Capítulo 4: Simulación en OMNeT++

Para describir el comportamiento de los modelos de Transición se utiliza OMNeT++ como herramienta principal. Este capítulo tiene como objetivo dar una introducción acerca de este simulador empleado comúnmente en las redes de comunicaciones. El Apéndice C muestra de manera detallada su instalación, así como lo necesario para la creación, compilación y ejecución de archivos.

4.1 Introducción

OMNeT++ es una herramienta de modelado y simulación pública, basado en componentes modulares y con un ambiente de simulación de arquitectura abierta y con un fuerte soporte de GUI., creado por András Varga en el 2003 en la Universidad Técnica de Budapest. Es un simulador de ambiente discreto. Su área de aplicación primaria es la simulación de redes de comunicación, y debido a su arquitectura genérica y flexible, ha sido utilizada exitosamente en redes basadas en colas de espera y arquitectura de hardware. Múltiples modelos de simulación de fuente abierta han sido publicados, en el campo de las simulaciones de Internet (IP, IPv6, MPLS, etc), movilidad, simulaciones ad-hoc y otras áreas.

Ya que OMNeT++ provee una arquitectura modular, estos componentes (módulos) están programados en C++, después son ensamblados en componentes y modelos más grandes utilizando un lenguaje de alto nivel (NED). Su soporte de GUI, junto con su arquitectura modular las simulaciones pueden ser incluidas fácilmente en aplicaciones propias. Aunque OMNeT++ no es un simulador en si, actualmente está ganando popularidad como una plataforma de simulación de redes en la comunidad científica así como la industria. OMNeT++ corre para las plataformas Linux, Unix-like y Win32 (Windows 2000 y XP).

4.2 Funcionamiento de OMNeT++

OMNeT++ proporciona las herramientas básicas para realizar simulaciones, pero por sí mismo no proporciona ningún componente específico para la simulación de redes de computadoras, simulaciones de colas, simulaciones de arquitectura de sistemas o cualquier otra área. En su lugar, estas áreas de aplicación son proporcionadas por varios modelos de simulación y arquitecturas tales como Mobility Framework o INET Framework. Estos modelos son desarrollados completamente independientes a lo que es OMNeT++, y siguen sus propios ciclos.

Así pues, lo que OMNeT++ proporciona en sí es una librería clase C++ que permite la creación de componentes de simulación como módulos simples y canales; también, se proporciona la infraestructura para reunir las simulaciones de estos componentes y configurarlos en el lenguaje NED o en archivos tipo *ini*. También proporciona interfaces donde se puede observar y manipular el tiempo de ejecución o ambientes para la simulación como *Tkenv, Cmdenv*; herramientas para facilitar crear simulaciones y evaluar sus resultados como *GNED, Scalars y Plove*. Mas adelante, se mencionarán un poco más a detalle estos componentes que proporciona OMNeT++.

4.2.1 Simulación con OMNeT++

OMNeT++ brinda una arquitectura modular. Los modelos son ensamblados por componentes reutilizables: módulos, que bien configurados son reutilizables y pueden ser combinados de distintas maneras, como los bloques de LEGO™. Los módulos pueden conectarse con otros a través de puertos o *gates*, y combinarlos entre sí para formar módulos compuestos. Las conexiones son creadas dentro de un nivel básico de jerarquía: un sub-módulo puede conectarse con otro igual, o con aquel que contenga en sí un modulo compuesto. Cada modelo de simulación es un caso del modelo compuesto. Tanto los componentes como la topología se relacionan directamente con los archivos NED.

Para poder realizar una simulación en OMNeT++ se pueden seguir 6 pasos básicos, que en general, pueden ser aplicados para el modelado. A continuación un vistazo rápido de cómo utilizar OMNeT++:

