta trayectoria hacen necesario e imprescindible este filtrado. Pueden filtrarse pefijos, ASN, atributos y anuncios, con los siguientes comandos, respectivamente: *prefix-list*, *filter-list*, y *route-map*.

- **Prefix-list:** es una forma eficiente y muy rápida de filtrado, así como fácil de configurar y usar. Se basa en el filtrado de prefijos para aceptar o rechazar una ruta.

- **Filter-list:** restringe las rutas que serán anunciadas o aceptadas basándose en su AS_PATH.

- **Route-map:** se utiliza para ignorar ciertas rutas y, por consiguiente, evitar su anuncio basándose en la coincidencia de valores de distintos criterios de configuración.

4.9. iBGP

iBGP (Internal BGP) es un mecanismo utilizado para transmitir información entre encaminadores pertenecientes a un mismo Sistema Autónomo. La idea de utilizar iBGP y no otro protocolo de encaminamiento interior (IGP) es que no se pierde la información relacionada con los atributos eBGP porque iBGP los reenvía. Para evitar bucles, si una ruta es aprendida por iBGP, no será anunciada a otro vecino iBGP. Para que todos los encaminadores frontera de un AS (que a su vez están conectados a otros encaminadores frontera de AS diferentes a través de eBGP) aprendan todas las rutas, es necesario que todos ellos estén conectados entre sí mediante sesiones iBGP, lo que se conoce como ***full-mesh***.

En redes grandes formadas por un número alto de encaminadores, aplicando full-mesh, el número de sesiones BGP crece de forma cuadrática y se genera tráfico duplicado innecesario. Para evitar estos problemas existen dos alternativas de configuración: *reflectores de rutas* y *confederaciones* (Figura 4.8).

4.9.1. Reflectores de rutas

Con el fin de evitar sobrecarga y bucles en la topología de la red, es posible utilizar un mecanismo en el que se declara *reflector de rutas* a uno de los encaminadores frontera del AS, siendo el resto de encaminadores frontera del mismo AS sus *clientes.*

El conjunto formado por el reflector de rutas y todos sus clientes se denomina *cluster.*

La política de reenvío de rutas que sigue un reflector de rutas cuando recibe un anuncio de un encaminador vecino, se detalla en la Tabla 4.1.

Recibida por/reenviada a	eBGP	Todos los clientes del AS	Todos los no-clientes del AS
eBGP	–	✔	✔
Cliente	✔	✔	✔
No-cliente	✔	✔	–

Tabla 4.1. Política de reenvío de rutas en un Reflector de Rutas

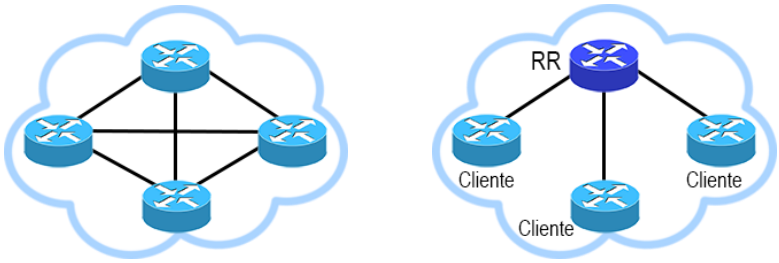


Figura 4.8. Configuración full-mesh y reflector de rutas

4.9.2. Confederaciones

Las confederaciones tienen el mismo propósito que los reflectores de rutas, siendo estas una colección de sistemas autónomos anunciados como uno solo al resto de encaminadores BGP que no forman parte de la confederación. Cuando varios AS son agrupados en una confederación, poseen un único número de identificación común, que será anunciado al exterior, pero de cara al interior cada uno consta de su propio ASN (Figura 4.9). Dividir un AS en dominios más pequeños puede ser muy útil si este consta de un número elevado de encaminadores BGP, de forma que sea más sencillo controlar la política de rutas de este (aunque esta puede volverse muy compleja), además de que se reduce considerablemente el número de conexiones BGP intradominio. A pesar de esto, el hecho de subdividir un AS innecesariamente puede, como ya se ha mencionado, incrementar la complejidad de las políticas de encaminamiento, además de hacerse mucho más difícil la coordinación con otros AS no perteneciente a la confederación. Si no se proporciona una correcta configuración puede aparecer información duplicada, lo que resultará en gasto innecesario de los recursos del sistema, solapamiento de rutas y problemas en la convergencia.

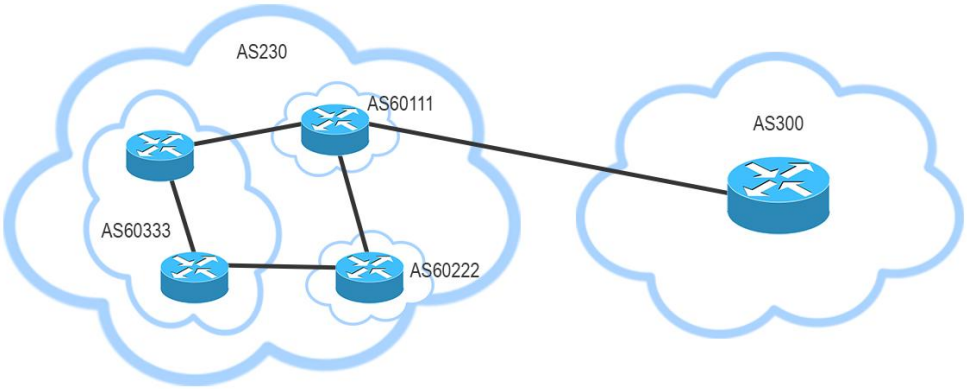


Figura 4.9. Esquema de una confederación BGP

4.10. Wedgies

BGP es una herramienta encargada de distribuir información de encaminamiento y crear rutas de una forma determinista. Existe una clase de configuraciones BGP para las cuales hay más de un estado final posible, y donde otros estados finales distintos del esperado son también estables, pero son alcanzados de forma no determinista. Estos estados estables pero inesperados reciben el nombre de BGP Wedgies.

4.10.1. ¾ Wedgies

Para explicar en qué consiste una *¾ wedgie* es necesario basarnos en un caso concreto. . En el ejemplo utilizado (Figura 4.10), AS1 anuncia sus prefijos a AS2 como ruta de "Backup only", es decir, como enlace secundario en caso de fallo, y a AS4 como enlace primario. AS4 anunciará los prefijos de AS1 a AS3. AS3 recibirá los anuncios de AS4 a través de un enlace de *peering* y seleccionará los prefijos de A1 con la ruta {AS4, AS1}. AW3 anunciará estos prefijos a AS2. AS2 recibirá dos rutas con prefijos de AS1, la primera a través de la conexión directa con AS1 y la segunda a través de la ruta {AS3, AS4, AS1}. AS2 preferirá la ruta más larga, ya que la ruta directa está marcada como secundaria, y la preferencia local de

AS2 preferirá el anuncio recibido mediante AS3 que el recibido mediante AS1 (Figura 4.11).

Este es el resultado esperado, donde en un estado "normal" no fluye tráfico desde AS2 hacia AS1 a través del enlace de backup y AS2 alcanza AS1 a través de la ruta entre AS3 y AS4, usando el enlace primario hacia AS1.

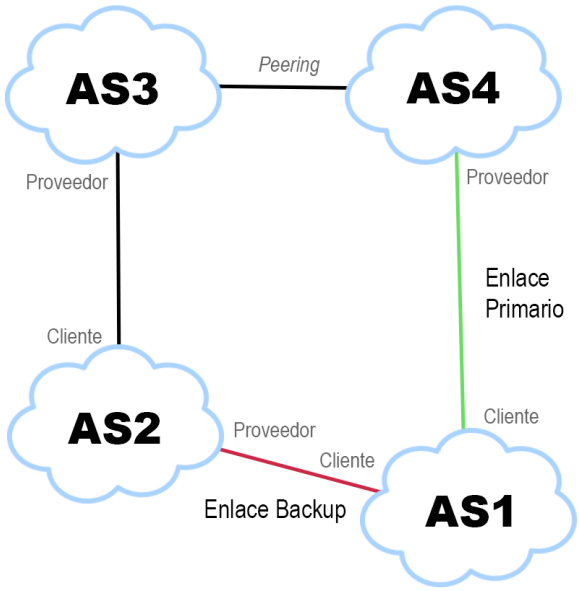


Figura 4.10. Topología de un escenario propenso a sufrir una ¾ *wedgie*

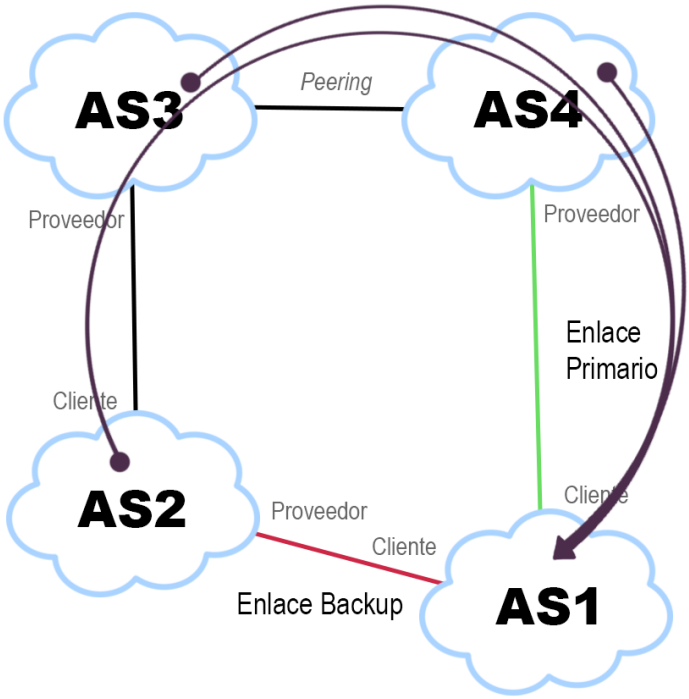


Figura 4.11. Elección del enlace primario ante el enlace de backup

El resultado esperado sólo se consigue si AS1 anuncia sus rutas por el enlace primario a AS4 antes de anunciar sus rutas por el enlace de backup a AS2. Si el enlace primario AS1 – AS4 se cae, provocando un fallo en la sesión BGP entre AS1 y AS4, entonces AS4 retirará su anuncio de rutas de AS1 a AS3, quien a su vez, retirará su anuncio hacia AS2. Al desaparecer el anuncio de ruta hacia AS1 por el enlace primario, AS2 elegirá el enlace de backup como ruta hacia AS1 y la anunciará hacia AS3, quien también la anunciará hacia AS4. Por lo tanto, todo el tráfico dirigido hacia AS1 utilizará el enlace de backup. (Figura 4.12). Este es el estado esperado según la política de decisión de BGP.

Cuando la conexión entre AS4 y AS1 se restablece, no se volverá al estado original. AS4 aprenderá la ruta hacia AS1 por el enlace primario y volverá a enviar el anuncio a AS3 usando la ruta {AS4, AS1}. AS3, aplicando la preferencia por defecto de preferir las rutas anunciadas por un cliente sobre las anunciadas mediante

peering, seguirá escogiendo la ruta {AS2, AS1} para alcanzar AS1(Figura 4.13).

Por lo tanto, AS3 no enviará ninguna actualización de ruta a AS2. Después del restablecimiento del enlace primario, el tráfico desde AS3 a AS1 y desde AS2 a AS1 atravesará el enlace de backup para llegar a su destino, a pesar de que el enlace primario está operativo de nuevo.

El estado esperado solo puede ser restablecido si AS1 desactiva deliberadamente su sesión eBGP con AS2, aunque esté fluyendo tráfico por ese enlace (Figura 4.14).

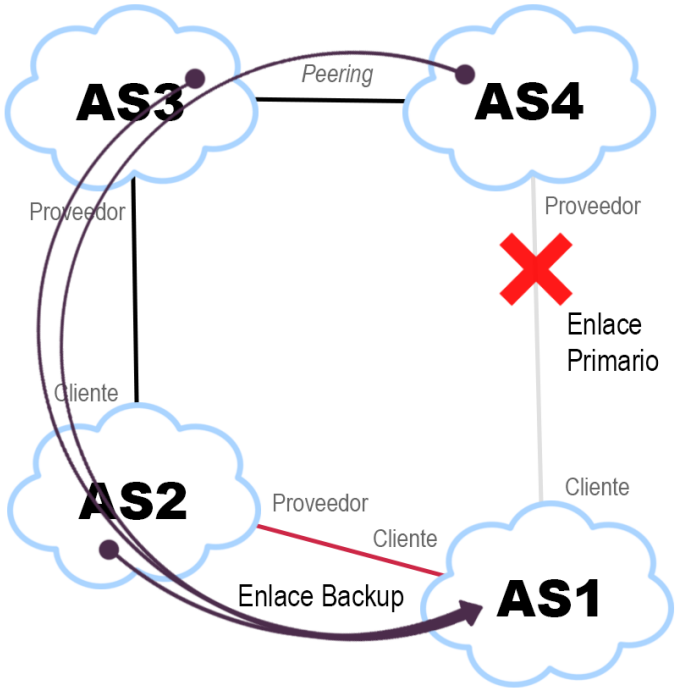


Figura 4.12. Caída del enlace primario y selección de rutas por el enlace de backup

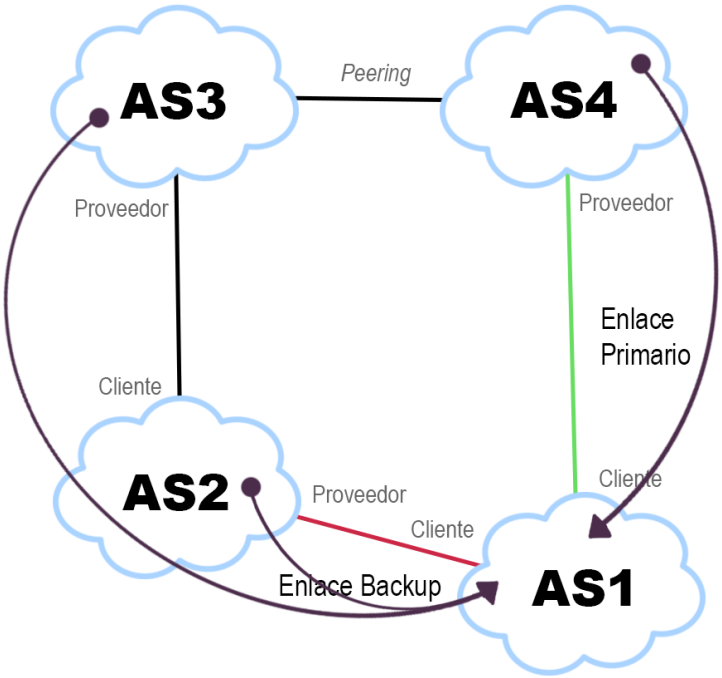


Figura 4.13. Restablecimiento del enlace principal

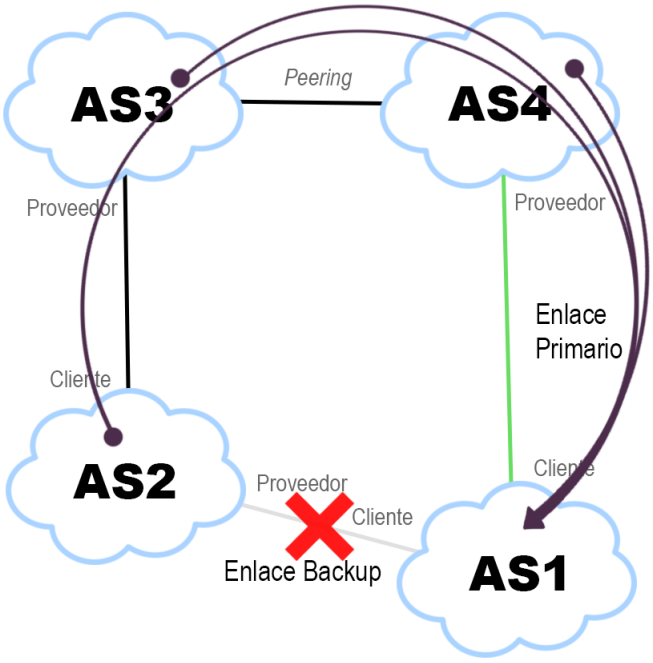


Figura 4.14. Estado esperado después de tirar intencionadamente el enlace de backup

4.10.2. Full Wedgies

En el ejemplo utilizado para explicar una *full wedgie* (Figura 4.15), AS2 y AS5 son ambos proveedores hacia AS1 con un enlace backup cada uno, y AS4 es el proovedor principal. El flujo de tráfico esperado, sería elegir las rutas que pasan por el enlace principal para alcanzar el destino, dejando de lado los enlaces backup (Figura 4.16).

Cuando la conexión entre AS1 y AS4 se cae (Figura 4.17) y es posteriormente restablecida (Figura 4.18), AS3 continuará dirigiendo su tráfico hacia AS1 a través de AS2, que es el enlace principal de backup.

