UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

Programación de opciones IPv6 Hop by Hop en *NS3*

Alumno: Miguel García Caño

Tutor: Francisco Javier Sánchez-Roselly Navarro
Depto.: Ingeniería de Telecomunicación

Octubre, 2014

Grado en Ingeniería Telemática

Escuela Politécnica Superior de Linares

Índice General

1. MEMORIA

1.1 Índice

1.2 Introducción

El presente proyecto introduce cambios en el software de simulación de redes de código abierto *Network Simulator 3* (*NS3*), de forma que soporte en su implementación el uso de las opciones *Hop by Hop* de *IPv6*.

Implementa opciones que pueden ser de interés al futuro usuario del simulador y se demuestra el correcto funcionamiento de la implementación mediante la simulación de unos escenarios que ponen a prueba los requisitos de diseño de estas opciones.

1.3 Objeto

Los objetivos que se alcanzan con este proyecto son los siguientes:

- Revisión de la extensión *Hop by Hop* de *IPv6* y sus opciones en la implementación de *IPv6* de *NS3*.
- Modificación y añadido de características que completen la implementación de las opciones de *Hop by Hop* en el simulador *NS3*.
- Desarrollo de un método eficiente de procesamiento de opciones *Hop by Hop* mediante el uso de *Packet Tags*.
- Programación de aplicaciones que simulen escenarios en los que se valide el buen funcionamiento de *Hop by Hop*.

1.4 Referencias

[1] ALLMAN, M., FLOYD, S., JAIN, A. y SAROLAHTI, P. (1999). *Quick-Start for TCP and IP*. *RFC 4782*.
<http://tools.ietf.org/html/rfc4782> [Consulta: 11 de agosto de 2014]

[2] ATKINSON, R., ST. JOHNS, M. y THOMAS, G. (2009). *Common Architecture Label IPv6 Security Option (CALIPSO). RFC 5570.* <http://tools.ietf.org/html/rfc5570> [Consulta: 12 de marzo de 2014]

[3] BORMAN, D., DEERING. S y HINDEN,R. (1999). *IPv6 Jumbograms. RFC 2675.* <http://tools.ietf.org/html/rfc2675> [Consulta: 1 de diciembre de 2013]

[4] CONTA, A. y DEERING, S. (1998). *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 2463.* <http://tools.ietf.org/html/rfc2463> [Consulta: 1 de diciembre de 2013]

[5] DEERING, S. y HINDEN, R. (1998). *Internet Protocol, Version 6 (IPv6) Specification. RFC 2460.* <http://tools.ietf.org/html/rfc2460> [Consulta: 1 de diciembre de 2013]

[6] JACKSON, A. y PARTRIDGE, C (1999). *IPv6 Router Alert Option. RFC 2711.* < http://tools.ietf.org/html/rfc2711> [Consulta: 1 de diciembre de 2013]

[7] *NS3*-PROJECT (2013). *Documentación Doxygen de la API de Network Simulator 3 versión 3.18.* <http://www.nsnam.org/docs/release/3.18/doxygen> [Consulta: 1 de noviembre de 2013]

[8] *NS3*-PROJECT (2013). *Manual de Network Simulator 3 versión 3.18.* <http://www.nsnam.org/docs/release/3.18/manual/ns-3-manual.pdf> [Consulta: 15 de octubre de 2013]

[9] *NS3*-PROJECT (2013). *Guía de instalación de Network Simulator 3.* <http://www.nsnam.org/wiki/Installation> [Consulta: 15 de octubre de 2013]

1.5 Definiciones y abreviaturas

- ***CSMA***: ***C****arrier* ***S****ense* ***M****ultiple* ***A****ccess.* Protocolo de acceso al medio en el que un nodo, antes de empezar a transmitir datos, comprueba si el medio de transmisión que quiere utilizar está siendo utilizado por otro nodo con el fin de evitar colisiones.

- ***GNU***: ***GNU** is **N**ot **U**nix*. Sistema operativo de código abierto dirigido por la *Free Software Foundation*.

- **GPL**: *GNU **G**eneral **P**ublic **L**icense*. Licencia de software redactada por la *Free Software Foundation*.

- ***ICMP***: ***I**nternet **C**ontrol **M**essage **P**rotocol*. Protocolo de nivel de red utilizado para controlar el correcto funcionamiento de la red a través del envío de mensajes.