1. Un modelo de OMNeT++ está formado por componentes (módulos) los cuales se comunican entre sí intercambiando mensajes. Estos módulos pueden ser anidados, es decir, que varios módulos pueden agruparse todos juntos y formar un módulo compuesto (*compound module*). Cuando se crea el modelo, se necesita crear un mapa del sistema en una jerarquía de módulos comunicados entre sí.
2. Se define la estructura del modelo en lenguaje NED (se menciona más adelante a detalle su funcionamiento y algunos ejemplos de este editor).Los archivos NED se pueden editar en cualquier editor de texto o en GNED, el editor gráfico de OMNeT++.
3. Los componentes activos del modelo (módulos simples) se programan en C++, utilizando los modelos de simulación y librería tipo *class*.
4. Se proporciona un archivo omnetpp.ini para conservar la configuración de OMNeT++ y los parámetros del modelo creado. Un archivo *config* puede describir diversas simulaciones con parámetros diferentes.
5. Se crea el programa de simulación y se corre. Al hacer esto se crea un vínculo entre el código de OMNeT++ y la interfaces de usuario que proporciona OMNeT++.
6. Los resultados de la simulación se escriben en un vector y un archivo escalar de salida. Se pueden utilizar *Plove* y *Scalars* para visualizarlos. Ver disco adjunto para ejemplo de simulación.

4.2.2 Componentes de OMNeT++

Algunos de los componentes principales que conforman a OMNeT++ son, por ejemplo, las librerías de simulación inteligente:

- Compilador para la topología de descripción de lenguaje NED (nedc)
- GUI para la ejecución del simulador, enlaces dentro de las simulaciones ejecutables (*Tkenv*)
- Interfaz al usuario de las líneas de comando de la simulación ejecutada (*Cmdenv*)
- Herramienta de graficación por vectores de graficas de resultados (*Plove*)
- Herramienta de visualización escalar de graficas de resultados (*Scalars*)
- Herramienta de documentación de modelo (*opp_neddoc*)

- Utilidades (herramienta de generación de semillas de números aleatorios, herramienta de creación de archivos, etc.)
- Documentación, simulaciones de muestra, etc.

La interfaz de usuario OMNeT++ es utilizada junto con la ejecución de la simulación. Su principal uso es mostrar el contenido del modelo de una forma visible para el usuario, para iniciar/parar la simulación, permitir al usuario intervenir y cambiar una variable u objetos dentro del modelo. Esto es importante al momento de desarrollar y depurar el proyecto. El hecho de tener acceso y poder modificar el modelo, permite a los usuarios conocer el comportamiento de lo que se modela. Una interfaz grafica agradable a la vista del usuario permite demostrar como funciona el modelo internamente. Actualmente, dos interfaces para el usuario son manejadas:

- *Tkenv*: Gráfica basada en Tk, para usuarios de Windows (X-Window, Win95, WinNT etc.)
- *Cmdenv*: Interfaz al usuario de líneas de comando para la ejecución en serie.

Las pruebas y depuraciones se hacen sobre *Tkenv*, mientras que los experimentos de simulación se hacen con *Cmdenv*.

Tkenv es una interfaz grafica portátil que apoya la ejecución interactiva de la simulación, la traza y la depuración (*debugging*) de errores. *Tkenv* es recomendable en la etapa de desarrollo de una simulación o para presentaciones, ya que permite una mirada detallada, de manera gráfica, del estado de la simulación en cualquier punto de la ejecución, y de lo que sucede internamente de la red creada. Sus características más importantes son:

- Ventanas separadas para cada modulo de texto de salida
- Pueden verse en una ventana mensajes planificados acerca del progreso de la simulación
- Ejecución evento por evento
- Animación de la ejecución
- Puntos de prueba (*breakpoints*)
- Ventanas de inspección para examinar o cambiar los objetos y las variables del modelo

- Despliegue gráfico de los resultados de la simulación durante la ejecución. Estos resultados pueden ser desplegados como histogramas o diagramas basados en el tiempo
- La simulación puede reiniciarse en cualquier punto.
- Imágenes acerca del modelo, sus objetos, variables, pueden ser creados.

Cuando se utiliza junto con un depurador (*debugger*) de C++, se crea un ambiente propicio para la prueba y corrección de errores. La posibilidad que se tiene de mostrar los resultados de la simulación durante la ejecución de la misma, puede acelerar el proceso de verificación de la correcta operación del programa de simulación y proveer un ambiente para experimentar con el modelo durante la ejecución.

Cmdenv es la interfaz para las líneas de comando, es una pequeña, portátil y rápida interfaz de usuario que compila y corre en todas las plataformas (ya sea Unix o Windows). *Cmdenv* esta diseñado para una ejecución en serie.

Plove es una herramienta bastante útil para graficar y analizar los archivos finales que genera OMNeT++. *Plove* puede graficar vectores de salida en una o más gráficas. Se puede especificar el estilo de gráfica (líneas, puntos, etc) para cada vector. Se pueden modificar las etiquetas de los ejes, escalas, etc.