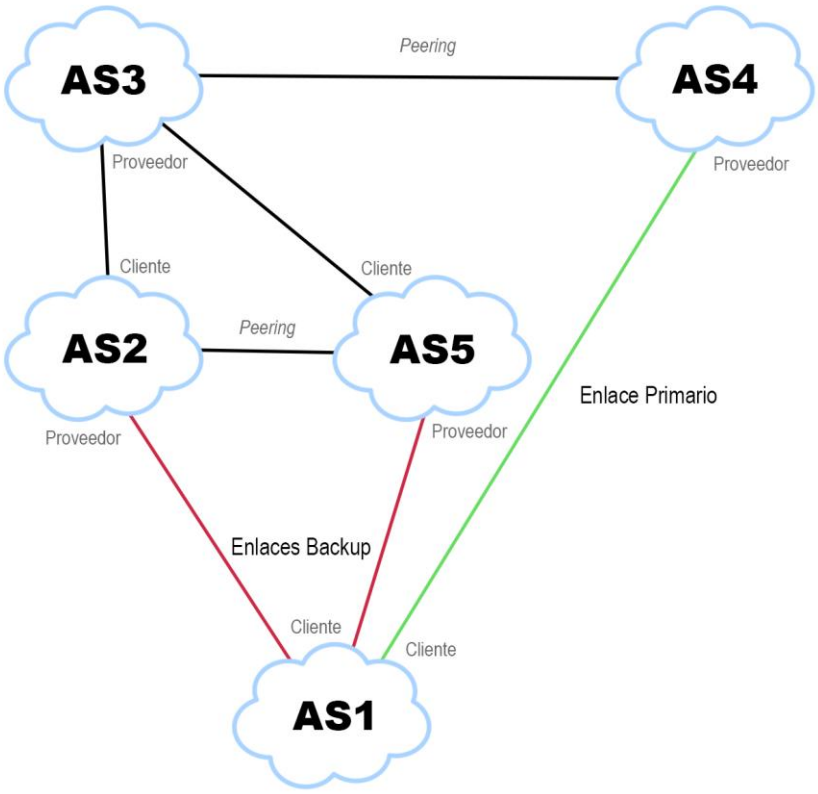


Figura 4.15. Topología de un escenario propenso a sufrir una *full wedgie*

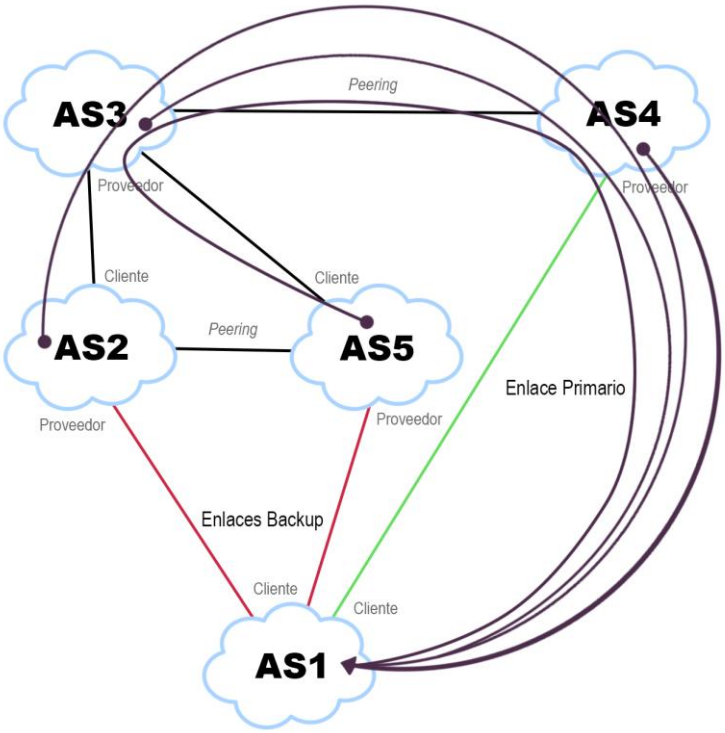


Figura 4.16. Flujo esperado de tráfico utilizando el enlace principal

En este caso, restablecer una única vez el enlace entre AS2 y AS1 no hará volver al estado inicial esperado, ya que la mejor ruta hacia AS1 según la política de decisión establecida será entonces AS5, que se comporta como enlace secundario de backup, y AS2 y AS3 aprenderán una ruta hacia AS1 a través de AS5 (Figura 4.19).

La acción requerida para corregir esta situación es reiniciar simultáneamente ambos enlaces de backup hacia AS2 y AS5 (Figura 4.20). Esta no es la solución más intuitiva y obvia, ya que en algún instante sólo uno de estos enlaces llevará tráfico de backup y además es necesario reiniciar ambos enlaces al mismo tiempo para volver al estado esperado.

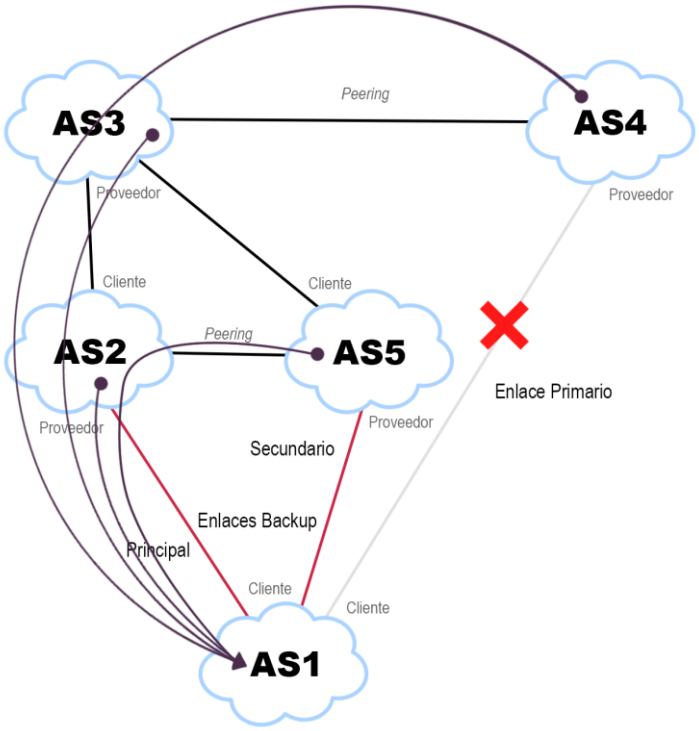


Figura 4.17. Caída del enlace primario y elección del enlace principal de backup hacia AS1

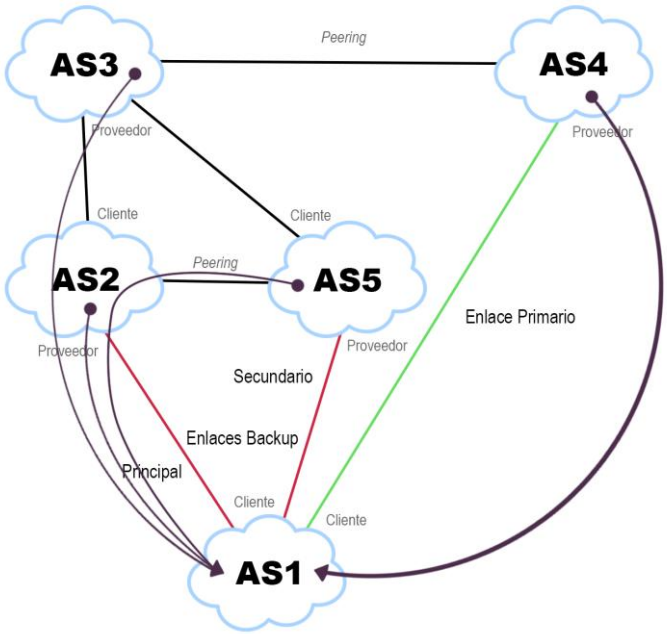


Figura 4.18. Restablecimiento del enlace primario

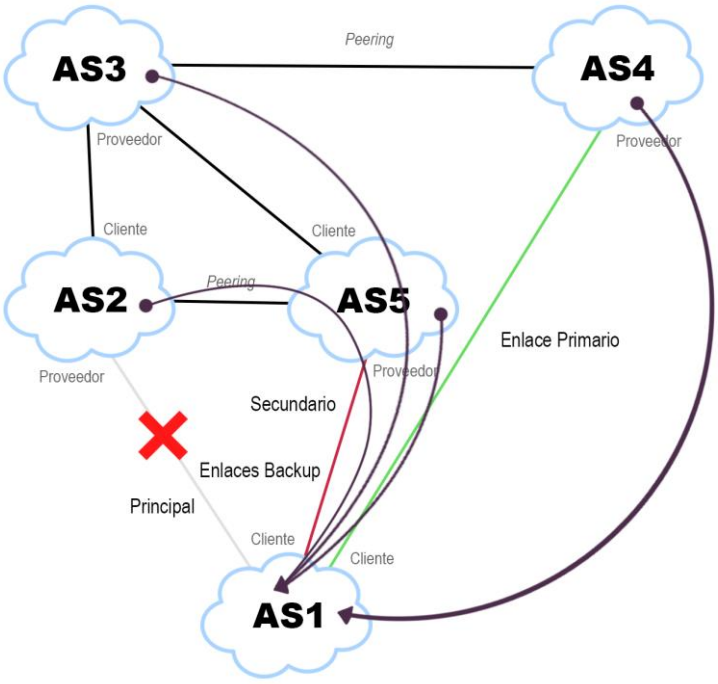


Figura 4.19. Caída del enlace principal de backup después de intervención manual

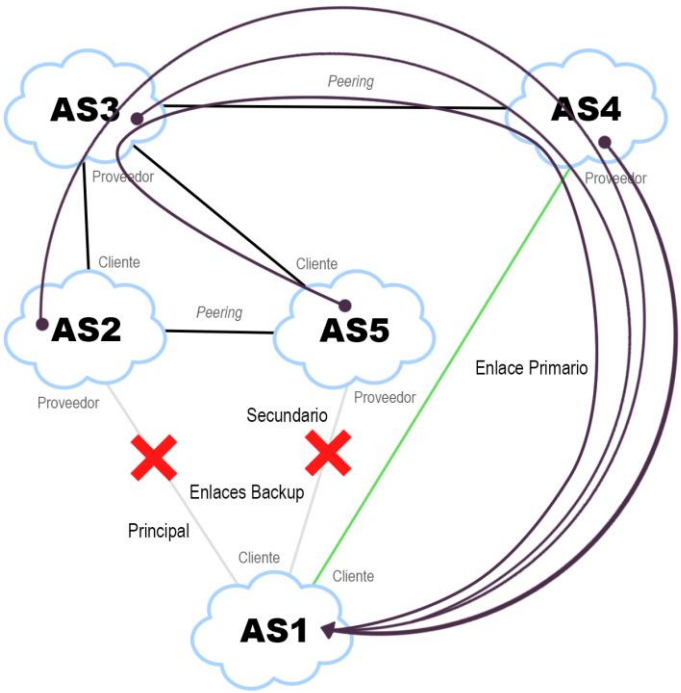


Figura 4.20. Caída simultánea de ambos enlaces de backup tras intervención manual

Capítulo 5

Framework de simulación: OMNeT++

5.1. Introducción

OMNeT++ es una herramienta de modelado y simulación por eventos discretos basado en el lenguaje de programación C++, desarrollada principalmente para crear simuladores de redes. Ofrece un IDE basado en Eclipse, un entorno de ejecución gráfico y da soporte a distintos tipos de redes. Además, también ofrece distintos frameworks que permiten añadir funcionalidad basada en dominios, como protocolos de Internet, modelado del rendimiento, y redes inalámbricas, que son presentados como proyectos independientes que podemos incluir como referencia en nuestro proyecto.

En los útimos años, ha ganado popularidad como plataforma de simulación de redes en la comunidad científica, así como en el terreno industrial y actualmente cuenta con una comunidad de usuarios en constante crecimiento.

Aunque Omnet++ no es simulador en sí, ofrece una infraestructura y las herramientas necesarias para generar simulaciones. Uno de los ingredientes fundamentales de esta infraestructura es una arquitectura basada en componentes para modelos de simulación. Estos modelos son ensamblados a partir de componentes reutilizables llamados módulos, que pueden ser combinados de innumerables maneras como piezas de LEGO. Los módulos se conectan unos con otros a través de puertas (llamados

puertos por otros sistemas) y son combinados para formar módulos compuestos. Entre ellos se comunican mediante el paso de mensajes.

OMNeT++ posee una arquitectura genérica, por lo que puede ser utilizado en varios dominios, como modelado de redes de comunicación cableadas e inalámbricas, modelado de protocolos, modelado de multiprocesadores y otros sistemas de hardware distribuidos, validación de arquitecturas hardware, evaluación de aspectos de rendimiento de sistemas software complejos, y, en general modelado y simulación de cualquier sistema en el que sea apropiado utilizar una aproximación por eventos discretos.

OMNeT++ es la versión libre para propósito académico y sin fines de lucro de OMNEST, la versión comercial. Para fines comerciales, es necesario conseguir las licencias de OMNEST de parte de Simulcraft Inc, la compañía encargada de desarrollar y comercializar este software.

5.2. El lenguaje NED

El lenguaje NED (*Network Description*) sirve para ayudar al usuario a definir la estructura del modelo de simulación. NED permite declarar módulos simples, conectarlos y ensamblarlos para formar módulos complejos. Es posible etiquetar módulos complejos como redes, que serán en sí mismas modelos de simulación.

Los archivos NED pueden convertirse a XML y viceversa sin producirse una pérdida de información. Esto resulta muy útil, por ejemplo, a la hora de extraer información o cuando queremos generar un archivo NED a partir de una información almacenada en una base de datos SQL.

El lenguaje NED cuenta con numerosas características que permiten una buena escalabilidad en proyectos de grandes dimensiones:

- **Jerárquico:** en OMNeT++, cada módulo puede simplificarse en módulos más pequeños y utilizarse como un módulo complejo.

- **Basado en componentes:** los módulos simples y los complejos son reutilizables, lo que sirve no solo para evitar copiar código, sino que además permiten la existencia de las llamadas librerías de componentes (como el framework INET, por ejemplo).

- **Herencia:** pueden existir clases que sean subclases de un módulo. Estos módulos derivados puede contener parámetros, puertas, más submódulos y conexiones adicionales.

- **Paquetes:** el lenguaje NED mantiene la estructura de paquetes de Java, para reducir el riesgo de nombres idénticos entre diferentes modelos. Además, para hacer más fácil la especificación de dependencias entre modelos de simulación, incluye `NEDPATH` (similar al `CLASSPATH` de Java).

- **Tipos internos:** los tipos de módulos y canales usados localmente por un módulo compuesto pueden ser definidos dentro de dicho módulo, con el fin de reducir el número de nombres.

- **Metadatos:** existe la posibilidad de anotar información acerca de tipos, parámetros, puertas y submódulos simplemente añadiendo propiedades. Estos metadatos no son utilizados directamente por el kernel, pero pueden servir de ayuda para otras herramientas, el entorno de ejecución o incluso para otros módulos dentro del modelo.

5.3. Eventos

OMNeT++ utiliza mensajes para representar eventos. Cada evento es representado por una instancia de la clase *cMessage* o una de sus subclases. Estos mensajes son enviados de un módulo a otro. Pueden implementarse temporizadores enviándose el módulo en cuestión un mensaje a sí mismo.

Los eventos van procesándose en orden de llegada. Más exactamente, si dos mensajes llegan a la vez, se aplican las siguientes reglas:

- El mensaje con un **tiempo menor** de llegada es ejecutado en primer lugar.

- El que tenga un **menor valor de prioridad** se ejecuta primero.

- El que haya sido **enviado antes** es el primero en ejecutarse.

5.4. IDE y *Workbench*

El IDE (Integrated Development Environment) de OMNeT++ está basado en la plataforma Eclipse, la que extiende con nuevos editores, vistas, *wizards*, y otras funcionalidades. OMNeT++ añade funcionalidad para crear y configurar modelos (archivos NED y INI), llevando a cabo ejecuciones en modo *batch* y posterior análisis de los resultados, gracias a que Eclipse ofrece diversas funcionalidades como edición C++, integración de SVN y otras opciones especiales, como modelado UML, depuración o acceso a bases de datos.

El entorno de OMNeT++ es fácilmente reconocible para aquellos que están familiarizados con la plataforma Eclipse.

La ventana principal de OMNeT++ consiste en varias Vistas y Editores. Eclipse es un sistema muy flexible, ya que permite al usuario mover, modificar el tamaño, ocultar o mostrar varios paneles, editores o ventanas. Esto permite al usuario customizar el entorno a su gusto y necesidades (Figura 5.1).

De un vistazo, los elementos más llamativos de la ventana son el *Project Explorer* a la izquierda, donde pueden verse todos los proyectos cargados en OMNeT++ (tanto si se encuentran abiertos como cerrados), la parte del *editor*, donde podemos abrir y modificar cualquier tipo de archivo (siendo las extensiones más frecuentes .ned, .ini, .xml, .cc, .h y .msg), la *barra de herramientas*, desde la que podemos realizar numerosas acciones, como crear un nuevo proyecto, ejecutar o depurar un proyecto, etc y finalmente, en la parte inferior tenemos un *panel* en el que podemos incluir distintas herramientas muy útiles, como la consola, el progreso de la ejecución, problemas encontrados, etc.

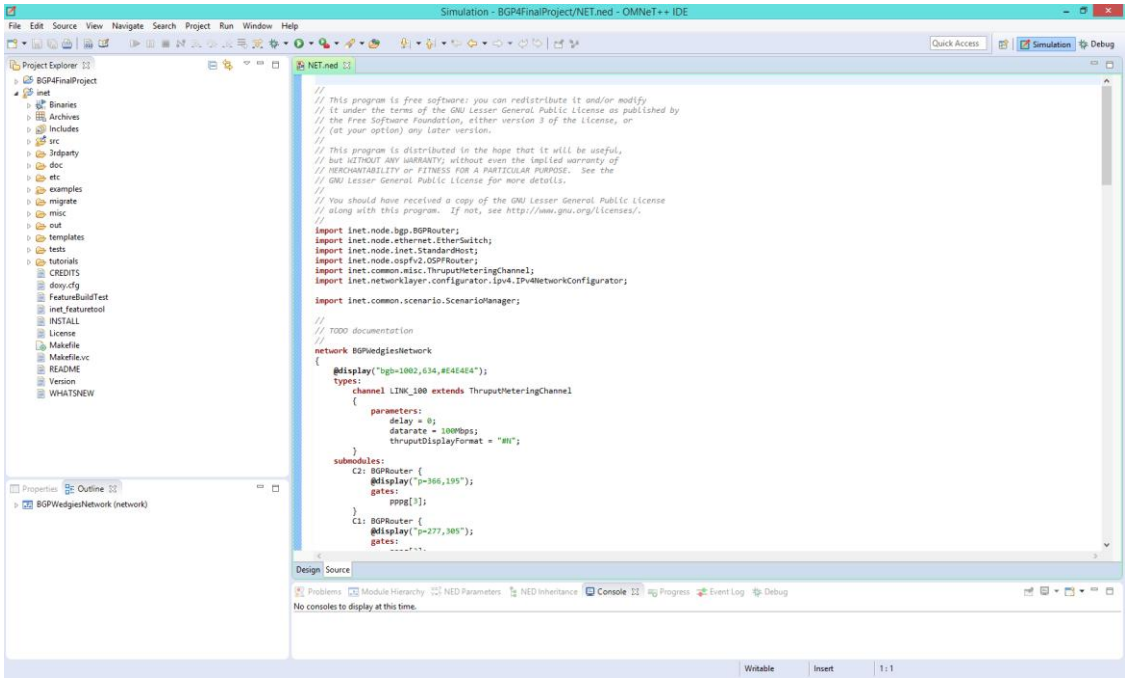


Figura 5.1. Entorno de trabajo de OMNeT++

5.5. Instalación

Ver Apéndice A.