- **ICMPv6**: ***I**nternet **C**ontrol **M**essage **P**rotocol **v**ersion* **6**. Implementación del protocolo *ICMP* para *IPv6*.

- ***IP***: ***I**nternet **P**rotocol*. Protocolo de transmisión de datos no orientado a conexión a través de una red conmutada.

- ***IPv6***: ***I**nternet **P**rotocol **v**ersion* **6**. Versión 6 del protocolo *IP*.

- ***Jumbogram***: Paquete *IPv6* que contiene un campo *payload* de 32 bits de longitud. Este término puede también hacer referencia a la opción de *IPv6 Jumbogram*, tratada en este proyecto.

- ***LOG***: Registro en el que un programa guarda los eventos que se van sucediendo durante su ejecución.

- ***MAC***: ***M**ultiple **A**ccess **C**ontrol*. Control de acceso que se produce en una red que aplique reglas de control de tráfico *Calipso*.

- ***MAC (dirección MAC)***: Dirección física de un elemento de la red.

- ***MLD***: ***M**ulticast **L**istener **D**iscovery*. Protocolo utilizado por un router *IPv6* para descubrir la presencia de otros nodos

- ***MTU***: ***M**aximum **T**ransfer **U**nit.* Tamaño máximo en bytes que una unidad de datos puede tener para ser transferida en un protocolo.

- ***NS3***: ***N**etwork **S**imulator **3***. Simulador de redes de datos de código abierto.

- ***Payload***: Carga útil de la cabecera de un paquete.

- **RAM**: ***R**andom **A**ccess **M**emory*.

- ***RFC***: ***R****equest* ***F****or* ***C****omments*. Publicaciones de la organización *Internet Engineering Task Force* que describen procedimientos, métodos o nuevas ideas relativas al desarrollo de *Internet*.

- ***TCP***: ***T****ransmission* ***C****ontrol* ***P****rotocol*. Protocolo de transporte que ofrece, entre otras características, fiabilidad en la transmisión de un datagrama.

- ***UDP***: ***U****ser* ***D****atagram* ***P****rotocol*. Protocolo de transporte de datagramas no fiable que busca la eficiencia de transmisión.

1.6 Antecedentes

Este apartado describe las características principales sobre *IPv6* y *NS3* que han sido estudiadas en detalle para la realización del proyecto y que son necesarias comprender para el seguimiento de este documento.

1.6.1 Internet Protocol Version 6

El protocolo *IPv6* supone la evolución del globalmente extendido *IPv4*. Esta versión 6 del protocolo, definida en la *RFC 2460* [5], introduce respecto a su predecesor los siguientes cambios, solucionando ciertas limitaciones de la versión *IPv4*:

- Se modifica el tamaño de las direcciones *IP* a 128 bits, teniendo en cuenta los 32 bits de los que se dispone en *IPv4*. Se soluciona así el problema del crecimiento exponencial de dispositivos conectados a Internet, disponiendo de un considerable mayor número de direcciones utilizables; y facilita la auto-configuración de direcciones y rutas en los dispositivos.

- Se simplifica el formato de cabecera del protocolo, ya que los paquetes de *IPv4* incluían en su cabecera campos ya inutilizados u opcionales que añadían un coste de procesamiento del paquete de forma innecesaria.

- Mayor flexibilidad en el uso de opciones y la introducción de las cabeceras de extensión, las cuales son objeto de estudio en este proyecto. El uso de éstas permite

una mayor eficiencia en el procesamiento de los paquetes y facilita la creación de nuevas opciones en el futuro.

- Se facilita la identificación de flujos de tráfico mediante el uso de etiquetas.

Aunque en la práctica *IPv4* sigue siendo el protocolo ampliamente utilizado en Internet a nivel de red, irá dejando paso a la versión 6; y todo equipamiento, tanto software como hardware, deberá estar preparado con implementaciones funcionales de *IPv6* para el progresivo salto a este protocolo.

Puesto que en este proyecto ha tratado principalmente con el protocolo *IPv6* y paquetes que contienen las cabeceras de este protocolo, en los siguientes apartados se describen los formatos de cabecera utilizados durante su desarrollo.