Las gráficas se pueden salvar en archivos de distintos formatos (EPS, GIF, etc) o copiarlas directamente como un *clipboard* (figura 4.1). Se pueden filtrar los resultados obtenidos (promediarlos, truncarlos, reducirlos, histogramas, etc), así como también se pueden crear nuevos filtros según sea la necesidad.

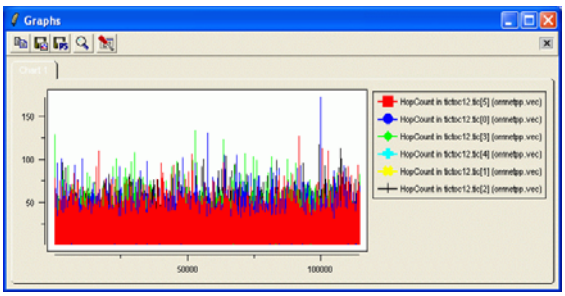

Figura 4.1: Ejemplo de una medición y gráfica hecha con Plove.

Scalars es una herramienta bastante útil para crear tablas y gráficas x-y de la salida escalar de OMNeT++. Estas gráficas se pueden guardar en formatos tales como GIF.

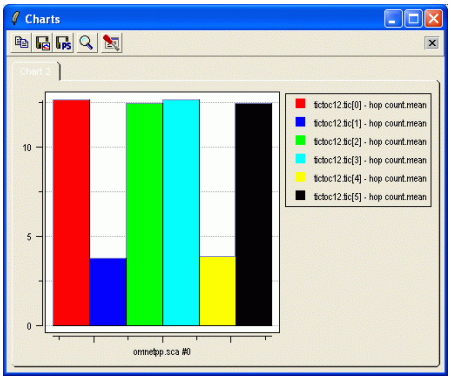

Figura 4.2: Ejemplo de una gráfica tomada por Scalars.

4.3 Modelado de IPv6 en GNED

Para empezar el modelado en IPv6, se comienza diseñando un ambiente gráfico. Éste representa la interfaz que va a mostrar al usuario los procesos programados en C++. Este editor gráfico sólo diseña la estructura básica del modelo a crear, su comportamiento esta dado mediante un programa en C++. Primero, se creó una topología base para realizar todos los modelos. Ésta es arbitraria y sólo para fines de simulación (ver Figura 4.3). Esta topología está compuesta por dos Sistemas Finales en los extremos (*End_system*) y un sistema de ruteadores conectados en forma de red que se encargan de direccionar la información.

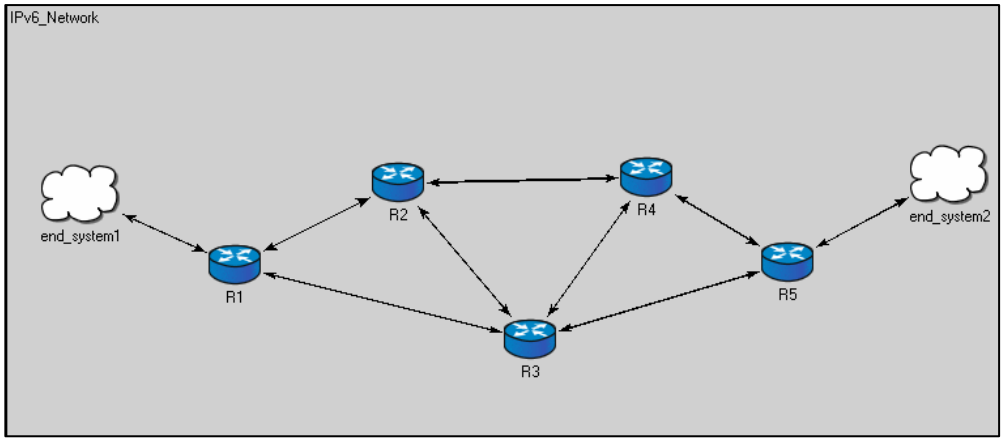

Figura 4.3: Topología básica de simulación.

Cada uno de los dispositivos en esta topología es representado internamente por una serie de submódulos que indican el comportamiento del módulo que lo contiene. El apéndice E muestra el código generado en GNED para este gráfico. Las Figuras 4.4 y 4.5, muestran la estructura interna de un *end_system* (que equivale a un *host*) y un ruteador, respectivamente.