5.6. INET

INET es un framework de modelado para redes cableadas, inalámbricas y móviles que contiene implementaciones de los protocolos IPv4, IPv6, TCP, UDP, SCTP, BGP, OSPF y varios modelos de aplicación. Los modelos que contiene a nivel de la capa de enlace son PPP (Point-To-Point), Ethernet y 802.11.

El framework INET se construye sobre OMNeT++, y utiliza el mismo concepto: módulos que se comunican a través del paso de mensajes. Hosts, encaminadores, switches y otros dispositivos de red son representados en OMNeT++ como módulos compuestos. Estos módulos son ensamblados a partir de módulos simples que representan protocolos, aplicaciones y otras unidades funcionales. Una red en OMNeT++ no es más que un módulo compuesto

formado por hosts, encaminadores y otros módulos. La interfaz externa de estos módulos es descrita en un archivo NED, que describe parámetros, puertas, submódulos y conexiones.

Los módulos en INET son organizados de forma jerárquica en paquetes de acuerdo al *Modelo OSI*. Los paquetes más altos contienen: *inet.applications* (correspondiente a la capa de Aplicación), *inet.transport* (correspondiente a la capa de Trasporte), *inter.networklayer* (correspondiente a la capa de Red) y *inet.linklayer* (correspondiente a la capa de Enlace). Otros paquetes contenidos en INET son, por ejemplo: *inet.base*, *inet.util*, *inet.world*, *inet.mobility* y *inet.nodes* (Figura 5.2).

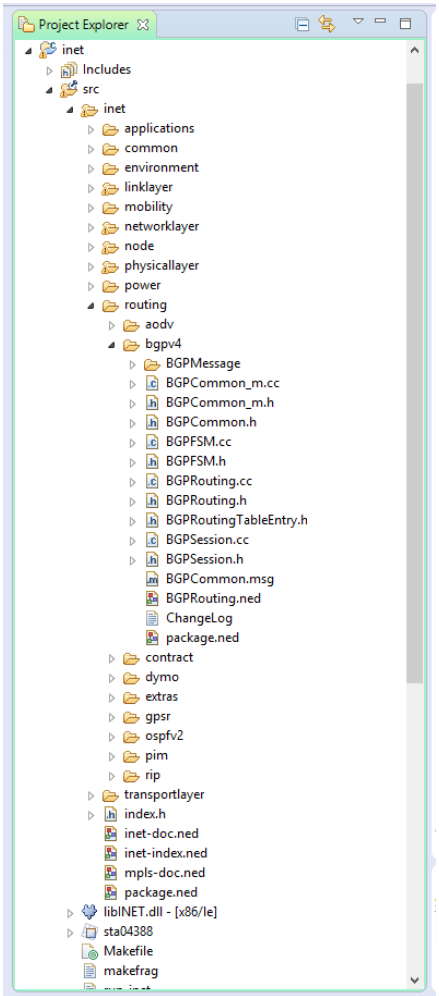


Figura 5.2. Directorio INET

El paquete *inet.nodes* contiene varios hosts, encaminadores, switches, puntos de acceso y otros módulos previamente ensamblados. Además, cuenta con varios ejemplos de cada modelo de simulación con el fin de probar escenarios reales antes de construir el tuyo propio e ir familiarizándose con la sintaxis y el funcionamiento de los modelos. El ejemplo más popular y con el que se recomienda empezar, es el proyecto *tictoc,* que contiene varios escenarios, empezando con uno muy simple que va aumentando en complejidad. Este ejemplo simula básicamente el paso de mensajes entre módulos y es una buena opción cuando no estamos familiarizados con el framework.

Existen diversos frameworks derivados de INET, utilizados sobre todo para simular redes móviles (INETMANET), redes inalámbricas (MiXiM), redes peer-to-peer (OverSim) y algunos más con distintos propósitos, como Veins o SimuLTE.

Para incluir INET en nuestro proyecto, hay que añadirlo como referencia. Para ello, hacemos click derecho en nuestro proyecto y nos dirigimos a *Properties*. En la ventana que aparece, nos dirigimos a *Project References* y ahí, marcamos el proyecto *inet* (Figura 5.3).

Para crear y ejecutar una simulación, es necesario escribir un archivo NED que contenga la red que vamos a simular, con sus correspondientes hosts, encaminadores y otros dispositivos de red conectados juntos. Para escribir y/o editar este tipo de archivo podemos hacerlo a través del editor gráfico de OMNeT++ o utilizar un editor de texto.

Debido a todo esto, INET se convierte en un componente indispensable para OMNeT++, sin el cual muchos de los escenarios posibles en redes no podrían llevarse a cabo sin su existencia. INET implementa una gran variedad de protocolos, innumerables funciones y ofrece la posibilidad de combinar módulos previamente definidos para crear infinitas combinaciones diferentes y con ello dar lugar a distintos modelos de simulación que pueden ser de ayuda para un trabajo futuro en lo que respecta al ámbito de las comunicaciones en Internet.

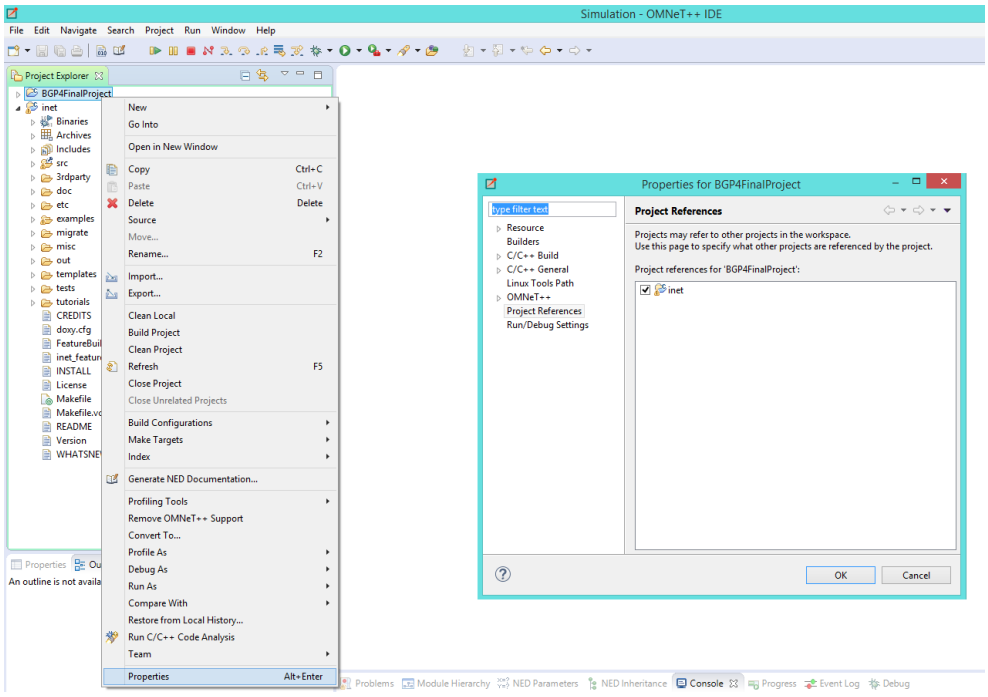


Figura 5.3. Inclusión de INET en nuestro proyecto

5.7. Proceso de decisión de rutas en INET

Antes de introducrir los escenarios creados para la simulación, es conveniente saber cómo se toman las decisiones de encaminamiento en INET. El proceso de decisión de rutas de INET, después de la implementación de un atributo primordial en el proceso de decisión, como es LOCAL_PREFERENCE, sigue estrictamente la definición del **RFC 4271** apartado 9.1:

- **Fase 1**: se calcula el grado de prioridad de cada una de las rutas recibidas de un vecino a través del método implementado para este proyecto *degreeOfPreference()*.

- **Fase 2**: esta fase se lanza una vez ha terminado totalmente la Fase 1. En ella, se elige la mejor de todas las rutas disponibles para cada destino en concreto y se instalan en Loc-RIB (Routing Information Base), una tabla de encaminamiento de nivel "superior"

que mantiene BGP independiente de las tablas de rutas de cada encaminador.

- **Fase 3**: esta fase se lanza una vez Loc-RIB ha sido modificado. En ella, se transmiten las rutas instaladas en Loc-RIB a cada vecino, de acuerdo con las políticas establecidas en PIB. Opcionalmente, en esta fase pueden llevarse a cabo otras tareas como agregado de rutas o reducción de información.

5.7.1. Fase 1: Cálculo del grado de prioridad

La función de decisión de esta fase es invocada en el momento en que un encaminador BGP recibe de un vecino un mensaje UPDATE que anuncia una nueva ruta, una ruta a reemplazar o una ruta eliminada. Esta función de decisión bloquea los Adj-RIB-In (Adjacent Routing Information Base, Incoming) de cada encaminador, una tabla de encaminamiento que BGP mantiene para cada encaminador con información entrante de cada uno de sus vecinos específicamente (análogamente, cada encaminador también posee una tabla de rutas denominada Adj-RIB-Out (Adjacent Routing Information Base, Outcoming) que contiene la información saliente a cada uno de sus vecinos). Las Adj-RIB-In son bloqueadas antes de operar con la información que contienen y son desbloqueadas después de operar sobre las rutas nuevas o inviables que contengan estas tablas.

Para cada una de las nuevas rutas, o rutas de reemplazo viables, el encaminador local determina el grado de prioridad de estas rutas como sigue:

- Si la ruta es aprendida de un vecino perteneciente al mismo AS, es decir, por encaminamiento interno, o se toma el valor del atributo LOCAL_PREF o se calcula el grado de preferencia de la ruta basándose en la política de decisión preconfigurada, pudiendo aparecer en este último bucles.

- Si la ruta es aprendida por encaminamiento externo, el encaminador local calcula el grado de prioridad basándose

en la política de decisión preconfigurada. Si el valor devuelto indica que la ruta no es elegible, la ruta es excluida de la siguiente fase de selección de ruta. En caso de obtener una ruta elegible, el valor devuelto ha de ser utilizado como valor del atributo LOCAL_PREF en cualquier anuncio transmitido por iBGP.

5.7.2. Fase 2: Selección de ruta

La función de decisión de la Fase 2 en invocada una vez ha finalizado la Fase 1. En esta fase, son consideradas todas las rutas que han sido elegibles en las tablas Adj-RIB-In de cada encaminador. Al igual que en la fase anterior, todas las tablas Adj-RIB-In son bloqueadas al comienzo, y desbloqueadas una vez se ha completado el proceso.

Si el atributo NEXT_HOP de una ruta BGP contiene una dirección IP que no puede resolverse, es decir, la dirección de NEXT_HOP no aparece en la tabla de rutas del encaminador que recibe el mensaje UPDATE, esta ruta ha de ser excluida del proceso de decisión de esta fase.

Si el atributo AS_PATH de una ruta BGP contiene un bucle, también hay que descartar dicha ruta. Para detectar estos bucles, se examina previamente el AS_PATH completo y se comprueba que el ASN del encamiandor local no aparece en la lista de AS de AS_PATH. Es fundamental que los encaminadores BGP no tomen decisiones de encaminamiento que puedan causar bucles.

Para cada conjunto de destinos para los que existen rutas elegibles en las tablas Adj-RIB-In, el encaminador local escoge la ruta basándose en lo siguiente:

- El grado de prioridad más alto de entre todas las rutas que llevan al mismo destino (a través de una sentencia de código dentro de *decisionProcess()* en *BGPRouting.cc*).
- Es la única ruta que lleva a ese destino.
- Es la resultante de un proceso de desempate en caso de dos o más rutas con el mismo grado de prioridad (resultado del método *tieBreakingProcess()* de *BGPRouting.cc* y llamado desde *decisionProcess()*).

El encaminador local instalará la ruta finalmente elegida en Loc-RIB, reemplazando cualquier ruta con el mismo destino que actualmente se encuentre registrada en dicha tabla. Una vez la ruta ha sido instalada la tabla de rutas, hay que asegurarse de que las anteriores rutas que llevaban al mismo destino han sido correctamente eliminadas, para evitar inconsistencias.
El encaminador locar debe determinar después cuál es el siguiente salto a través del valor que contiene en NEXT_HOP la nueva ruta.

5.7.3. Fase 3: Transmisión de rutas

La función de decisión de la Fase 3 es invocada inmediatamente después de terminar la Fase 2 o cuando ocurra uno de los siguientes eventos:

- Cuando las rutas mantenidas en Loc-RIB cambian
- Cuando rutas generadas localmente y aprendidas por algo que no sea BGP, cambian
- Cuando se establece una nueva conexión entre dos encaminadores BGP

En el momento de ejecución de esta fase, todas las rutas contenidas en Loc-RIB son procesadas e instaldas en las tablas Adj-RIB-Out, como información que será transmitida a los encaminadores vecinos. Una ruta puede no ser instalada si no contiene una dirección de destino o un valor de NEXT_HOP alcanzables.
Otras funciones, como el agregado de rutas, puede llevarse a cabo de manera opcional en esta fase.
Cuando todas las tablas Adj-RIB-Out han sido actualizadas, se procede al envío de mensajes UPDATE mediante el método *updateSendProcess()*.

5.9. Envío de mensajes UPDATE después del proceso de decisión

Durante el proceso de envío de mensajes UPDATE, se distribuyen las rutas elegidas en el paso anterior a otros encaminadores BGP, que pueden pertenecer o no al mismo sistema autónomo que el emisor.

Cuando un encaminador BGP recibe un mensaje UPDATE de un vecino por iBGP, la información de esa ruta no se transmite a otros encaminadores interiores del mismo AS, a no ser que dicho encaminador actúe como Reflector de Rutas.

En cuanto a eBGP, estos mensajes se envían normalmente entre encaminadores BGP frontera pertenecientes a distintos AS.

Capítulo 6

Funcionalidades añadidas a INET

6.1. Implementación del atributo *local preference*

En nuestro estudio del protocolo BGP, es necesario que cada ruta aprendida tenga una prioridad, ya que la implementación consta de un enlace primario con mayor prioridad y un enlace de respaldo (activo solo en caso de que el enlace primario deje de funcionar), que, precisamente para hacerlo menos preferible, hay que asignarle una prioridad más baja para que el resto de Sistemas Autónomos prefieran el otro enlace a la hora de encaminar su tráfico. Como este estudio está enfocado en la detección de BGP-Wedgies y el escenario elegido es propenso a sufrir una, ha sido necesario implementar el atributo LOCAL_PREF (bien conocido y discrecional), haciendo una modificación del código fuente de los archivos correspondientes al proceso de decisión de BGP del paquete INET. Siguiendo la sección 9.1 (*Decision Process*) del RFC4271, se han llevado a cabo los siguientes pasos:

- Creación de un fichero XML (*localPrefConfig.xml*) que contiene la información acerca de la preferencia para cada ruta. Este fichero se organiza en atributos, que son *RouterID* (para conocer qué encaminador es el que recibe la ruta), *Peer* (para saber a través de quién aprende la ruta o cuál es el siguiente salto), *Destination* (indica el destino final de la ruta) y *LocalPref* (valor de preferencia para esa ruta).

- Añadir el atributo LocalPref a *BGPRoutingTableEntry.h*

- En *BGPRouting.cc*:
  - Añadir el método *loadLocalPrefFromXML()* a la clase *BGPRouting.cc*, que se encarga de hacer una lectura del fichero XML y dejar la información almacenada en un vector de tipo *LPInfo*, que contiene los atributos presentes en el fichero XML. Este método se invoca desde el método *initialize()* de la misma clase.

  - Creación de un método *degreeOfPreference()* que, dada una ruta por parámetro devuelve su valor entero de LocalPref. Este método es invocado en el método *processMessage(),* esto es, en el momento en que un encaminador recibe un mensaje UPDATE.

  - En el método *decisionProcess()* se ha modificado ligeramente la parte del código donde se llamaba directamente al *tie-breaking,* puesto que anteriormente, al no tener una implementación para LocalPref, si una ruta se encontraba ya en la tabla de rutas, pasaba directamente a comprobar la longitud de su AS_PATH y el valor de su origen, siendo el valor de LocalPref el atributo prioritario para la decisión final de encaminamiento. De esta manera, solo en caso de que el valor de LocalPref para dos rutas con el mismo destino sea igual, se pasa a decidir cuál de ellas es elegida mediante *tie-breaking.*

6.2. Caída de la conexión física

Con el propósito de estudiar y simular las *BGP-Wedgies*, es necesario tener la posibilidad de realizar modificaciones en la topología en un instante de tiempo t para ver qué cambios se producen en el encaminamiento. En este caso la modificación necesaria consiste en tirar un enlace BGP, de manera que los encaminadores BGP de cada extremo del enlace dejen de anunciar sus rutas por el mismo.

El paquete INET proporciona una herramienta que permite hacer ciertos cambios en la topología en tiempo de ejecución utilizando parámetros definidos en un fichero XML, como

añadir/eliminar enlaces físicos, conectar/desconectar puertas o cambiar atributos de conexión, entre otros. Esta utilidad se llama *ScenarioManager* y ha sido utilizada como base para nuestro propósito enfocado al estudio de las *BGP-Wedgies*.

ScenarioManager actúa de cara a la topología de la red, trabajando directamente sobre los módulos que la componen. Es altamente útil ya que se puede acceder a todos los módulos de la red, a los submódulos de estos y a su vez a toda la implementación. De esta forma, para conseguir nuestro objetivo, es necesario desactivar conexiones BGP y volverlas a activar, por lo que además ha sido necesario intervenir en esa parte. Para ello, se siguen los pasos siguientes:

- Acceso a los módulos y tiempos especificados en el fichero XML *scenario.xml*, en este caso, *BGPRouter* y tiempo *t*.