1.6.1.1 Formato de cabecera de IPv6

Un paquete de *IPv6* [4] tiene la estructura de la siguiente figura:

Figura 1.6.1 – Estructura de un paquete *IPv6*

Esta estructura contiene los siguientes elementos:

- La cabecera básica de *IPv6*, la cual deberá llevar todo paquete de *IPv6*.
- Tantas cabeceras de extensión como sean necesarias. El paquete puede no llevar cabeceras de extensión si no son requeridas.
- Los datos de los protocolos superiores a *IP*, tales como *TCP* o *UDP*.

La cabecera básica de *IPv6* tiene un tamaño fijo de 40 bytes distribuidos de la siguiente forma:

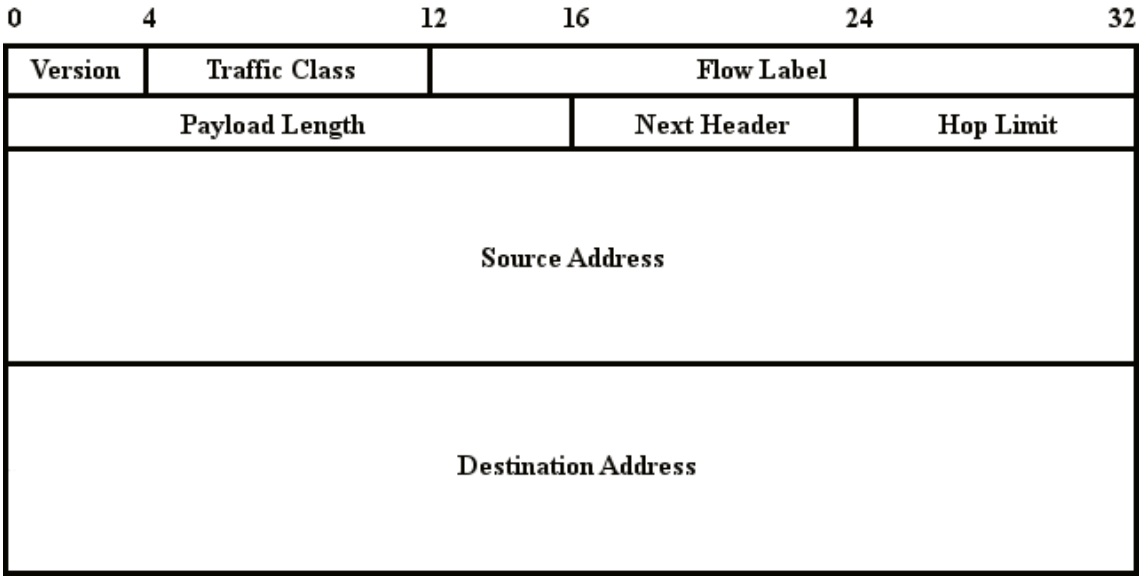

Figura 1.6.2 – Formato de cabecera básica de *IPv6*

- ***Version* (4 bits):** Identifica la versión de IP utilizada. Su valor es 6.
- ***Traffic Class* (8 bits):** Indica la clase o prioridad del paquete.
- ***Flow Label* (20 bits):** Este campo es utilizado para identificar diferentes flujos de tráfico según el valor que contenga, permitiendo que puedan ser tratados de distinta manera por el nodo que trate con el paquete.
- ***Payload Length* (16 bits):** Indica el tamaño, medido en bytes, de la carga o *payload* del paquete *IPv6*. Este *payload* incluye las cabeceras de extensión del paquete y los datos de las capas superiores.
- ***Next Header* (8 bits):** Indica el tipo de cabecera que procede a esta cabecera básica de *IPv6*, pudiendo ser una cabecera de extensión o el protocolo de la capa superior.
- ***Hop Limit* (8 bits):** Indica el número de nodos por el que el paquete puede pasar. Este número es disminuido una unidad por cada paso por un nodo, y es descartado en el momento en el que su valor es 0.
- ***Source Address* (32 bits):** Indica la dirección *IP* del nodo que origina el paquete.
- ***Destination Address* (32 bits):** Indica la dirección *IP* del nodo destinatario del paquete.

Las cabeceras de extensión son aquellas utilizadas en *IPv6* con el propósito de incluir información adicional en el paquete en la capa de *IP*. Estas cabeceras de extensión se sitúan entre la cabecera básica de *IPv6* y las cabeceras de los protocolos superiores a *IP*; y contienen, al igual que la cabecera básica, un campo *Next Header* que identifica la siguiente cabecera de extensión, en el caso de que la haya, o el protocolo de nivel superior a *IP*.