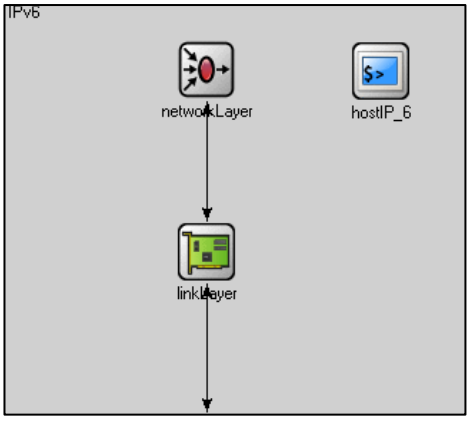

Figura 4.4: Estructura básica de un end_system, que cuenta con una capa de red y una de enlace de datos.

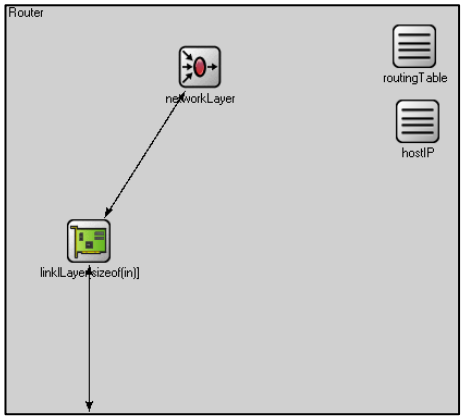

Figura 4.5: Estructura básica de un ruteador.

El módulo que pertenece al *host* (Figura 4.4) está compuesto por tres bloques básicos:

- *networkLayer (IP.cpp)*
- *linkLayer (EthPP.cpp)*
- *hostIP (HostIP.cpp)*

El bloque de *networkLayer* representa a la Capa de Red y está definido por su archivo *IP.cpp*. *linkLayer* se relaciona con la Capa de Enlace de datos y está definido en *EthPPP.cpp*; por último, *hostIP* contiene la dirección IP del módulo y se define en *HostIP*.

El módulo de *router* (Figura 4.5) está compuesto por los mismos bloques básicos que el *host* y a simple vista parecerá que son exactamente idénticos, pero lo que los diferencia el uno del otro es el archivo creado en C++ que describe su comportamiento.

Estos módulos creados se pueden utilizar tantas veces se necesiten ya que utilizan el mismo archivo C++ creado. A continuación, se mostrará el trabajo gráfico de cada uno de los modelos a utilizar en las simulaciones. El código en C++ principal de cada módulo esta contenido en el disco adjunto.

Durante la sección de pruebas y resultados, se mostrarán los eventos de salida generados por éste código.

4.3.1 Modelado de Pila-dual en GNED

Como se mencionó en el capítulo 3, la pila-dual es un mecanismo de transición que tiene dos pilas, cada una con un protocolo de IP diferente. Cada pila es activada según sea la necesidad del dispositivo; es decir, si se recibe un paquete de información de tipo IPv4, la pila activada será la correspondiente a esta versión. Lo contrario ocurre si la información es de tipo IPv6.

La comunicación que ofrece este tipo de mecanismo es sólo entre dispositivos del mismo tipo. Así pues, un dispositivo IPv4 puede comunicarse con uno IPv6 siempre y cuando éste sea de pila doble y su pila IPv4 sea activada. La Figura 4.6 muestra la topología básica manejada en la simulación donde cada dispositivo mostrado sostiene en su interior una doble pila.