- En el instante t, el módulo *ScenarioManager* recibe un automensaje (previamente programado tras la lectura del fichero xml), que indica que la puerta especificada ha de desconectarse. Entonces, se procede a la desconexión de dicha puerta y, por consiguiente, se desconecta también el enlace que une esa puerta y la puerta del otro lado, perteneciente al encaminador BGP vecino con el que hay establecida una conexión BGP.

- Tras la desconexión, cada uno de los encaminadores dejan de recibir mensajes KEEPALIVE del otro encaminador, lo que provoca que el HOLDTIMER expire y se proceda al cierre de la sesión BGP y por consiguiente, el cierre de la sesión TCP.

Después del proceso seguido por *ScenarioManager*, ha de enviarse un mensaje UPDATE para notificar que la sesión BGP ha fallado. Más tarde, de cada encaminador implicado en la sesión caída, se tienen que eliminar las rutas aprendidas por el otro lado de la conexión. Después, se envía un mensaje UPDATE para que el resto de encaminadores que aprendían rutas mediante esa sesión, eliminen dichas rutas de sus tablas de encaminamiento.

6.3. Implementación del mensaje *Notification*

Con el fin de ampliar las funcionalidades provistas por INET y OMNeT++, se ha implementado el tipo de mensaje NOTIFICATION, el cual es necesario enviar cuando se produce un error en la red (Sección 4.6). Para ello, igualmente ha sido necesario modificar los ficheros de INET correspondientes a los mensajes BGP. Los pasos seguidos han sido los siguientes:

- Creación de un mensaje NOTIFICATION, con extensión .msg (*BGPNotification.msg*), en el que se define su tamaño mínimo (21 Bytes), su tipo (3, por defecto) y sus campos *Error code* y *Suberror Code*.

- Creación de la correspondiente cabecera *BGPNotification.h,* que enlaza con INET y BGP y permite la creación a partir de ella de los ficheros *BGPNotification_m.h* y *BGPNotification_m.cc*, que contienen parámetros y funciones propias de la clase *cMessage* de OMNeT++ necesarias para su buen funcionamiento.

- Añadir funciones de envío de un mensaje NOTIFICATION a los archivos *BGPFSM.h* y *BGPFSM.cc*, correspodientes a la máquina de estados de BGP, para que, en caso de error, se envíe un mensaje de este tipo y sea manejado donde corresponda, además de cambiar el estado de la máquina del estado actual en ese momento al estado *idle*.

6.4. Eliminación de rutas aprendidas

Cuando se cierra una sesión BGP, sea cual sea el motivo, es necesario eliminar las entradas correspondientes a las rutas aprendidas asociadas a esa sesión tanto de los encaminadores extremos de dicha sesión, como de los encaminadores que las aprendieron a través de uno de ellos.

6.4.1. Encaminadores extremos de la sesión cerrada

Para eliminar las rutas aprendidas por un encaminador a través de su vecino BGP por la sesión caída, en el método *Established::HoldTimer_Expires()*(evento que se lanza en el momento en que expira el temporizador de HOLD cuando la sesión BGP se encuentra en estado ESTABLISHED y que es invocado una vez por cada encaminador extremo de la sesión*)* obtenemos la tabla BGP de cada uno de los encaminadores y llamamos al método *deleteBGPEntry(tablaBGP[i])*, siendo tablaBGP[i] la entrada correspondiente aprendida por su vecino (haciendo previamente la comprobación para asegurarnos de que la ruta se aprende del vecino *session.getPeerAddr() == tablaBGP[i] ->getGateway()*).

6.4.2. Resto de encaminadores

Para eliminar las rutas aprendidas a través de cualquiera de los dos encaminadores extremo de la sesión BGP caída, es necesario que estos envíen un mensaje UPDATE de borrado de rutas. Este caso ya viene contemplado en el propio mensaje UPDATE, diferenciándose del caso del anuncio de rutas en que éste no lleva ni información sobre los atributos ni sobre el NLRI (*Network Layer Reachability Information*). Por lo tanto, para configurarlo correctamente para que el encaminador que lo recibe sepa que debe eliminar la ruta contenida en el mensaje, es necesario poner a 0 la longitud de los campos correspondientes a los atributos BGP y al NLRI.

En el método de eliminado de rutas, hacemos que se envíe un mensaje UPDATE de este tipo (en este caso, solo por uno de los encaminadores, ya que el otro, al tener por el otro lado el enlace de backup, no anunció ninguna ruta por ahí) al siguiente encaminador al que está conectado. Ese hará lo mismo con su vecino, y así sucesivamente.

Para crear este tipo de mensaje, primero es necesario definir un nuevo tipo de ruta, que es DELETED_ROUTE, de forma que cuando se va a enviar el mensaje a través del método

updateSendProcess() de *BGPRouting.cc,* se detecte que es una ruta a eliminar y se añadan ciertos parámetros al mensaje, como son activar el array de *withdrawnRoutes*, estableciéndolo de tamaño 1, y poniendo a 0 el NLRI y el array de *pathAttributes*. También tenemos un atributo del tipo *BGPUpdateWithdrawnRoutes*, denominado *unfeasibleRoutes, en el que se establecen el prefijo IP de la dirección a elimnar (prefix)* y la longitud del mismo (*length*), que establece los bits correspondientes a la red a través del valor del *netmask.* Después de establecer estos ajustes, se envía el mensaje.

En el proceso de procesar un mensaje recibido de este tipo, en el método *processMessage()*, establecemos los parámetros extraídos del mensaje para crear una nueva entrada que será comparada con lo que aparece en la tabla para posteriormente eliminarla en el caso de haber una correspondencia.

Después, se invoca el método *decisionProcess()*, donde, si se encuentra una correspondencia entre la entrada pasada por parámetro y una entrada antigua en la tabla BGP del encaminador y, además, es un mensaje UPDATE de eliminado de rutas y proviene del encaminador utilizado para alcanzar el destino, dicha entrada será eliminada de la tabla de rutas.

Capítulo 7

Escenario BGP-4: Estudio de wedgies

7.1. Topología

El escenario desarrollado para su posterior simulación y análisis aparece en la Figura 7.1.

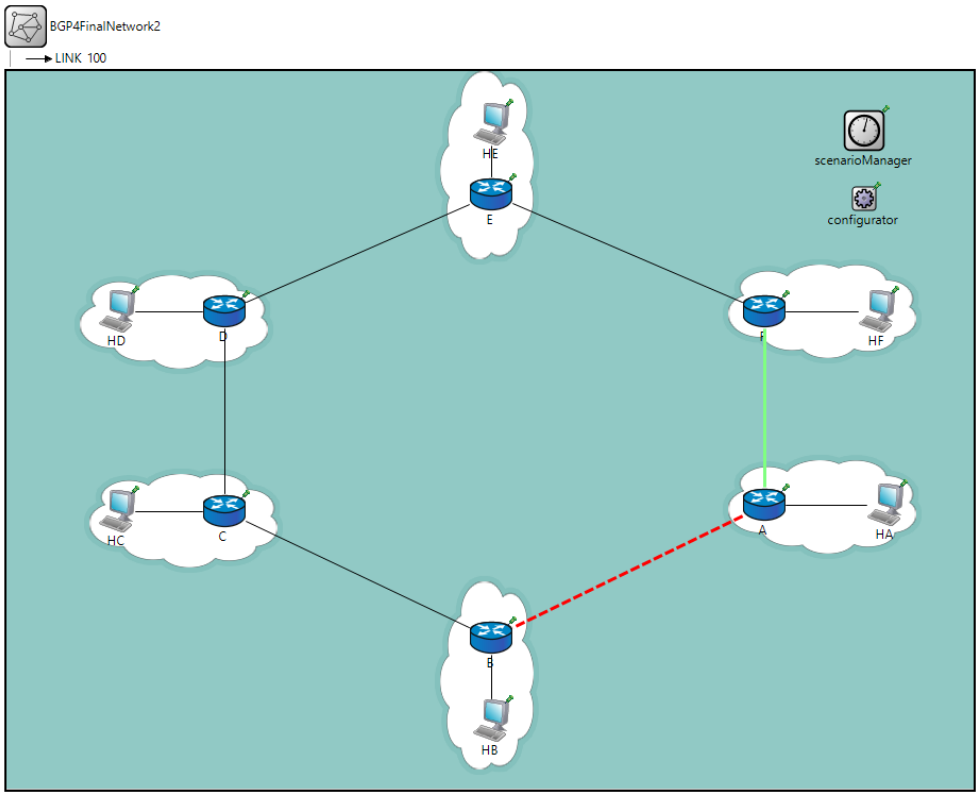


Figura 7.1. Escenario propenso a sufrir una wedgie 3/4

7.2. Información acerca de elementos de la topología

En la Figura 6.2. se detallan los elementos topológicos de los que consta el escenario que se está simulando.

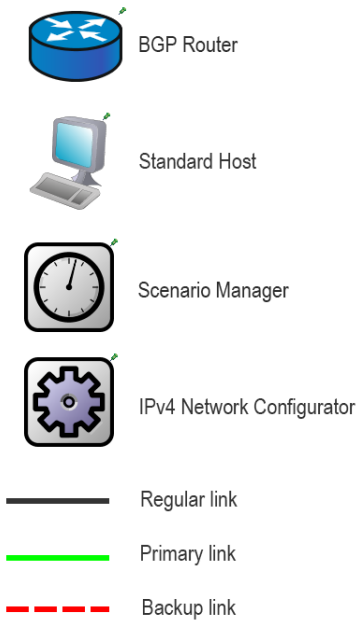


Figura 7.2. Descripción de los elementos de topología utilizados

7.3 Configuración: Modelo gráfico y estructura

La definición de la estructura de nuestro modelo de simulación se lleva a cabo en un fichero de extensión **.ned**. Como ya vimos en la sección 5.2, este tipo de archivos nos servirá de ayuda para configurar nuestra topología, puesto que consta de una parte de editor y una parte gráfica (Figura 7.3).

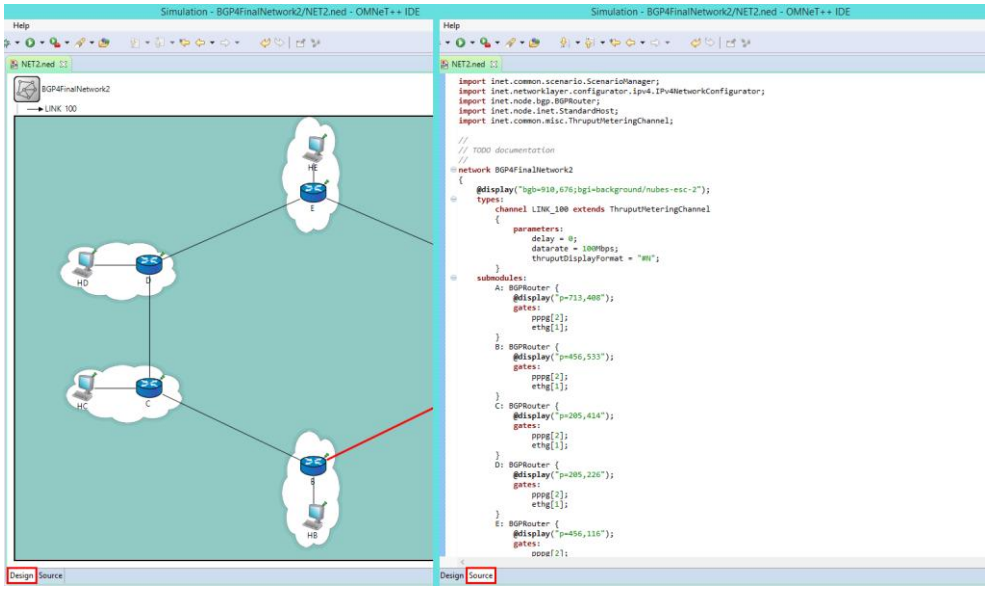


Figura 7.3. Parte gráfica y editor de un fichero .ned

Antes de empezar a configurar este archivo, es recomendable hacer un diseño previo de lo que va a ser la topología final. Para comenzar con la parte gráfica, disponemos de una *Paleta de Componentes,* a través de la cual podemos elegir qué elementos vamos a utilizar (Figura 7.4). Desde aquí podemos crear módulos simples, módulos complejos y redes.

En este caso, crearemos una red (la cual posteriormente será necesario enlazar al fichero de configuración del proyecto) y añadiremos encaminadores BGP (*BGPRouter*), encaminadores OSPF (*OSPFRouter*), *switches* (*EtherSwitch*) y *hosts* (*StandardHost*). Algunos de estos elementos vienen integrados en OMNeT++ por defecto, pero otros son específicos del paquete INET, por lo que para poder acceder a ellos es necesario haber incluido previamente INET en nuestro proyecto. Además, necesitaremos conectar todos estos elementos a través de enlaces (de diversos tipos), los cuales son accesibles a través del menú *Connection* de la paleta.

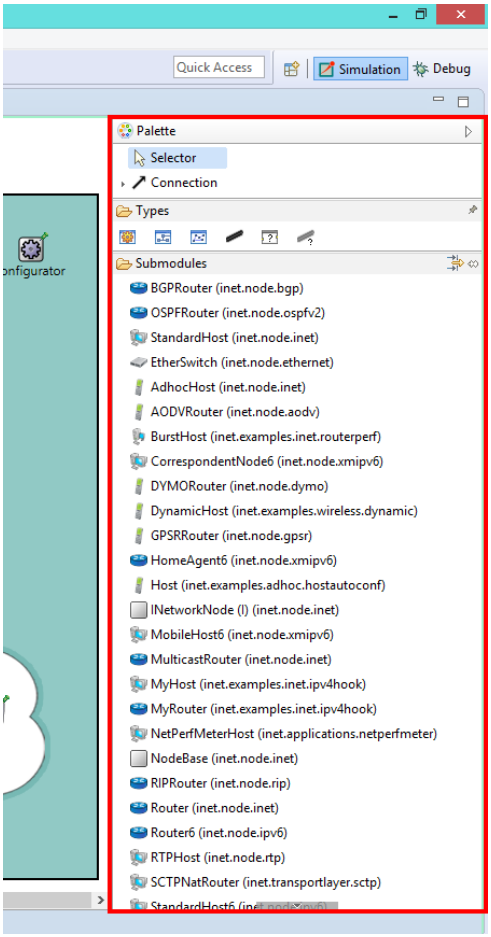


Figura 7.4. Paleta de elementos topológicos en INET

Además de incluir todos los nodos de los que constará nuestra red, es necesario añadir también un configurador IP (*IPv4Configurator*) para poder asignar correctamente direcciones IP a los nodos.

Cuando añadimos un elemento gráfico a la red, por defecto, en la parte de edición, solo nos aparecerán de forma automática los parámetros de posicionamiento. El resto de configuración requiere de introducir la información manualmente a través del editor, por ejemplo, los puertos de cada elemento (*gates*), que pueden ser para albergar enlaces punto a punto (*ppp*) o conexiones Ethernet (*eth*).

En cuanto a las interconexiones entre elementos, al hacerlo de forma gráfica pueden no aparecer todos los puertos aun habiéndolos definido previamente en el editor, por lo que se recomienda introducirlas manualmente, habiendo hecho un diseño de conexiones

previo, para evitar solapamientos y los consecuentes errores a la hora de ejecutar el proyecto.

7.4. Caso de estudio: BGP-Wedgies

El escenario que se ha implementado es una topología BGP propensa a sufrir una wedgie ¾. Esto quiere decir que la topología presenta un enlace primario, por el que se encaminará todo el tráfico en estado normal, y un enlace de respaldo, secundario o de *backup*, que se activará solo en caso de caída del enlace primario. Se ha establecido la red 192.168.1.0/24, pertenenciente al Sistema Autónomo 60111, como punto neurálgico del sistema, resultando así un estudio de alcance de esa red para el resto de Sistemas Autónomos. Así, puede comprobarse cómo el resto de Sistemas Autónomos ignoran las rutas que tengan que atravesar el enlace de backup y prefieren las rutas que atraviesan el enlace principal, aunque estas tengan mayor número de saltos. En un instante de tiempo t definido, el enlace principal sufrirá una caída, activándose así, el enlace de respaldo. Ante esta nueva situación, todo el tráfico se encaminará a través de este enlace, lo que forma parte de la política de encaminamiento. Nuevamente, cuando el enlace principal vuelve a estar operativo, el tráfico no vuelve a encaminarse por este enlace, sino que sigue utilizando el secundario. Este estado final no es el esperado, por lo que se ha producido una *wedgie*. Para volver al estado inicial y esperado es necesario intervenir manualmente, tirando de nuevo la conexión backup para que el tráfico vuelva a encaminarse por el enlace principal y volver a reactivarlo después. De esta forma se vuelve al estado inicial.

7.5. Análisis del escenario

Este escenario está compuesto únicamente por 6 Sistemas Autónomos, cada uno de los cuales consta de un encaminador BGP (tipo ASBR). Se ha decidido utilizar una topología sencilla, puesto que el objetivo de este proyecto es el encaminamiento externo y no interno, dejando de lado el protocolo OSPF. De esta manera, solo se

establecen sesiones BGP externas (eBGP) para alcanzar las redes del sistema.

El escenario está compuesto, además, por diversos hosts, conectados cada uno respectivamente al encaminador BGP de su AS a través de la interfaz *eth0*.

La política de encaminamiento fijada está basada en el valor del atributo Local_Preference, de manera que todo el tráfico se encamina por el enlace primario, que une los encaminadores A y F, y el enlace de backup queda como respaldo. Gracias a la implementación de Local_Preference puede conseguirse el estado esperado, ya que de otra manera OMNeT++ se basa primeramente en el número de saltos o número de encaminadores contenidos en el atributo AS_PATH y, en caso de empate, elige la ruta con un valor del atributo Origin menor.

En la Figura 7.5. puede verse el resultado esperado de encaminamiento para este escenario.