La figura 1.6.3 muestra un ejemplo de la estructura de un paquete *UDP* con dos cabeceras de extensión en la capa *IP*:

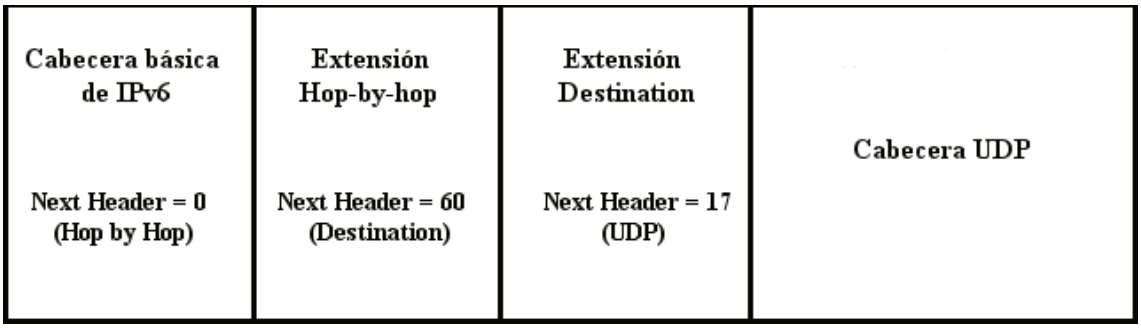

Figura 1.6.3 – Ejemplo de paquete con cabeceras de extensión

1.6.1.1 Extensión Hop by Hop

Este proyecto ha trabajado sobre una de las cabeceras de extensión existentes en *IPv6*: la extensión *Hop by Hop*, definida en la *RFC 2460* [5]. Esta cabecera añade al paquete información que será examinada en cada uno de los nodos del trayecto del paquete y no sólo por el nodo destino.

Por tanto, toda implementación de *IPv6* debe estar preparada para comprobar la presencia de una extensión *Hop by Hop* al momento de recibir un paquete, independientemente de que sea el nodo destinatario del paquete o un nodo intermedio.

Esta cabecera tiene el propósito de evitar tener que procesar un paquete completo en busca de información que deba ser procesada por todos los nodos intermedios.

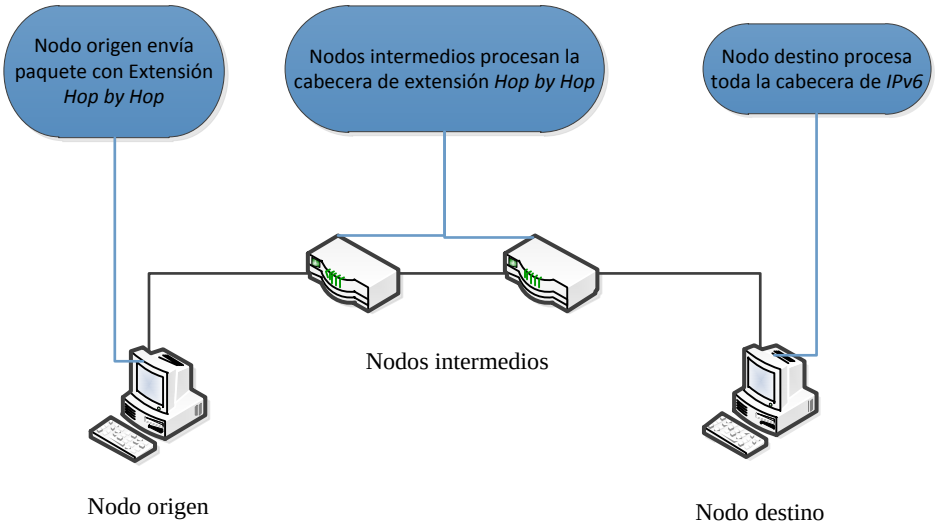

Figura 1.6.4 – Procesamiento de la extensión *Hop by Hop*

Por procesamiento de la cabecera se entiende el reconocimiento por parte del nodo de ésta, comprobando la ausencia de errores de formato en ella, y la realización de una serie de acciones relacionadas con dicha cabecera que tengan que ver con su funcionalidad específica.

En el caso de la extensión *Hop by Hop*, el procesamiento consiste en el examen de la cabecera en busca de opciones *Hop by Hop* de *IPv6*, las cuales serán examinadas y procesadas individualmente.