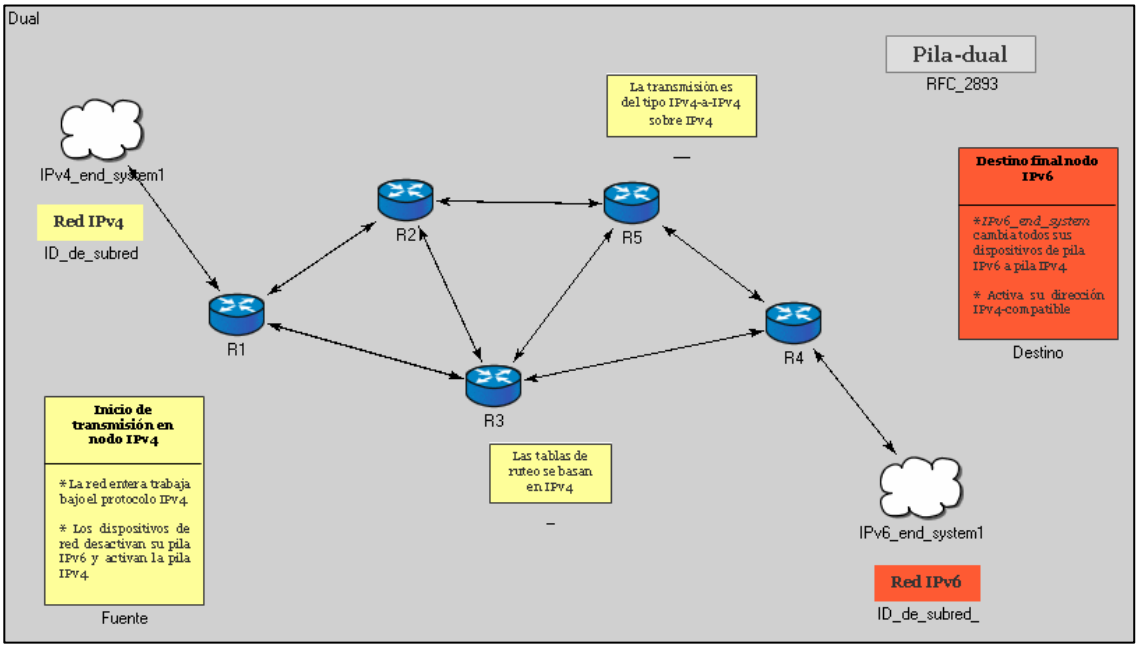

Figura 4.6: Modelado en GNED del mecanismo de transición de pila-dual.

Si la transmisión se inicia en un nodo de red IPv4, la estructura de ruteo por la cual pasa, también tiene que ser del tipo IPv4 (esto activando la pila correspondiente). Si el nodo final es del tipo IPv6, al detectar una versión diferente a la actual, procede a cambiar de pila para aceptar la información entrante. Este mismo proceso se realiza si el nodo origen de la información es del tipo IPv6. Para que estos dispositivos puedan manejar ambos protocolos, es necesario como ya se mencionó, que exista una pila doble. Como muestran las Figura 4.7 y 4.8, ya sea un usuario o un ruteador, las dos pilas se encuentran presentes y siempre dispuestas a ser habilitadas o deshabilitadas según sea necesario.

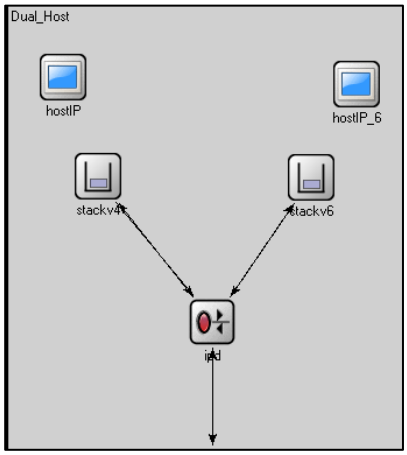

Figura 4.7: Estructura básica de un host de pila doble.

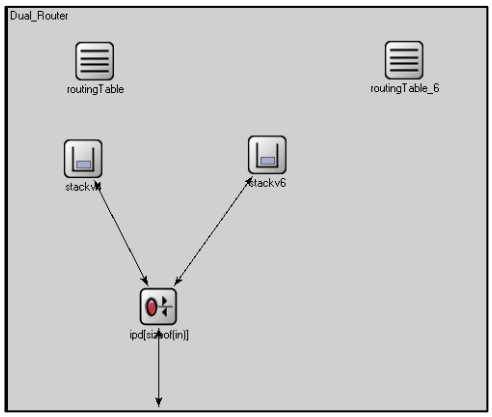

Figura 4.8: Estructura básica de un ruteador de pila doble.

Lo hecho anteriormente representa un modelado gráfico. Para describir su comportamiento es necesario crear un archivo en C++. A continuación se muestran los principales códigos que describen el comportamiento esencial del modelo.

4.3.2 Modelado de Traducción en GNED

Como su nombre lo menciona, este mecanismo traduce los encabezados de una versión de IP a la otra. Tiene la finalidad de proveer comunicación entre redes de distinto protocolo a comparación de la pila doble que sólo permite la comunicación entre redes del mismo tipo.