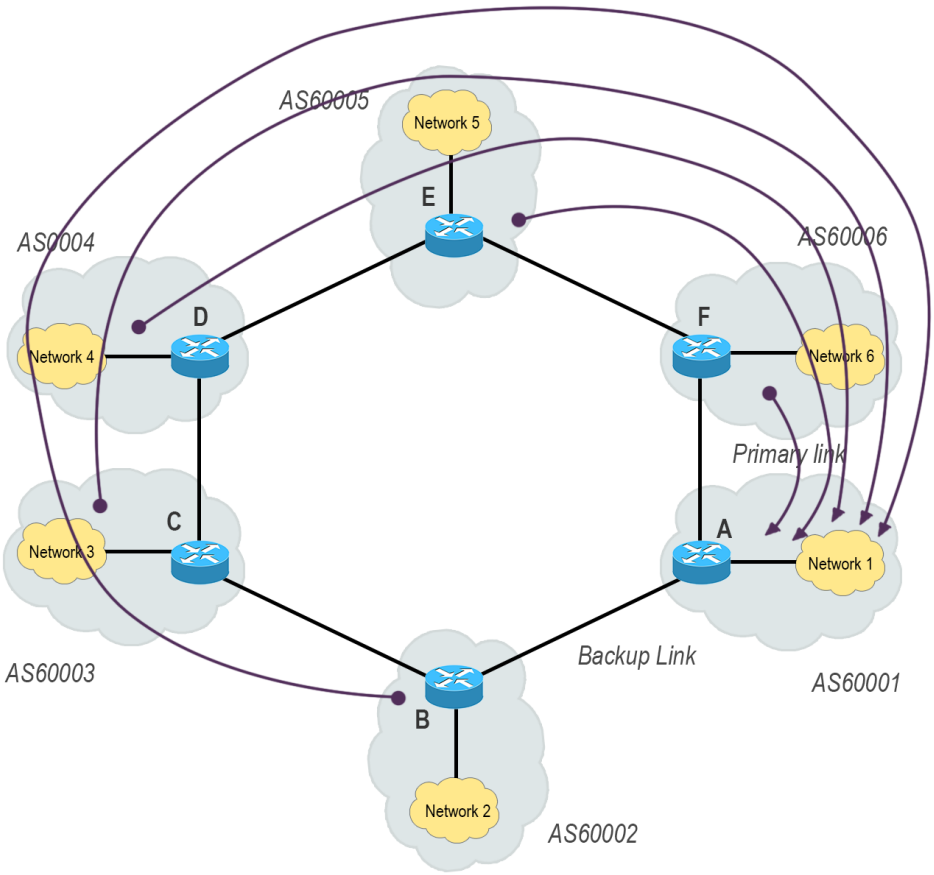


Figura 7.5. Comportamiento esperado del escenario

7.6. Encaminamiento por el enlace de backup

Como parte del estado esperado y tras haber desactivado el enlace primrario y habiendo sido eliminadas las rutas previamente aprendidas que seguían dicho enlace, los encaminadores deben aprender otro camino (anteriormente descartado debido a tener un valor de LOCAL_PREF menor) para poder encaminar su tráfico hacia el destino. Para ello, el encaminador extremo de sesión que por su otro extremo tiene una conexión de backup con otro encaminador vecino, tiene que anunciar nuevamente una ruta hacia la red de su propio AS por ese enlace, hacia el destino 192.168.1.0/24. Para ello, ha sido necesario, recuperar la entrada que lleva a ese destino de la tabla de encaminamiento IP de este encaminador, al que se encuentra directamente conectado (tipo IFACENETMASK). Después se ha conseguido el *SessionIndex* de la sesión de backup y se ha llamado con esa entrada y ese id de sesión como parámetros a un método denominado *updateSendProcessRestoreRoute(),* que llama a su vez a *updateSendProcess(),* encargado de anunciar nuevamente la ruta por el enlace de backup. Al recibir un mensaje de este tipo, se sigue el proceso de decisión normal para agregar la nueva ruta a la tabla de encaminamiento BGP.

7.7. Restablecimiento del enlace principal y posterior encaminamiento

Después de esto, en otro instante posterior de tiempo t, el enlace principal vuelve a estar disponible, pero el tráfico sigue encaminando por el enlace de backup. En este caso, nos encontramos ante una *wedgie*, ya que, ante la misma situación inicial, conseguimos otro estado, no esperado pero igualmente estable.
Para solucionar esta situación, es necesario tirar el enlace de backup, de la misma manera, donde las rutas anunciadas por el enlace de backup seguirían el mismo proceso de borrado y restablecimiento explicado anteriormente para el caso de la caída del enlace principal, y posteriormente volver a conectar el enlace de backup, de forma que volvemos a encontrarnos en la situación inicial con el estado esperado.

7.8. Configuración: BGP

La configuración BGP se hace en el archivo *BGPConfig.xml*. En él, se definen los AS que tendrá el sistema y a cada uno de ellos se le asignan las direcciones IP de sus encaminadores frontera que van hacia el interior. Asimismo, se definen las sesiones externas BGP entre encaminadores frontera de distintos AS. Ver Sección C.1.3.

7.9. Configuración: Direccionamiento IP

La configuración de direcciones IP se hace en el archivo *IPv4Config.xml*. Ver Sección C.1.4.

7.10. Configuración: OSPF

La configuración OSPF se hace en el archivo *OSPFConfig.xml*. En él se definen las áreas en las que se va a dividir la topología y por qué enlaces se habla OSPF. Ver Sección C.1.5.

7.11. Configuración: Local_Preference

La configuración para el atributo Local_Preference se establece en el fichero *localPrefConfig.xml*. Ver Sección C.1.6.

7.12. Configuración: fichero de inicialización (.ini)

El archivo de configuración omnet.ini inicializa los valores de los parámetros de los módulos presentes en el proyecto y enlaza los

ficheros necesarios de configuración, además de definir otros elementos como el tiempo de simulación, por ejemplo. Ver Sección C.1.7.

7.13. Configuración: ScenarioManager (scenario.xml)

La configuración para el módulo ScenarioManager se establece en el fichero *scenario.xml*. Ver Sección C.1.8.

7.14. Ejecución

Una vez finalizada la ejecución, procedemos al análisis de dos elementos clave: la tabla de encaminamiento IP, donde aparecen todas las rutas aprendidas, y la tabla de encaminamiento BGP, donde solo aparecen las rutas aprendidas mediante BGP, bien sea externo o interno.

Tabla de encaminamiento IP

Como podemos observar, la tabla de encaminamiento IP está formada por diversas entradas, cada una de las cuales consta de los siguientes parámetros:

- **dest (*destination*):** contiene la dirección IP de destino (puede ser un host o una red).
- **gw (*gateway*):** contiene la dirección IP del encaminador siguiente en la ruta para llegar al destino.
- **mask (*netmask*):** másacara de subred de la red destino.
- **metric:** métrica utilizada
- **if (*interface*):** la interfaz por la que ha de salir el tráfico para alcanzar el destino.
- **type:** puede ser directo (*direct*) o remoto (*remote*). Para rutas directas, el siguiente salto es la dirección IP de destino, mientras que para rutas remotas, es la dirección de gateway.

- **source:** en este caso puede ser *OSPF (*si la ruta ha sido aprendida por OSPF*)*, *BGP(*si la ruta ha sido aprendida por BGP*)* o *IFACENETMASK (*se añade la dirección de red de cada una de sus interfaces*).*

Rutas remotas y por defecto han sido previamente eliminadas en la configuración.

Tabla de encaminamiento BGP

Además de la tabla de encaminamiento IP, cada encaminador posee una tabla de encaminamiento BGP, donde se detallan los siguientes parámetros:

- **Destination:** contiene la dirección IP de la red destino y la máscara de subred.

- **PathType:** Indica si la ruta ha sido aprendida por encaminamiento exterior (*EGP*) o encaminamiento interior (*IGP*).

- **LocalPreference:** preferencia para esa ruta en concreto.

- **NextHops:** indica el siguiente salto en la ruta para alcanzar el destino. Tanto si la ruta es aprendida por eBGP o iBGP, el siguiente salto será el encaminador frontera del AS que envió la ruta por encaminamiento exterior al actual AS.

- **AS**: contiene la lista de sistemas autónomos que ha de atravesar para llegar al destino y en qué orden.

7.14.1. Tablas de encaminamiento

Como ejemplo dentro de este escenario, se muestran las tablas de encaminamiento IP y BGP del encaminador F antes de la caída del enlace principal(para IP, Figura 7.6, para BGP, Figura 7.7) y

después de la caída (para IP, Figura 6.8, para BGP (Figura 6.9)), perteneciente al AS60006. El encaminador F se encuentra directamente conectado al encaminador A, al que se encuentra unido mediante el enlace primario, y a E, al que se encuentra conectado mediante un enlace normal.

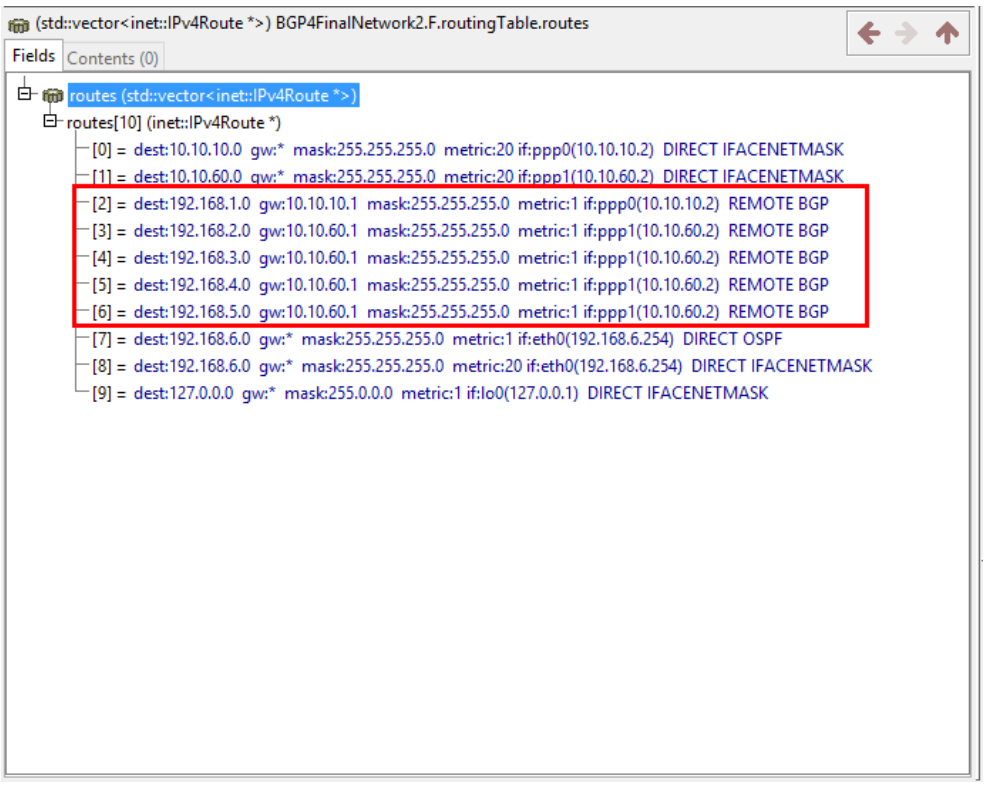


Figura 7.6. Tabla de encaminamiento IP para el encaminador F antes de la caída del enlace principal

Como puede apreciarse en la Figura 6.6, el encaminador F (y el resto de una forma similar), antes de la caída del enlace principal, aprende todas las rutas por BGP remoto. El resto de rutas que aparecen en la tabla se aprenden o por interfaz directa (IFACENETMASK). Nos fijaremos especialmente en la ruta con destino 192.168.1.0/24. En la tabla vemos que, el siguiente salto en la ruta es el encaminador con ID 10.10.10.1/24 (perteneciente al AS60001, correspondiente al encaminador A) saliendo por la interfaz *ppp0* (10.10.10.2/24).

La Figura 7.7. muestra la tabla de encaminamiento BGP para el encaminador F antes de la caída del enlace principal. Se puede observar que aparecen todas las rutas que la tabla de encaminamiento IP indicaba que se aprendían mediante BGP. Si nos fijamos expresamente en la ruta con destino 192.168.1.0/24 vemos que se aprende por EGP (BGP externo), que tiene una preferencia local de 300 (lo que hace que se utilice el enlace principal y no otro), siendo su siguiente salto 10.10.10.1/24 y llegando al AS60001 de un salto, puesto que está conectado directamente al encaminador que alberga dicha red.

El resto de información contenida en la tabla aporta igualmente información útil sobre el resto de rutas aprendidas, detallando de la misma forma el número de saltos, la preferencia local de la ruta, etc, y pudiendo comprobar que todas siguen el enlace principal.

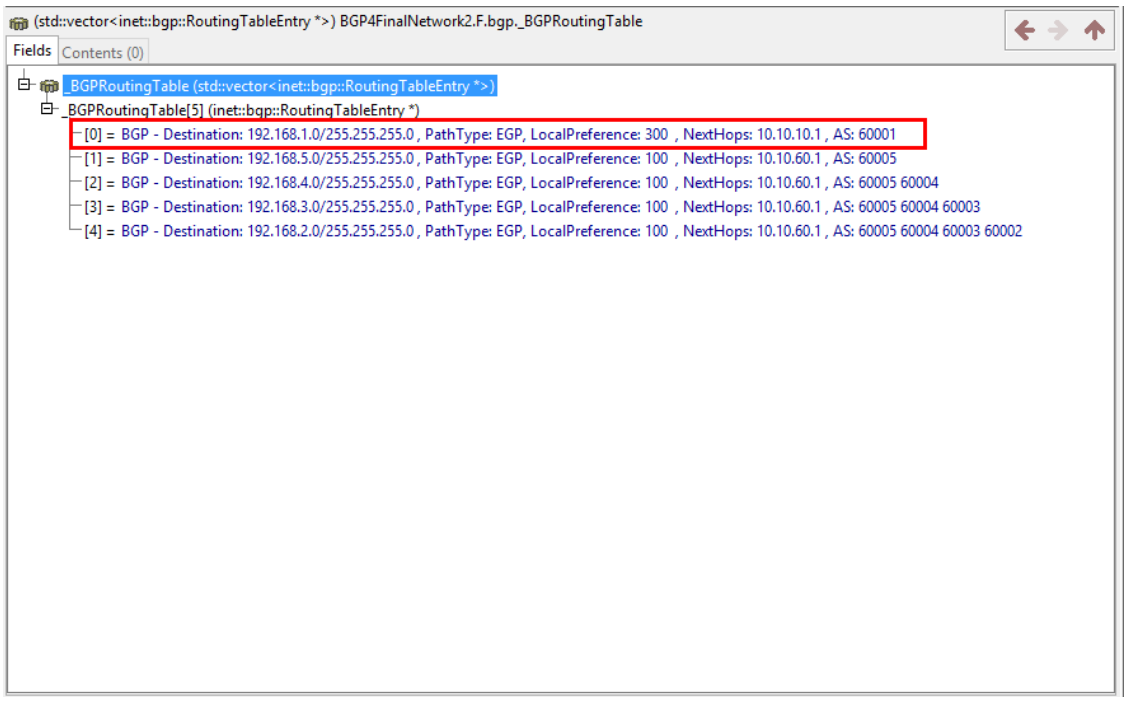


Figura 7.7. Tabla de encaminamiento BGP para el encaminador F antes de la caída del enlace principal

Siguiendo con el mismo ejemplo, pasamos ahora a analizar las tablas IP y BGP del encaminador F después de la caída del enlace principal (Figura 7.8 y Figura 7.9, respectivamente). En esta nueva situación, el estado esperado es que las rutas sean ahora aprendidas mediante el enlace de respaldo o backup.