El formato de la extensión *Hop by Hop* es el indicado por la figura 1.6.5:

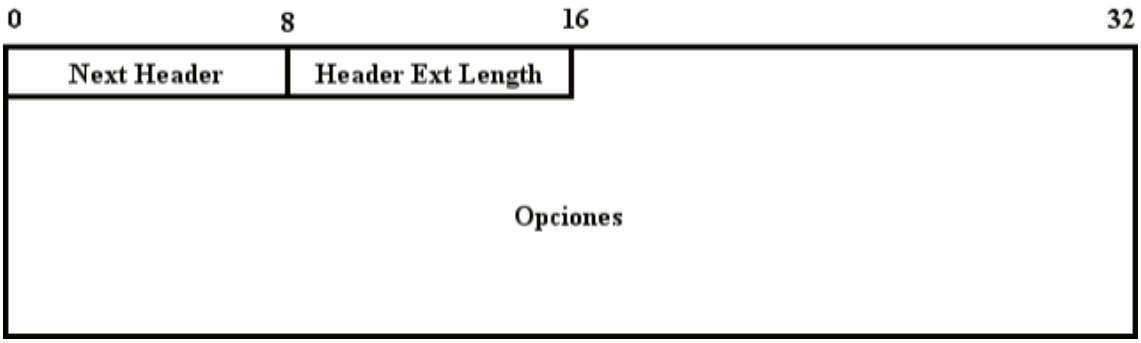

Figura 1.6.5 – Formato de cabecera de extensión *Hop by Hop*

La extensión *Hop by Hop* está compuesta por dos campos de tamaño fijo y de una o varias opciones de *IPv6*. Estos campos son:

- ***Next Header* (8 bits):** Al igual que el campo del mismo nombre de la cabecera básica de *IPv6*, indica el tipo de cabecera de extensión o cabecera del protocolo de nivel superior a *IP* que procede a la cabecera de extensión *Hop by Hop*.

- ***Header Extension Length* (8 bits):** Indica el tamaño, medido en bytes, de la cabecera de extensión *Hop by Hop*, excluyendo en esta cuenta los 8 primeros bytes.

Tras estos dos campos, se suceden todas las opciones de *IPv6* que hayan sido añadidas a la cabecera, las cuales tendrán un tamaño que, sumado a los dos campos anteriores, harán que el tamaño total de la cabecera *Hop by Hop* sea un múltiplo de 8 bytes.

1.6.1.2 Opciones Hop by Hop de IPv6

Las opciones *Hop by Hop* contienen información que puede ser de interés a los nodos intermedios durante el trayecto de un paquete *IPv6*. Cada opción tiene definida su propia cabecera y se sitúan posteriormente a los dos campos principales de la cabecera de la extensión *Hop by Hop*. En el caso de que haya más de una opción, se sitúan una detrás de otra y son procesadas por cada nodo receptor en el orden en el que se encuentran.

El tamaño y el formato de la cabecera de estas opciones, definidos en la *RFC 2460* [5], pueden variar de una a otra, pero todas comparten la estructura de la figura 1.6.6:

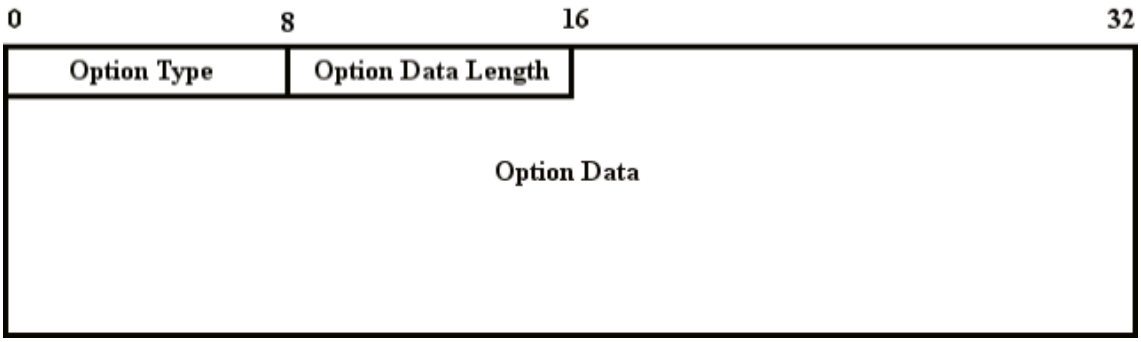

Figura 1.6.6 – Formato de cabecera de una opción *Hop by Hop*

- ***Option Type* (8 bits):** Indica el número identificador del tipo de opción.