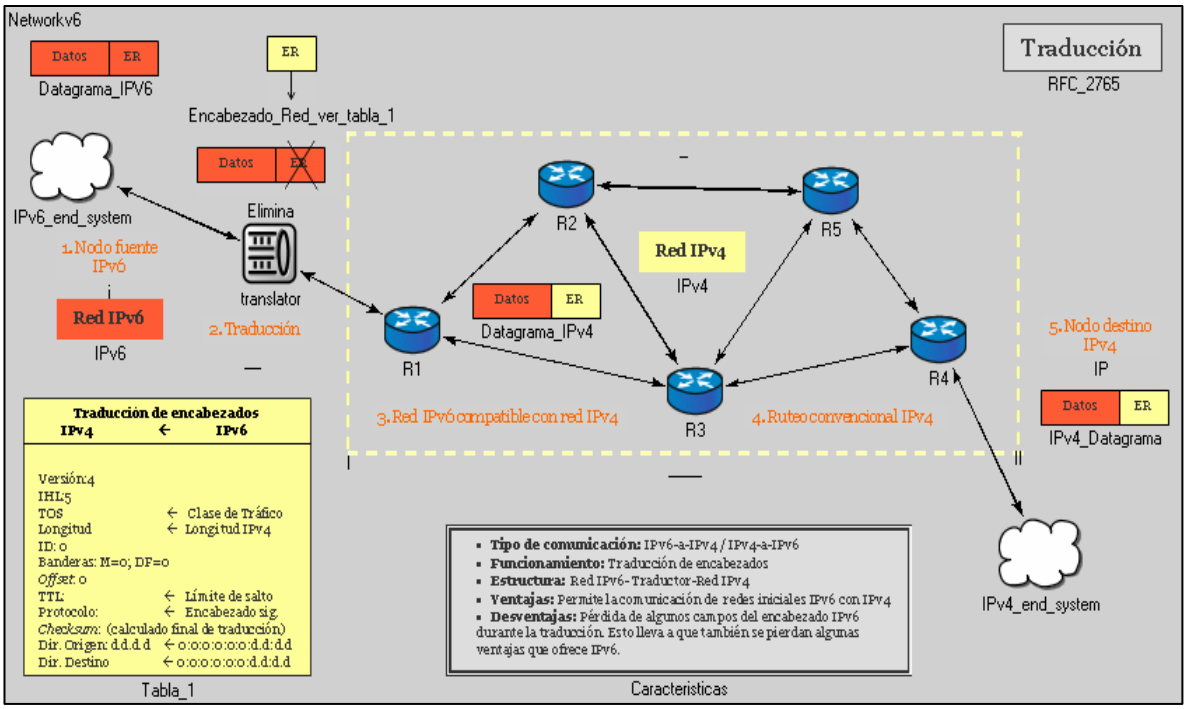

Figura 4.9: Modelado del mecanismo de Traducción.

Para que esta traducción se realice, es necesaria la presencia de un traductor. Este dispositivo es similar a un ruteador de pila doble, ya que también contiene dos pilas que son las encargadas de realizar la traducción según se necesite (ver Figura 4.10).

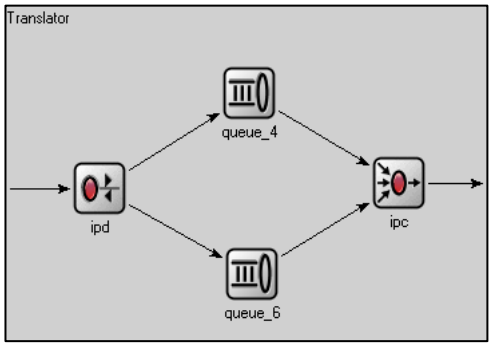

Figura 4.10: Estructura del traductor de encabezados.

Este traductor recibe la información, y mediante un discriminador selecciona del encabezado de red la versión del protocolo utilizado. En base a esto, se direcciona a una de las dos pilas disponibles y se procede a hacer la traducción de encabezados (ver capítulo 3 para mayor referencia). Una vez hecha la traducción, se procesa la información regularmente. Una desventaja de la traducción de encabezados, es la pérdida del encabezado de red original de la información. Esto lleva a que ciertos campos se pierdan y con ello algunas características. Generalmente este mecanismo es utilizado para permitir la comunicación de IPv6 con IPv4 sobre un ambiente de ruteo IPv4; aunque claro, la situación inversa se presenta.