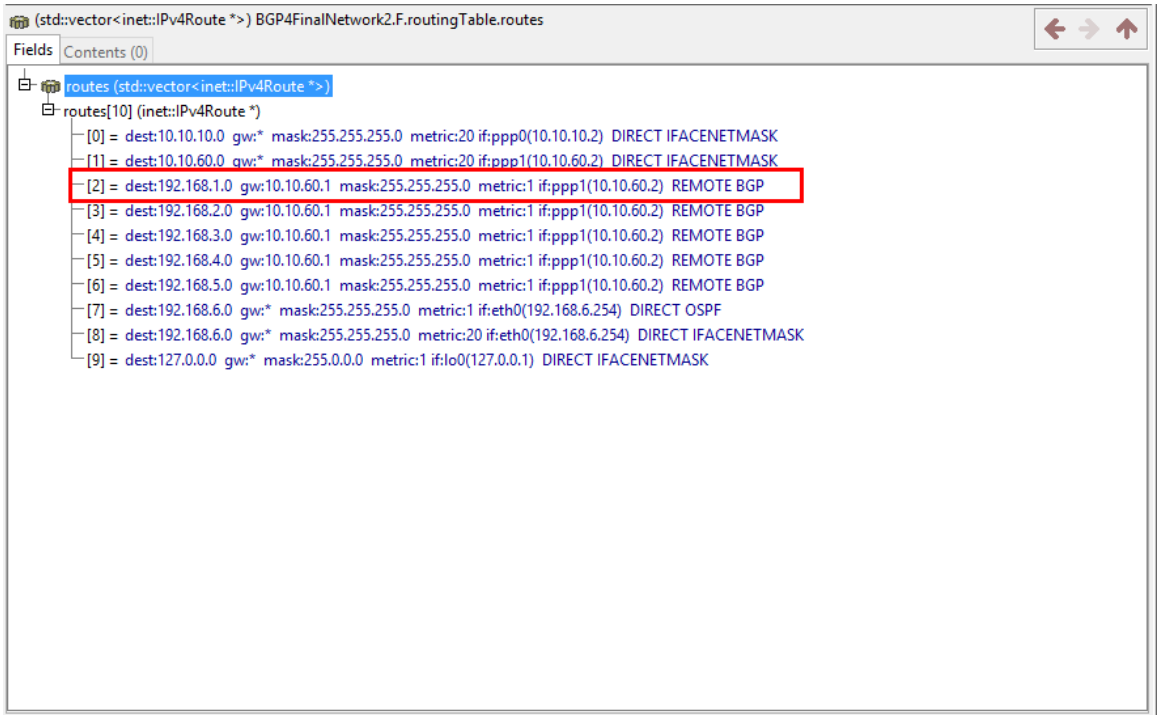


Figura 7.8. Tabla de encaminamiento BGP para el encaminador F después de la caída del enlace principal

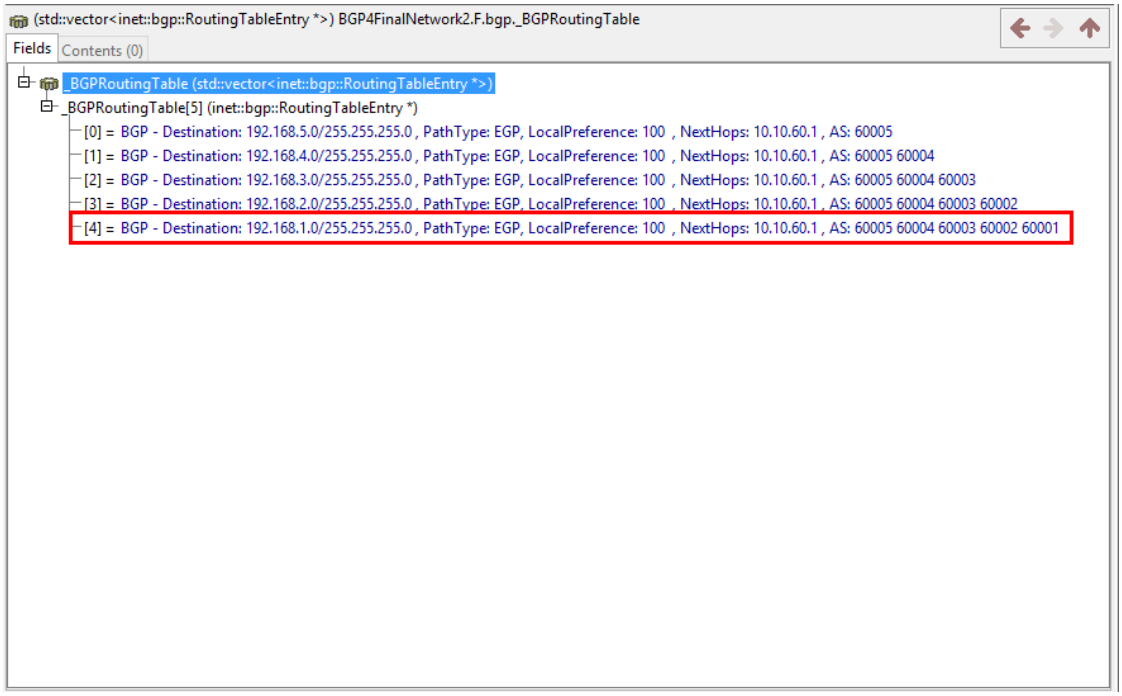


Figura 7.9. Tabla de encaminamiento BGP para el encaminador F después de la caída del enlace principal

En la tabla de encaminamiento IP, vemos que, después de la caída del enlace principal, se han producido modificaciones. Por ejemplo, ahora el *gateway* es 10.10.60.1, ID del encaminador E, en el AS60005 y la salida del AS actual ahora es por la interfaz ppp1 (10.10.60.2/24) en lugar de por ppp0 (10.10.10.2/24)

En la tabla de encaminamiento BGP, vemos que para alcanzar ahora la red 192.168.1.0/24, la preferencia local de la ruta es de 100, el siguiente salto ha cambiado y ahora es 10.10.60.1 (conducente al encaminador E del AS60005) y el AS_PATH está formado por AS60005, AS60004, AS60003, AS60002 y AS60001, el camino más largo, que además atraviesa el backup ya que el enlace primario ha desaparecido.

7.15. Eventos

Durante la ejecución, se van sucediendo una serie de eventos en forma de mensajes que utilizan los elementos de la topología para comunicarse entre sí. Estos eventos aparecen detallados en la parte inferior de la ventana de simulación y aportan información sobre las acciones que se realizan a través de cada protocolo utilizado en tiempo de ejecución. Más exactamente, podemos comprobar en qué instante de tiempo de la ejecución se producen y el número de evento. Por ejemplo, en la Figura 7.10, podemos ver como los encaminadores A y F, pertenecientes a distintos AS y conectados a través de un enlace físico, establecen asimismo una sesión BGP y más tarde intercambian rutas de encaminamiento. Para ello, intercambian mensajes Open, KeepAlive y Update después de establecer una sesión TCP. Sucede exactamente lo mismo instantes después con los encaminadores A y B, estableciendo A una segunda sesión con su otro vecino, B.

Event#	Time	Src/Dest	Name
#427	16.000001	A --> F	SYN
#436	16.00000708	F --> A	RST+ACK
#450	16.500001	F --> A	SYN
#459	16.50000708	A --> F	SYN+ACK
#472	16.50001316	F --> A	ACK
#478	16.50001692	F --> A	BGPOpen(l=29,1msg)
#487	16.50001892	A --> F	BGPOpen(l=29,1msg)
#501	16.500025	A --> F	ACK
#509	16.500027	F --> A	ACK
#521	16.50003076	F --> A	BGPKeepAli(l=19,1msg)
#527	16.50003276	A --> F	BGPKeepAli(l=19,1msg)
#541	16.50003804	A --> F	ACK
#549	16.50004004	F --> A	ACK
#561	16.5000438	F --> A	BGPUpdate(l=28,1msg)
#567	16.5000458	A --> F	BGPUpdate(l=28,1msg)
#580	16.5000518	A --> F	ACK
#587	16.5000538	F --> A	ACK
#616	17.000001	A --> B	SYN
#625	17.00000708	B --> A	RST+ACK
#639	17.500001	B --> A	SYN
#648	17.50000708	A --> B	SYN+ACK
#661	17.50001316	B --> A	ACK
#667	17.50001692	B --> A	BGPOpen(l=29,1msg)
#676	17.50001892	A --> B	BGPOpen(l=29,1msg)
#690	17.500025	A --> B	ACK
#698	17.500027	B --> A	ACK
#710	17.50003076	B --> A	BGPKeepAli(l=19,1msg)
#716	17.50003276	A --> B	BGPKeepAli(l=19,1msg)
#731	17.50003804	A --> B	ACK
#739	17.50004004	B --> A	ACK
#751	17.5000438	B --> A	BGPUpdate(l=28,1msg)
#757	17.5000458	A --> B	BGPUpdate(l=56,2msg)
#771	17.5000528	A --> F	BGPUpdate(l=28,1msg)

Figura 7.10. Eventos generados a partir de la simulación

Capítulo 8

Conclusiones y trabajo futuro

Como conclusión de este estudio, se puede decir que la forma en que se encamina el tráfico en Internet es decisiva a la hora de establecer comunicaciones firmes y que satisfagan las expectativas del usuario final. Comprender cómo se utilizan los protocolos de encaminamiento a nivel práctico y cómo cualquier cambio provocado en la red puede influir directamente en la decisión final de encaminamiento, resulta esencial para crear estas comunicaciones.

El estudio e investigación de los diferentes protocolos de encaminamiento, tanto interior como exterior, es necesario para seguir desarrollando buenas prácticas en cuanto a comunicaciones globales se refiere, y mejorar la infraestructura actual, ya sea creando nuevos protocolos de encaminamiento, nuevas arquitecturas de red o simplemente mejorar ciertos métodos ya implantados.

El protocolo BGP ofrece gran flexibilidad de cara al encaminamiento interdominio a la vez que se refuerzan las políticas de encaminamiento y se evita, en medida de lo posible, la aparición de bucles. Aún así, queda mucho camino que recorrer en lo relacionado con el encaminamiento a través de BGP, pues, a pesar de que este protocolo provee un servicio que funciona relativamente bien, presenta bastantes problemas, como son la convergencia o problemas de seguridad en escenarios no configurados correctamente. Otro problema que aparece constantemente es el crecimiento cuadrático de las tablas de encaminamiento al producirse anuncios de rutas masivos por no haber posibilidad de utilizar un agregado de rutas. Si a estos problemas le añadimos además el hecho de que los Sistemas Autónomos que forman Internet están bajo la gestión de

grandes empresas comerciales que principalmente buscan el beneficio propio, esto puede desencadenar que en años venideros, las comunicaciones se compliquen aún más.

Lo ideal, sin duda, sería proponer un plan de acción ante todo lo que no funciona como debería en lo que respecta al encaminamiento en Internet y trabajar al mismo tiempo que Internet va creciendo en lo que a desarrollo, mantenimiento y estudio de las redes se refiere, sin olvidarse, por supuesto, de la cada día mayor demanda de recursos y funcionalidades del usuario final.

Puesto que la mayoría de herramientas que se dedican a simular BGP no ofrecen grandes funcionalidades ni una implementación exacta del protocolo, resulta muy útil el hecho de mejorar una herramienta como es OMNeT++ y su framework INET para poder llevar a cabo un estudio exhaustivo de BGP y sus características. El trabajo realizado en este proyecto, se ha basado principalmente en añadir funcionalidades esenciales al framework INET para conseguir una implementación más completa del protocolo BGP, puesto que lo inicialmente provisto sólo nos permitía realizar una simulación muy básica de este protocolo.

En primer lugar, se ha implementado el atributo LOCAL_PREFERENCE, ya que es el primer atributo en ser comprobado en el proceso de decisión de rutas y, careciendo de su implementación, el primer atributo tenido en cuenta era AS_PATH. Esto no es una idea muy acertada, puesto que ni sigue el estándar del RFC 4271 ni presenta una situación de un problema real, ya que fácilmente pueden presentarse rutas con diferente grado de preferencia con el mismo número de saltos hasta llegar al destino.

También se ha llevado a cabo la implementación del mensaje NOTIFICATION, esencial para informar de la aparición de un error en una sesión BGP. Sin su implementación, en primer lugar, no se ajusta al estándar, y, en segundo lugar, lo correcto es que la sesión BGP se cierre al recibir este mensaje y no repetir código en cada situación de error.

Otra aportación ha sido el eliminado de rutas, ya que, como se ha comentado anteriormente, las opciones que ofrecen OMNeT++ e INET se limitan a una simulación estándar de BGP, donde no se producen errores en la red. Para ello, también se ha introducido en el proceso de decisión la posibilidad de que el mensaje UPDATE contenga rutas para ser eliminadas en lugar de para ser añadidas.

El hecho de que puedan anunciarse rutas previamente anunciadas y descartadas también es otra de las mejoras implementadas, ya que, después de producirse un error en la sesión y de eliminar rutas que fueron elegidas frente a otras con el mismo destino, esas otras han de ser anunciadas de nuevo e incorporadas a las tablas de rutas.

Para llevar a cabo este proyecto, además, ha sido necesario realizar un estudio previo muy a fondo del protocolo BGP, a través del RFC 4271, toda la información y documentación aportada por CISCO, *papers*, otros proyectos de investigación, etc. Así como aprender a utilizar el software OMNeT++ y el paquete INET, a través de sus respectivos manuales, guías de usuario, guías de IDE y pruebas de todo tipo utilizando todos los ejemplos incorporados en INET.

Como trabajo futuro, cabe destacar el hecho de que, al ser BGP un protocolo que presenta ciertas restricciones a día de hoy y al no poseer herramientas que implementan el protocolo correctamente en su totalidad, resulta muy interesante el poder completar la implementación de un software para conseguir simular el funcionamiento real del protocolo BGP. Algo en lo que se debería hacer más hincapié es el tema de las *wedgies* y el proceso de decisión, ya que de ello depende el buen funcionamiento de la red y el comportamiento correcto y esperado. Así como adaptar estas simulaciones a IPv6, ya que es algo que llegará en algún momento y conviene ir adelantándose a los acontecimientos y proveer soporte a esta nueva implementación.

Capítulo 9

Conclusions

As a conclusion of this study, we can say that the manner the traffic is routing over the Internet is decisive when it comes to establish solid communictions that satisfy final users expectations. Understanding how routing protocols are used in practice and how changes in a network can directly impact on the routing final decision, are essential topics in order to create the previously mentioned strong communications.

The study and investigation of the different existing routing protocols, both internal and external, are crutial to keep developing good practices for global communications and improve the current infrastructure, by creating new routing protocols, new network architectures or just enhancing existing methods.

The Border Gateway Protocol provides a high degree of flexibility when interdomain routing is concerned, while it is working on enforcing routing policies and the loops are prevented. However, the routing through BGP is a long way to walk, because, even though this protocol provides a proper functioning and a pretty good performance, it presents several problems, like slow convergence or security problems in presence of bad-configured scenarios.

Another problem is the routing tables cuadratic growth when a massive number of routes is announced, in case a route aggregation could not be performed. Adding to this the fact that the Autonomous Systems that create the Internet are under the management of big commercial companies whose principal interest is their own profit, can result in communications difficulties in the coming years.

Proposing an action plan against everything that is not working properly would be ideal, as well as working hand in hand with the Internet growth in terms of development, maintenance and study about networks, without forgetting the everyday growing demand of resources and features from the final user.

Due to basically most tools dedicated to simulate BGP do not provide loads of functionalities or an exact implementation of this protocol, the fact of improving a tool like OMNeT++ and its framework INET is very useful, in order to develop a deep BGP study and its features. The work done has been mainly based on adding new essential features to INET, in order to get a more completed implementation, since the features provided initially only let us make a very basic simulation of this protocol.

Firstly, an implementation of the attribute LOCAL_PREFERENCE has been made, because this is the first checked attribute in the decision process. Otherwise, the first checked attribute was AS_PATH. This is not a good idea, since it does not follow the standard and it does not show a real problem, because it is easy to find some different routes with a different value of priority but the same number of hops to reach the destination.

Also, the NOTIFICATION message has been implemented, essential to inform that there has been a problem with the BGP session. Firstly, if we do not have an implementation of this message, we are not following the standard, and, secondly, the session should be closed after receiving this message, being inefficient to duplicate code every time we have an error condition.

Another contribution has been route deletion, since OMNeT++ and INET provide limited functions to simulate BGP. In order to achieve it, it has been necessary to introduce the possibility of deleting routes using the UPDATE message instead of adding them in the decision process.

Some other improvements are related to announcing routes that were previously deleted because they were not preferred again. After a session error, the routes learned through that session were deleted, so it is necessary to restore the previously discarded routes in order to have another route to reach the same destination.

Furthermore, it has been crutial to develop a first deep study of the BGP protocol, reading the RFC 4271, every piece of information and documentation provided by CISCO, some papers,

another investigation projects, etc. As well as learning how to use the frameworks OMNeT++ and INET, reading their respective manuals, user guides, IDE guides and try every INET example.

To conclude, and related to future work, we can highlight the fact that BGP presents some restriction nowadays and the available tools do not implement the protocol in the right way, enhancing a software tool to get the most of BGP becomes really interesting. Something to keep and eye on are the BGP-Wedgies and the decision process, given that the good functioning of the network and the intended behavior depend on it. As well as adapting these simulations to IPv6, since it is something that is approaching and it would be wise to anticipate events and provide support to this new implementation.

Apéndice A

Instalación y configuración de OMNeT++

A.1. Información general

Este apéndice describe cómo instalar OMNeT++ en las distintas plataformas soportadas. OMNeT++ ha sido testeado y probado para soportar los siguientes sistemas operativos:

- Windows 8, 7 y XP
- Mac OS X 10.7 (Lion), 10.8 (Mountain Lion), 10.9 (Mavericks) y 10.10 (Yosemite)
- Distribuciones Linux

El IDE de simulación puede ser utilizado en las siguientes plataformas:

- Linux x86 32/64-bit
- Windows 8, 7 y XP
- Mac OS X 10.7 (Lion), 10.8 (Mountain Lion), 10.9 (Mavericks) y 10.10 (Yosemite)

A.2. Instalación y configuración en Windows

Las versiones soportadas por OMNeT++ para Windows son las versiones Intel de 32-bit de Windows XP y posteriores, como Windows 7 y Windows 8. Las versiones de 64-bit son también soportadas, pero en algún momento de la instalación y/o ejecución puede producirse algún problema, debido a que los ficheros binarios que trae OMNeT++ son de 32-bit y las simulaciones también serán compiladas en modo 32-bit.

Para instalar OMNeT++ en Windows, el primer paso a seguir es descargar el código fuente de OMNeT++ de y asegurarse de que el fichero que estamos descargando es el apropiado para Windows, que se encuentra bajo el nombre de *omnetpp-4.6-src-windows.zip*. En este paquete, además de los ficheros necesarios para la instalación de OMNeT++, podemos encontrar un compilador C++, un entorno de línea de comandos y todas las librerías y programas requeridos por OMNeT++.

El siguiente paso consiste en copiar el archivo ZIP descargado en el directorio en el que queremos instalarlo, teniendo cuidado de no elegir un directorio cuyo nombre contenga espacios, ya que la instalación no sería satisfactoria. Por ejemplo, no debemos copiar el archivo en el directorio *Archivos de Programa*. Después, es necesario extraer el archivo ZIP directamente desde el *Explorador de Windows* o utilizando un programa externo como *Winzip* o *7Zip*, y renombrar el directorio generado con el nombre *omnetpp-4.6*.

Una vez hecho esto, lo que debemos encontrar dentro del directorio *omnetpp-4.6* son más directorios llamados *doc*, *images*, *includes*, *tools*, etc y archivos como *mingwenv.cmd*, *configure*, *Makefile*, entre otros.