4.3.3 Modelado de *Tunneling* en GNED

El *tunneling* tiene la característica de permitir la comunicación entre ambientes mixtos. Esto es, que IPv6 pueda se transportado a través de una red IPv4 con la ventaja, a diferencia de la traducción, que los encabezados originales y sus características no se pierden durante el proceso. A grandes rasgos, este mecanismo crea "túneles" de comunicación entre nodos distantes separados por ambiente diferentes al propio. Un túnel, tal y como lo conocemos en las autopistas y carreteras, tiene un principio y un fin. Análogamente los "túneles" creados tienen un nodo de inicio y otro de fin. La Figura 4.11 muestra la primera de éstas técnicas: *Tunneling* configurado. Esta técnica asigna el fin del "túnel" de acuerdo a la información de configuración de inicio del túnel y tablas.

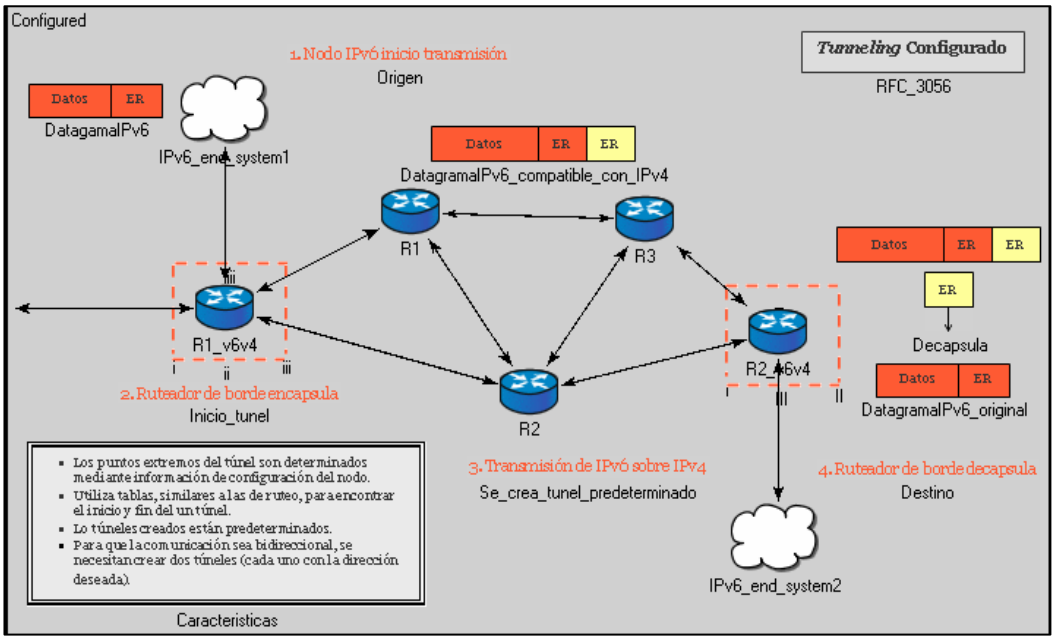

Figura 4.11: Técnica de tunneling Configurado.

Esta técnica se basa en direcciones IPv4 compatibles (ver Capítulo 2). La dirección de fin de túnel está determinada por la dirección IPv4-compatible contenida en la dirección destino del paquete original de información.

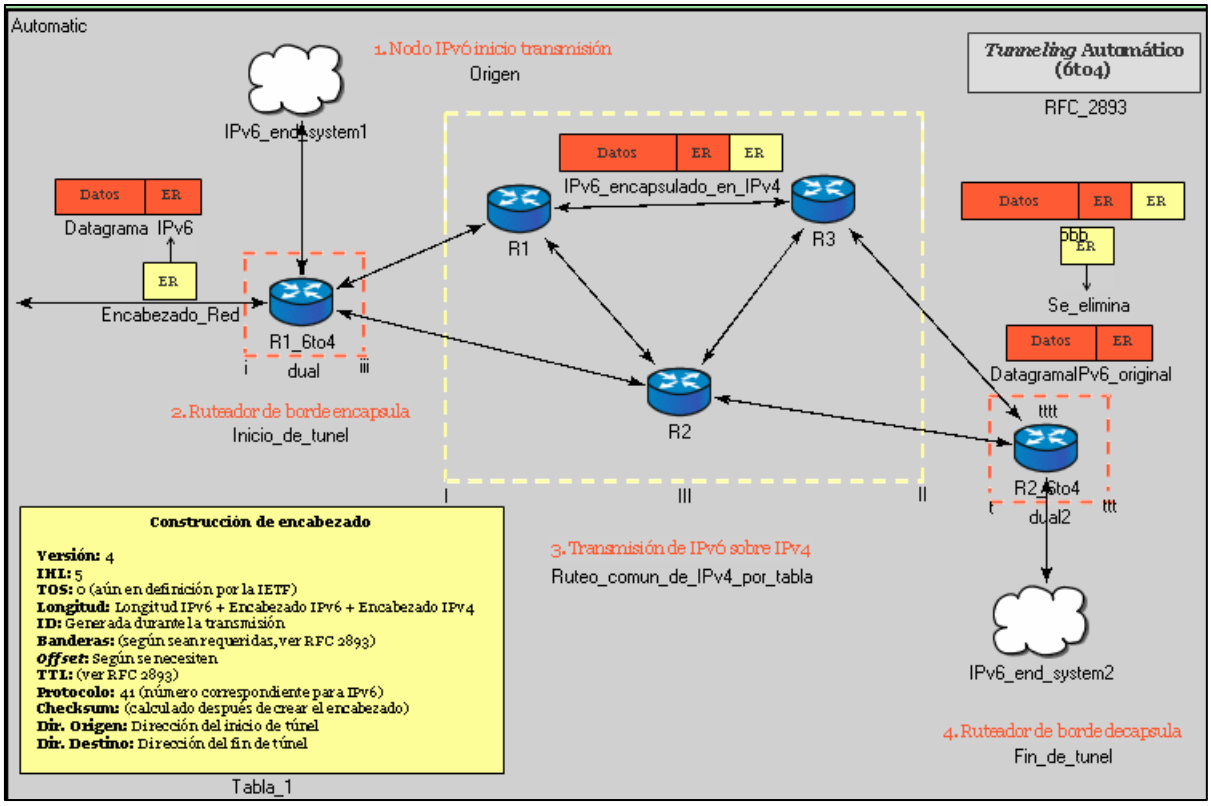

Figura 4.12: Técnicas de tunneling Automático.

Cualquiera de las técnicas anteriores, establecen "túneles" preestablecidos que son creados e introducidos manualmente dentro de las tablas que guardan los nodos que comienzan y terminan el "túnel". Para efectos de simulación, estas tablas se actualizadas manualmente y son creadas arbitrariamente, tienen el sólo propósito de demostrar el comportamiento de estas técnicas.

4.4 Resumen

De acuerdo a lo reportado en esta Capítulo, el primer paso que se realizó fue crear una interfaz gráfica que mostrara el comportamiento de cada uno de los modelos de transición a IPv6. Ésta fue realizada con ayuda de un editor gráfico de red conocido como GNED. Los bloques generados en este programa pueden utilizarse tantas veces sean necesarios por otros módulos creados. Esta herramienta gráfica de OMNeT++ permite mostrar la estructura interna de los dispositivos de una red de una forma

jerárquica, dándole una dimensión a la estructura creada y permitiendo así ver las capas que los conforman, lo que en cualquier otro simulador no se podría efectuar.

La topología utilizada para cada modelo es la misma y cuenta con el mismo número de dispositivos de ruteo, pero como es de notarse cada modelo presenta ligeras variaciones en función de su estructura general. Como el caso de Traducción que cuenta con un ruteador extra que hace la función de traductor de encabezados de red.

Así mismo, la estructura interna de estos dispositivos de red varía dependiendo del mecanismo modelado, por ejemplo el caso de Pila-dual que integra en su arquitectura el uso de dos pilas que dan el soporte a IPv4 e IPv6 simultáneamente. O por otro lado, el uso de una estructura diferente en el traductor, propuesto en el mecanismo de Traducción, que cuenta no sólo con dos pilas (similares a Pila-dual) para cada protocolo, sino que también incluye un módulo discriminador que se encarga de hacer la distinción entre protocolos.

Los modelos creados en esta sección tienen como finalidad, servir únicamente como medio gráfico durante la simulación y están basados en un RFC escrito específicamente para ellos. El comportamiento que éstos puedan tomar depende del código que se elabore para ellos en C++, este se incluye de manera completa en el disco adjunto.