Una vez instalado OMNeT++, es necesario configurarlo antes de llevar a cabo su primera ejecución. Para ello, iniciamos *mingwenv.cmd (*ejecutando en *Modo Administrador* en Windows 7 y 8*)*. Aparecerá una consola que contiene *MSYS*, una colección de utilidades GNU que permiten compilar aplicaciones y programas que dependen de herramientas UNIX en entornos no UNIX, como Windows (Figura A.1). Desde ella vamos a configurar OMNeT++. Lo primero que debemos hacer es comprobar el contenido el fichero

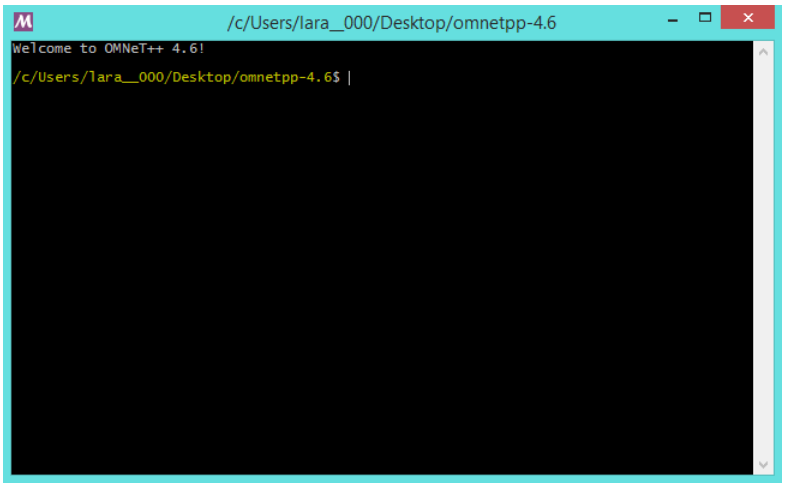


Figura A.1. Consola MinGW en Windows 8

configure.user, para asegurarnos de que contiene la configuración que necesitamos, aunque en la mayoría de los casos no será necesario hacer ningún cambio en este fichero. Para hacerlo, podemos utilizar cualquier editor de texto, como *notepad* o *notepad++.*

Después de esto, será necesario introducir el comando *$ ./configure* para llevar a cabo el primer paso de la configuración. Si no hemos seguido los pasos anteriormente descritos correctamente es posible que la ejecución de este comando nos de algún problema relacionado con el compilador *gcc.* Error que no debería aparecer puesto que *gcc* viene incluido con *MinGW.* Una vez haya terminado la ejecución de este comando satisfactoriamente, tendremos que introducir el comando $ *make.* Este proceso de compilación creará archivos binarios de depuración y ejecución.

Para finalizar la instalación y comprobar que ha sido satisfactoria, vamos a lanzar OMNeT++, escribiendo en la misma consola el comando *$ omnetpp* y automáticamente aparecerá el programa. Para no utilizar la consola cada vez que queramos lanzar OMNeT++, se recomienda crear un acceso directo en el Escritorio, enviando allí el fichero *omnetpp.exe* (contenido en el directorio *omnetpp-4.6/ide)*, bien copiando y pegando en el Escritorio o mediante la orden Enviar a > Escritorio.

A.3. Instalación y configuración en Mac OS X

Las versiones de Mac OS X que han sido testeadas y probadas son la 10.7 (Lion), 10.8 (Mountain Lion), 10.9 (Mavericks) y 10.10 (Yosemite).

En primer lugar, Mac OS X requiere una serie de paquetes necesarios para la ejecución de OMNeT++, que habrá que instalar previamente en caso de no haberlo hecho antes. Estos paquetes son el *JRE* (Java Runtime Environment) para Mac OS X, ya que en ninguna de las versiones viene instalado por defecto, la *línea de comandos para desarrolladores de OS X* y el *proyecto Quartz*, que contiene algunas cabeceras requeridas para simulaciones y entorno de ejecución. Para instalar cada una de ellas, nos dirigiremos a la dirección correspondiente:

- JRE: http://support.apple.com/kb/DL1572
- Línea de comandos para OS X: http://developer.apple.com/downloads/index.action?=Command%20Line%20Tools
- Proyecto Quartz: http://xquartz.macosforge.org/

Si nuestra versión de Mac OS X es la 10.9 (Mavericks) o la 10.10 (Yosemite), será necesario instalar además gdb (depurador), ya que, para estas nuevas versiones, el paquete de línea de comandos para OS X no contiene ni gcc (compilador) ni gdb (depurador), ambos necesarios para el correcto funcionamiento de OMNeT++. Podemos encontrar gdb para Mac OS X en http://macports.org. Una vez descargado, procedemos a la instalación abriendo Terminal y escribiendo el comando $ sudo port install gdb.

Una vez descargados e instalados todos los prerrequisitos, podemos comenzar la instalación de OMNeT++. Para ello, descargamos el archivo comprimido de http://omnetpp.org destinado a Mac OS X y Linux, denominado *omnetpp-4.6-src.tgz*. Después copiaremos el archivo comprimido en el directorio en el que queremos instalarlo. Usualmente se utiliza el directorio Home (/Users/<tu nombre de usuario>). Para extraer el archivo, desde una terminal, escribimos el comando *$ tar zxvf omnetpp-4.6-src.tgz*. Se creará un subdirectorio llamado omnetpp-4.6, que contendrá todos los archivos del simulador. Alternativamente, el archivo puede descomprimirse directamente desde *Finder*.

A continuación es necesario configurar OMNeT++, por lo que comprobaremos que la configuración contenida en *configure.user* es la adecuada y la que necesitamos y después escribiremos el comando *$ ./configure*. El script ./configure detecta qué software está instalado y la configuración del sistema. Después, escribe los resultados en el archivo Makefile.inc, que es leído posteriormente durante el proceso de compilación. Cuando ./configure ha terminado satisfactoriamente, puede compilarse OMNeT++ escribiendo en la terminal el comando *$ make*.

Una vez *make* ha terminado su ejecución, podemos lanzar OMNeT++ escribiendo por terminal *$ omnetpp*.

A.4. Instalación y configuración en Linux

Las distribuciones Linux cubiertas en este apéndice, así como cualquier distribución derivada de ellas, es soportada por OMNeT++. La lista de distribuciones Linux aceptadas es la siguiente:

- Ubuntu 12.04 LTS, 13.04
- Fedora Core 18
- Red Hat Enterprise Linux Desktop Workstation 6.4
- OpenSUSE 12.3

Algunas distribuciones derivadas de las anteriores, o de otras como Debian (por ejemplo, el propio Ubuntu está basado en Debian) se muestran en la Tabla A.1.

Ubuntu	Debian	Fedora
Kubuntu	Knoppix	Simplis
Xubuntu	Mepis	Eeedora
Edubuntu	Xandros	Etc...
Linux Mint	Finnix	
BackTrack	Kali Linux	
Elementary OS	Etc...	
Etc...		

Tabla A.1. Distribuciones Linux que soportan OMNeT++

OMNeT++ requiere múltiples paquetes que han de ser necesariamente instalados para su correcto funcionamiento. Estos paquetes incluyen Java Runtime Environment y otras librerías y programas, que pueden ser instalados a través de repositorios accesibles a través de las distintas distribuciones Linux. Generalmente, se necesitan permisos de superusuario para instalar estos paquetes.

No todos los paquetes están disponibles a través de repositorios software, algunos de ellos han de ser descargados de otras páginas web e instalados manualmente.

Podemos descargar OMNeT++ de http://omnetpp.org, asegurándonos primero de que el archivo es el adecuado para Linux, que es el que se encuentra bajo el nombre de *omnetpp-4.6-src.tgz*. Una vez descargado, copiamos el archivo en el directorio donde queremos instalarlo. Se recomienda utilizar el directorio /home/<tu_nombre_de_usuario>. Para descomprimir el archivo, abrimos una terminal y escribimos el comando $ tar zxvf omnetpp-4.6-src.tgz. Esto creará un subdirectorio omnetpp-4.6 que contiene todos los archivos de OMNeT++.

Una vez hecho esto, necesitamos configurar OMNeT++. Para ello, escribiremos en la terminal el comando $ ./configure. El script ./configure detecta qué software está instalado y la configuración del sistema. Después, escribe los resultados en el archivo Makefile.inc, que es leído posteriormente durante el proceso de compilación. Cuando ./configure ha terminado satisfactoriamente, puede compilarse OMNeT++ escribiendo en la terminal el comando *$make*.

Una vez make ha finalizado satisfactoriamente, ya podemos lanzar OMNeT++ escribiendo en la terminal $ omnetpp.

Apéndice B

Instalación de INET

Para instalar el framework INET, podemos hacerlo de dos formas:

- Descargando el archivo con la última versión de INET de su página web: http://inet.omnetpp.org, descomprimiéndolo después y siguiendo las instrucciones que aparecen en el archivo INSTALL del directorio generado a partir del archivo comprimido (Figura B.1). Generalmente será necesario importar el nuevo proyecto dentro del workspace a través de File>Import en OMNeT++.

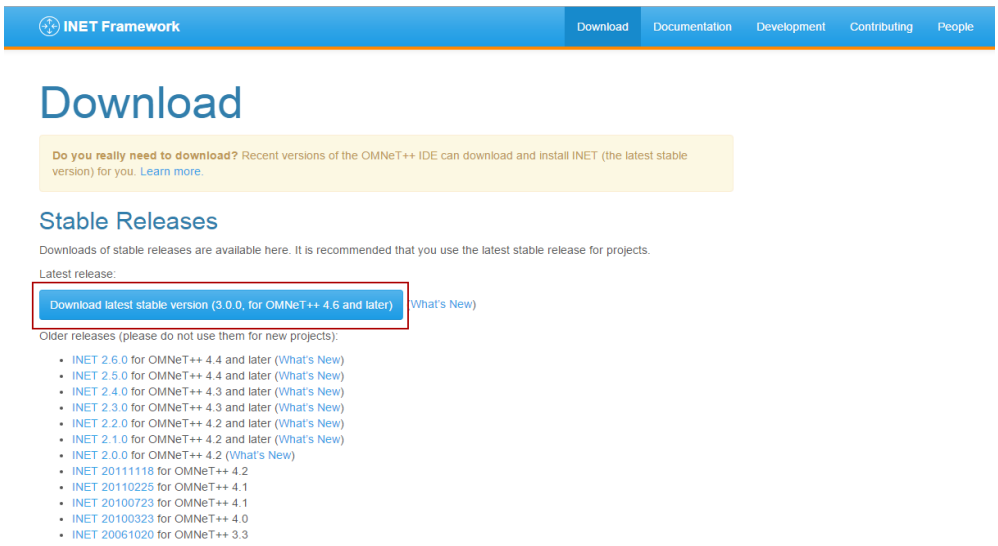


Figura B.1. Descarga del paquete INET desde http://inet.omnetpp.org/

- A través de OMNeT++. Nos dirigiremos a Help > Install Simulation Models, marcamos INET Framework (que aparecerá en su última versión) y hacemos click en Install Project (Figura B.2). Una vez hecho esto, el paquete queda instalado y listo para su uso.

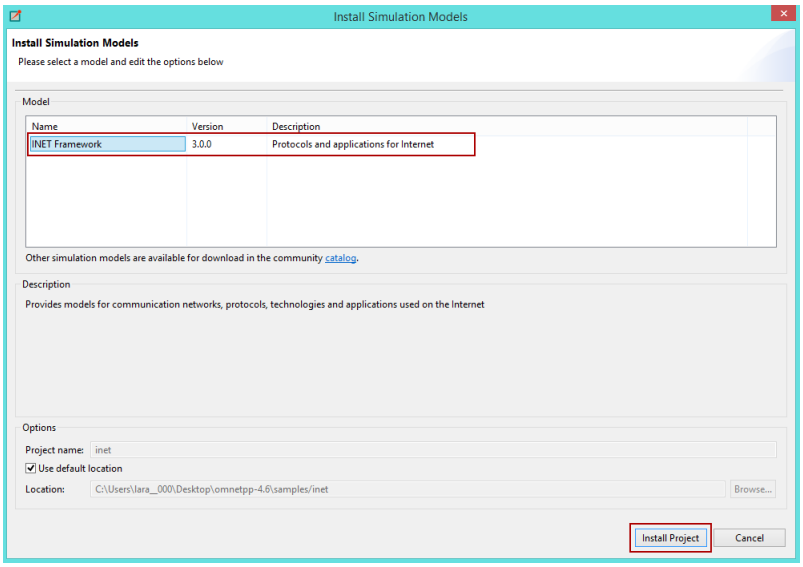


Figura B.2. Descarga del paquete INET desde OMNeT++

Apéndice C

Configuraciones

C.1. Escenario BGP-4

C.1.1. Topología

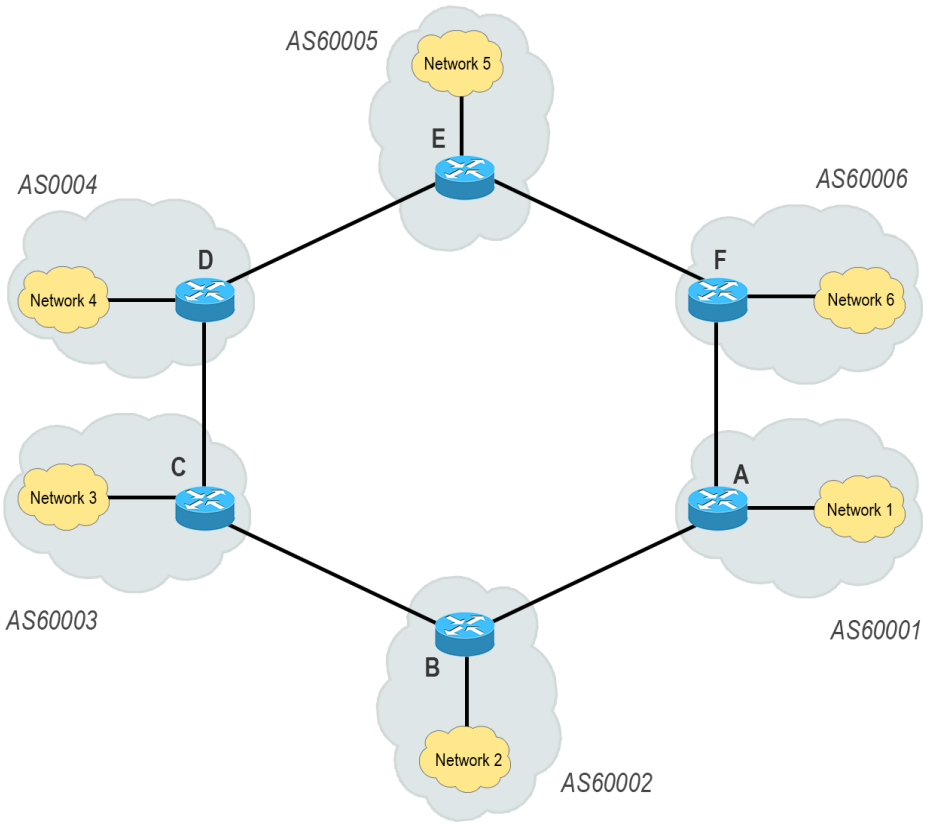


Figura C.1. Topología del escenario BGP-4 utilizado para la simulación

C.1.2. Configuración de la topología (*network.ned*)

```
import inet.common.scenario.ScenarioManager;
import
inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.node.bgp.BGPRouter;
import inet.node.inet.StandardHost;
import inet.common.misc.ThruputMeteringChannel;

//
// TODO documentation
//
network BGP4FinalNetwork2
{
    @display("bgb=910,676;bgi=background/nubes-esc-2");
    types:
        channel LINK_100 extends ThruputMeteringChannel
        {
            parameters:
                delay = 0;
                datarate = 100Mbps;
                thruputDisplayFormat = "#N";
        }
    submodules:
        A: BGPRouter {
            @display("p=713,408");
            gates:
                pppg[2];
                ethg[1];
        }
        B: BGPRouter {
            @display("p=456,533");
            gates:
                pppg[2];
                ethg[1];
        }
        C: BGPRouter {
            @display("p=205,414");
            gates:
                pppg[2];
```

```
        ethg[1];
}
D: BGPRouter {
    @display("p=205,226");
    gates:
        pppg[2];
        ethg[1];
}
E: BGPRouter {
    @display("p=456,116");
    gates:
        pppg[2];
        ethg[1];
}
F: BGPRouter {
    @display("p=713,226");
    gates:
        pppg[2];
        ethg[1];
}
HA: StandardHost {
    @display("p=827,408;i=device/pc");
    gates:
        ethg[1];
}
HB: StandardHost {
    @display("p=456,611;i=device/pc");
    gates:
        ethg[1];
}
HC: StandardHost {
    @display("p=104,414;i=device/pc");
    gates:
        ethg[1];
}
HD: StandardHost {
    @display("p=104,226;i=device/pc");
    gates:
        ethg[1];
}
HE: StandardHost {
    @display("p=456,50;i=device/pc");
```

```
            gates:
                ethg[1];
        }
        HF: StandardHost {
            @display("p=819,226;i=device/pc");
            gates:
                ethg[1];
        }
        configurator: IPv4NetworkConfigurator {
            @display("p=807,120;is=n");
            config = xmldoc("IPv4Config.xml");
            addStaticRoutes = false;
            addDefaultRoutes = false;
            addSubnetRoutes = false;
        }
        scenarioManager: ScenarioManager {
            @display("p=807,56;i=block/timer;is=n");
            script = xmldoc("scenario.xml");
        }
    connections allowunconnected:
        A.pppg[0] <--> LINK_100 { @display("ls=red,3,s"); }
<--> B.pppg[1];
        B.pppg[0] <--> LINK_100 <--> C.pppg[1];
        C.pppg[0] <--> LINK_100 <--> D.pppg[1];
        D.pppg[0] <--> LINK_100 <--> E.pppg[1];
        F.pppg[0] <--> LINK_100 { @display("ls=#80FF80,3,s");
} <--> A.pppg[1];
        F.ethg[0] <--> LINK_100 <--> HF.ethg[0];
        A.ethg[0] <--> LINK_100 <--> HA.ethg[0];
        B.ethg[0] <--> LINK_100 <--> HB.ethg[0];
        C.ethg[0] <--> LINK_100 <--> HC.ethg[0];
        D.ethg[0] <--> LINK_100 <--> HD.ethg[0];
        E.ethg[0] <--> LINK_100 <--> HE.ethg[0];
        E.pppg[0] <--> LINK_100 <--> F.pppg[1];
}
```

C.1.3. Configuración BGP (*BGPConfig.xml*)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<BGPConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
              xsi:schemaLocation="BGP.xsd">

    <TimerParams>
        <connectRetryTime> 120 </connectRetryTime>
        <holdTime> 180 </holdTime>
        <keepAliveTime> 60 </keepAliveTime>
        <startDelay> 15 </startDelay>
    </TimerParams>

   <AS id="60001">
      <Router interAddr="192.168.1.254"/>
   </AS>
    <AS id="60002">
      <Router interAddr="192.168.2.254"/>
   </AS>
    <AS id="60003">
      <Router interAddr="192.168.3.254"/>
   </AS>
    <AS id="60004">
      <Router interAddr="192.168.4.254"/>
   </AS>
    <AS id="60005">
      <Router interAddr="192.168.5.254"/>
   </AS>
    <AS id="60006">
      <Router interAddr="192.168.6.254"/>
   </AS>

   <Session id="1">
       <Router exterAddr= "10.10.10.1" />
       <Router exterAddr= "10.10.10.2" />
   </Session>
    <Session id="2">
        <Router exterAddr= "10.10.20.1" />
        <Router exterAddr= "10.10.20.2" />
    </Session>
```

```
    <Session id="3">
        <Router exterAddr= "10.10.30.1" />
        <Router exterAddr= "10.10.30.2" />
    </Session>
    <Session id="4">
        <Router exterAddr= "10.10.40.1" />
        <Router exterAddr= "10.10.40.2" />
    </Session>
    <Session id="5">
        <Router exterAddr= "10.10.50.1" />
        <Router exterAddr= "10.10.50.2" />
    </Session>
    <Session id="6">
        <Router exterAddr= "10.10.60.1" />
        <Router exterAddr= "10.10.60.2" />
    </Session>
</BGPConfig>
```

C.1.4. Configuración IPv4 (*IPv4Config.xml*)

```
<config>
      <interface hosts='A' names='ppp0' address='10.10.20.1'
netmask='255.255.255.0' />
      <interface hosts='A' names='ppp1' address='10.10.10.1'
netmask='255.255.255.0' />
      <interface hosts='B' names='ppp0' address='10.10.30.1'
netmask='255.255.255.0' />
      <interface hosts='B' names='ppp1' address='10.10.20.2'
netmask='255.255.255.0' />
      <interface hosts='C' names='ppp0' address='10.10.40.1'
netmask='255.255.255.0' />
      <interface hosts='C' names='ppp1' address='10.10.30.2'
netmask='255.255.255.0' />
      <interface hosts='D' names='ppp0' address='10.10.50.1'
netmask='255.255.255.0' />
      <interface hosts='D' names='ppp1' address='10.10.40.2'
netmask='255.255.255.0' />
      <interface hosts='E' names='ppp0' address='10.10.60.1'
netmask='255.255.255.0' />
```

```
      <interface hosts='E' names='ppp1' address='10.10.50.2'
netmask='255.255.255.0' />
      <interface hosts='F' names='ppp0' address='10.10.10.2'
netmask='255.255.255.0' />
      <interface hosts='F' names='ppp1' address='10.10.60.2'
netmask='255.255.255.0' />
      <interface hosts='A' names='eth0'
address='192.168.1.254' netmask='255.255.255.0' />
      <interface hosts='B' names='eth0'
address='192.168.2.254' netmask='255.255.255.0' />
      <interface hosts='C' names='eth0'
address='192.168.3.254' netmask='255.255.255.0' />
      <interface hosts='D' names='eth0'
address='192.168.4.254' netmask='255.255.255.0' />
      <interface hosts='E' names='eth0'
address='192.168.5.254' netmask='255.255.255.0' />
      <interface hosts='F' names='eth0'
address='192.168.6.254' netmask='255.255.255.0' />
      <interface hosts='HA' names='eth0'
address='192.168.1.1' netmask='255.255.255.0' metric='1' />
      <interface hosts='HB' names='eth0'
address='192.168.2.1' netmask='255.255.255.0' metric='1' />
      <interface hosts='HC' names='eth0'
address='192.168.3.1' netmask='255.255.255.0' metric='1' />
      <interface hosts='HD' names='eth0'
address='192.168.4.1' netmask='255.255.255.0' metric='1' />
      <interface hosts='HE' names='eth0'
address='192.168.5.1' netmask='255.255.255.0' metric='1' />
      <interface hosts='HF' names='eth0'
address='192.168.6.1' netmask='255.255.255.0' metric='1' />
</config>
```

C.1.5. Configuración OSPF (*OSPFConfig.xml*)

```
<?xml version="1.0"?>
<OSPFASConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="OSPF.xsd">

  <!-- Areas -->
  <Area id="0.0.0.1">
    <AddressRange address="HA" mask="HA" status="Advertise"
/>
  </Area>
  <Area id="0.0.0.2">
    <AddressRange address="HB" mask="HB" status="Advertise"
/>
  </Area>
  <Area id="0.0.0.3">
    <AddressRange address="HC" mask="HC" status="Advertise"
/>
  </Area>
  <Area id="0.0.0.4">
    <AddressRange address="HD" mask="HD" status="Advertise"
/>
  </Area>
  <Area id="0.0.0.5">
    <AddressRange address="HE" mask="HE" status="Advertise"
/>
  </Area>
  <Area id="0.0.0.6">
    <AddressRange address="HF" mask="HF" status="Advertise"
/>
  </Area>

  <!-- Routers -->
  <Router name="A" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.1"
interfaceOutputCost="1" routerPriority="1" />
  </Router>
  <Router name="B" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.2"
interfaceOutputCost="1" routerPriority="1" />
  </Router>
```

```
  <Router name="C" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.3"
interfaceOutputCost="1" routerPriority="1" />
  </Router>
  <Router name="D" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.4"
interfaceOutputCost="1" routerPriority="1" />
  </Router>
  <Router name="E" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.5"
interfaceOutputCost="1" routerPriority="1" />
  </Router>
  <Router name="F" RFC1583Compatible="true">
    <BroadcastInterface ifName="eth0" areaID="0.0.0.6"
interfaceOutputCost="1" routerPriority="1" />
  </Router>
</OSPFASConfig>
```

C.1.6. Configuración LocalPref (*localPrefConfig.xml*)

```
<lpconfig>

            <localpref routerid= '192.168.1.254'
peer='10.10.20.2' destination="192.168.2.0" lp="50" />
            <localpref routerid= '192.168.1.254'
peer='10.10.10.2' destination="192.168.2.0" lp="300" />
            <localpref routerid= '192.168.1.254'
peer='10.10.20.2' destination="192.168.3.0" lp="50" />
            <localpref routerid= '192.168.1.254'
peer='10.10.10.2' destination="192.168.3.0" lp="100" />
            <localpref routerid= '192.168.1.254'
peer='10.10.20.2' destination="192.168.4.0" lp="50" />
            <localpref routerid= '192.168.1.254'
peer='10.10.10.2' destination="192.168.4.0" lp="300" />
            <localpref routerid= '192.168.1.254'
peer='10.10.10.2' destination="192.168.5.0" lp="300" />
```

```
            <localpref routerid= '192.168.1.254'
peer='10.10.20.2' destination="192.168.5.0" lp="100" />
            <localpref routerid= '192.168.1.254'
peer='10.10.10.2' destination="192.168.6.0" lp="300" />
            <localpref routerid= '192.168.1.254'
peer='10.10.20.2' destination="192.168.6.0" lp="50" />

            <localpref routerid= '192.168.2.254'
peer='10.10.20.1' destination="192.168.1.0" lp="50" />
            <localpref routerid= '192.168.2.254'
peer='10.10.30.2' destination="192.168.1.0" lp="100" />
            <localpref routerid= '192.168.2.254'
peer='10.10.30.2' destination="192.168.3.0" lp="100" />
            <localpref routerid= '192.168.2.254'
peer='10.10.20.1' destination="192.168.3.0" lp="50" />
            <localpref routerid= '192.168.2.254'
peer='10.10.30.2' destination="192.168.4.0" lp="100" />
            <localpref routerid= '192.168.2.254'
peer='10.10.20.1' destination="192.168.4.0" lp="50" />
            <localpref routerid= '192.168.2.254'
peer='10.10.30.2' destination="192.168.5.0" lp="100" />
            <localpref routerid= '192.168.2.254'
peer='10.10.20.1' destination="192.168.5.0" lp="50" />
            <localpref routerid= '192.168.2.254'
peer='10.10.20.1' destination="192.168.6.0" lp="50" />
            <localpref routerid= '192.168.2.254'
peer='10.10.30.2' destination="192.168.6.0" lp="100" />

            <localpref routerid= '192.168.3.254'
peer='10.10.40.2' destination="192.168.1.0" lp="100" />
            <localpref routerid= '192.168.3.254'
peer='10.10.30.1' destination="192.168.1.0" lp="50" />
            <localpref routerid= '192.168.3.254'
peer='10.10.30.1' destination="192.168.2.0" lp="100" />
            <localpref routerid= '192.168.3.254'
peer='10.10.40.2' destination="192.168.2.0" lp="50" />
            <localpref routerid= '192.168.3.254'
peer='10.10.40.2' destination="192.168.4.0" lp="100" />
            <localpref routerid= '192.168.3.254'
peer='10.10.30.1' destination="192.168.4.0" lp="50" />
            <localpref routerid= '192.168.3.254'
peer='10.10.40.2' destination="192.168.5.0" lp="100" />
```

```
            <localpref routerid= '192.168.3.254'
peer='10.10.30.1' destination="192.168.5.0" lp="50" />
            <localpref routerid= '192.168.3.254'
peer='10.10.30.1' destination="192.168.6.0" lp="50" />
            <localpref routerid= '192.168.3.254'
peer='10.10.40.2' destination="192.168.6.0" lp="100" />

            <localpref routerid= '192.168.4.254'
peer='10.10.40.1' destination="192.168.1.0" lp="50" />
            <localpref routerid= '192.168.4.254'
peer='10.10.50.2' destination="192.168.1.0" lp="100" />
            <localpref routerid= '192.168.4.254'
peer='10.10.50.2' destination="192.168.2.0" lp="50" />
            <localpref routerid= '192.168.4.254'
peer='10.10.40.1' destination="192.168.2.0" lp="100" />
            <localpref routerid= '192.168.4.254'
peer='10.10.40.1' destination="192.168.3.0" lp="100" />
            <localpref routerid= '192.168.4.254'
peer='10.10.50.2' destination="192.168.3.0" lp="50" />
            <localpref routerid= '192.168.4.254'
peer='10.10.40.1' destination="192.168.5.0" lp="50" />
            <localpref routerid= '192.168.4.254'
peer='10.10.50.2' destination="192.168.5.0" lp="100" />
            <localpref routerid= '192.168.4.254'
peer='10.10.50.2' destination="192.168.6.0" lp="100" />
            <localpref routerid= '192.168.4.254'
peer='10.10.40.1' destination="192.168.6.0" lp="50" />

            <localpref routerid= '192.168.5.254'
peer='10.10.50.1' destination="192.168.1.0" lp="50" />
            <localpref routerid= '192.168.5.254'
peer='10.10.60.2' destination="192.168.1.0" lp="300" />
            <localpref routerid= '192.168.5.254'
peer='10.10.50.1' destination="192.168.2.0" lp="100" />
            <localpref routerid= '192.168.5.254'
peer='10.10.60.2' destination="192.168.2.0" lp="50" />
            <localpref routerid= '192.168.5.254'
peer='10.10.50.1' destination="192.168.3.0" lp="100" />
            <localpref routerid= '192.168.5.254'
peer='10.10.60.2' destination="192.168.3.0" lp="50" />
            <localpref routerid= '192.168.5.254'
peer='10.10.50.1' destination="192.168.4.0" lp="100" />
```

```
            <localpref routerid= '192.168.5.254'
peer='10.10.60.2' destination="192.168.4.0" lp="50" />
            <localpref routerid= '192.168.5.254'
peer='10.10.50.1' destination="192.168.6.0" lp="50" />
            <localpref routerid= '192.168.5.254'
peer='10.10.60.2' destination="192.168.6.0" lp="100" />


            <localpref routerid= '192.168.6.254'
peer='10.10.10.1' destination="192.168.1.0" lp="300" />
            <localpref routerid= '192.168.6.254'
peer='10.10.60.1' destination="192.168.1.0" lp="100" />
            <localpref routerid= '192.168.6.254'
peer='10.10.60.1' destination="192.168.2.0" lp="100" />
            <localpref routerid= '192.168.6.254'
peer='10.10.10.1' destination="192.168.2.0" lp="50" />
            <localpref routerid= '192.168.6.254'
peer='10.10.10.1' destination="192.168.3.0" lp="50" />
            <localpref routerid= '192.168.6.254'
peer='10.10.60.1' destination="192.168.3.0" lp="100" />
            <localpref routerid= '192.168.6.254'
peer='10.10.10.1' destination="192.168.4.0" lp="50" />
            <localpref routerid= '192.168.6.254'
peer='10.10.60.1' destination="192.168.4.0" lp="100" />
            <localpref routerid= '192.168.6.254'
peer='10.10.10.1' destination="192.168.5.0" lp="50" />
            <localpref routerid= '192.168.6.254'
peer='10.10.60.1' destination="192.168.5.0" lp="100" />


</lpconfig>
```

C.1.7. Configuración inicial (*omnet.ini*)

```
[General]

sim-time-limit =1950s
debug-on-errors = false
record-eventlog = true
```

```
output-scalar-file = results.sca
output-scalar-precision = 3

cmdenv-express-mode = true
cmdenv-module-messages = false
cmdenv-event-banners = false
cmdenv-message-trace = false

tkenv-plugin-path = ../../../etc/plugins

**.udpApp[0].**.scalar-recording = true
**.scalar-recording = false
**.vector-recording = false

# ip settings
**.ip.procDelay = 1us
**.rsvp.procDelay = 1us

# NIC configuration
**.queueType = "DropTailQueue"
**.queue.frameCapacity = 100

#tcp settings
**.tcp.mss = 1024
**.tcp.advertisedWindow = 14336
#**.tcp.sendQueueClass = "TCPMsgBasedSendQueue"
#**.tcp.receiveQueueClass = "TCPMsgBasedRcvQueue"
**.bgp.dataTransferMode = "object"
**.tcp.tcpAlgorithmClass = "TCPReno"
**.tcp.recordStats = false
#**.tcp.useDataNotification = true
#**.A1.bgp.numTcpApps = 2
#**.newKey = 100s
#**.A1.tcpType = "TCP"
#**.A1.tcpType= "TCPSessionApp"
**.A1.bgp.tClose=100s

#**.A1.bgp.tClose=100s

# OSPF configuration
**.ospfConfig = xmldoc("OSPFConfig.xml")
```

```
# bgp settings
**.bgpConfig = xmldoc("BGPConfig.xml")
**.lpConfig = xmldoc("localPrefConfig.xml")
#**.bgpsessions=xmldoc("TCPSessionsManager.xml")

[Config config1]
# Multi OPSF routing test + BGP
description = "BGP + OSPF"
network = BGP4FinalNetwork2
#UDP Host
**.H*.eth.mac.promiscuous = false

# UDPApp parameters
**.numUdpApps = 1
**.udpApp[0].messageLength = 32 bytes
**.udpApp[0].sendInterval = 2s
**.udpApp[0].localPort = 1234
**.udpApp[0].destPort = 1234

**.HA.udpApp[*].typename = "UDPSink"
**.HB.udpApp[*].typename = "UDPSink"
**.HC.udpApp[*].typename = "UDPSink"
**.HD.udpApp[*].typename = "UDPSink"
**.HE.udpApp[*].typename = "UDPSink"
**.HF.udpApp[*].typename = "UDPSink"
```

C.1.8. Configuración ScenarioManager (scenario.xml)

```
<scenario>

      <at t='500'>
            <disconnect src-module="F" src-gate="pppg[0]" />
      </at>

      <at t='800'>
```

```
            <connect src-module='F' src-gate='pppg[0]' dest-
module='A' dest-gate='pppg[1]' channel-
type='BGP4FinalNetwork2.LINK_100' />
      </at>

      <at t='1000'>
            <disconnect src-module='A' src-gate='pppg[0]' />
      </at>

      <at t='1200'>
            <connect src-module='A' src-gate='pppg[0]' dest-
module='B' dest-gate='pppg[1]' channel-
type="BGPFinalNetwork2.LINK_100" />
      </at>

</scenario>
```

Bibliografía

[1] Y. Rekhter, T. Li and S. Hares. A Border Gateway Protocol. RFC 4271 (Informational), January 2006.

[2] T. Griffin and G. Huston. BGP Wedgies. RFC 4264 (Informational), November 2005.

[3] Q. Vohra, from Juniper Networks and E. Chen, from Cisco Systems. BGP Support for Four-octet AS Number space. RFC 6793 (Informational), December 2012.

[4] E. Chen. Route Refresh Capability for BGP-4. RFC 2918 (Informational), September 2000.

[5] Fabero Jiménez, Juan Carlos. *Encaminamiento externo: BGPv4*. En: *Redes de Nueva Generación e Internet*. 85 diapositivas.

[6] A. Varga adn OpenSim Ltd. *OMNeT++ User Guide*. Version 4.6, 2014.

[7] A. Varga adn OpenSim Ltd. *OMNeT++ User Manual*. Version 4.6, 2014.

[8] A. Varga adn OpenSim Ltd. *INET Framework Manual for OMNeT++*, June 2012.

[9] H. Balakrishnan and N. Feamster. *Interdomain Internet Routing*, 2005.

[10] Cisco Systems. BGP *Best Path Selection Algorithm*. http:// http://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html

[11] Cisco Systems. *Border Gateway Protocol*. http:// http://docwiki.cisco.com/wiki/Border_Gateway_Protocol#BGP_Attributes

[12] Cisco Systems. *Sample Configuration for iBGP and eBGP With or Without a Loopback Address*. http:// http://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13751-23.html

[13] Cisco Systems. *BGP: Frequently Asked Questions*. http:// http://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/5816-bgpfaq-5816.html

[14] B. Quoitin and O. Bonaventure. *A Cooperative Approach of Interdomain Traffic Engineering*. Université Catholique de Louvain (UCL), Luvain-la-Neuve, Belgium,

[15] J. Stewart. *BGP4: Interdomain Routing in the Internet*. Addison Wesley, 1999.

[16] R. Sherwood. *Border Gateway Protocol v4*. Stanford University, California, 2009.

[17] W. Liu, C. Matthews, L. Parziale, N. Rosslot, C. Davis, J. Forrester and D.T. Britt. TCP/IP Tutorial and Technical Overview. IBM Redbooks Publication, 2006.

Never let the fear of striking out
keep you from playing the game