- ***Option Data Length* (8 bits)**: Indica la longitud, medida en bytes, del siguiente campo *Option Data*.

- ***Option Data* (tamaño variable):** Contiene los campos específicos de la opción.

Las opciones pueden tener requisitos específicos de alineamiento dentro de la cabecera; es decir, el comienzo de la cabecera de la opción puede tener como requisito estar desplazado a una cierta distancia medida en bytes desde el inicio de la extensión *Hop by Hop*.

Estos requisitos se especifican mediante la notación *xn+y*; queriendo decir esto que el primer bit del campo *Option Type* debe estar desplazado, respecto al inicio de la cabecera de extensión *Hop by Hop*, un múltiplo de *x* bytes más *y* bytes.

Tanto para cumplir los requisitos de tamaño de la cabecera *Hop by Hop*, la cual debe tener un tamaño total múltiplo de 8 bytes; como para cumplir los requisitos de alineamiento individuales de las opciones, la *RFC 2460* define dos opciones de relleno de bits que toda implementación de *IPv6* debe ser capaz de interpretar: *Pad1* y *PadN*.

Pad1 es la opción utilizada para añadir sólo 1 byte de relleno a la cabecera de extensión *Hop by Hop*. Esta opción, explicada en la *RFC 2460* [5], tiene la particularidad de no compartir la misma estructura que el resto de opciones, ya que no contiene el campo *Option Data Length* ni ningún tipo de campo adicional. El formato de cabecera es, por tanto, un byte con valor 0, como se muestra en la figura 1.6.7:

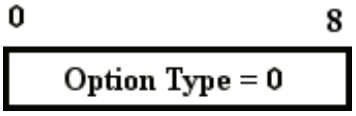

Figura 1.6.7 – Formato de cabecera de la opción *Pad1*

PadN, definida también en la *RFC 2460* [5], es la opción utilizada cuando es necesario rellenar dos o más bytes en la cabecera de extensión *Hop by Hop*. El tamaño de la cabecera dependerá del número de bytes que necesite rellenar. El formato de cabecera se muestra en la figura 1.6.8:

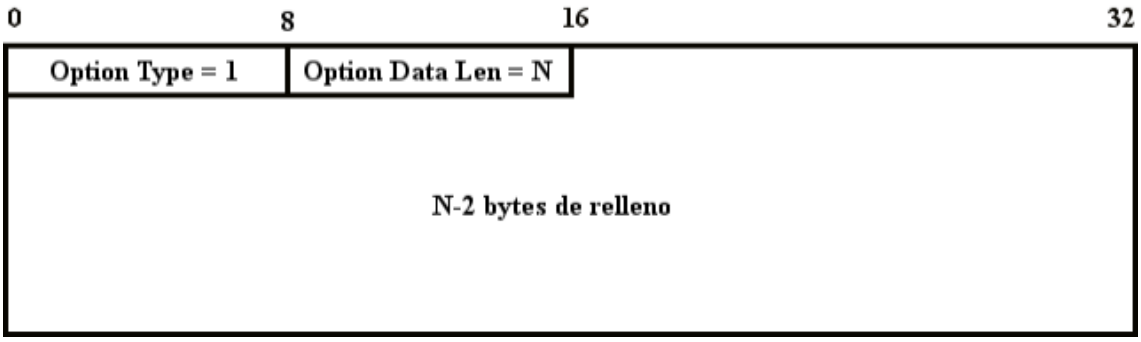

Figura 1.6.8 – Formato de cabecera de la opción *PadN*

A continuación se muestra un ejemplo de requisitos de alineamiento de las opciones dentro de la cabecera de extensión *Hop by Hop*. Se cuenta con un paquete que tiene las siguientes características:

- El protocolo superior a *IPv6* es *UDP*.
- Contiene una cabecera de extensión *Hop by Hop*.
- La cabecera *Hop by Hop* debe llevar la opción *Router Alert*, la cual tiene un tamaño de 4 bytes y unos requisitos de alineamiento de *2n+0*.

La figura 1.6.9 muestra el formato de cabecera *Hop by Hop* resultante para este ejemplo:

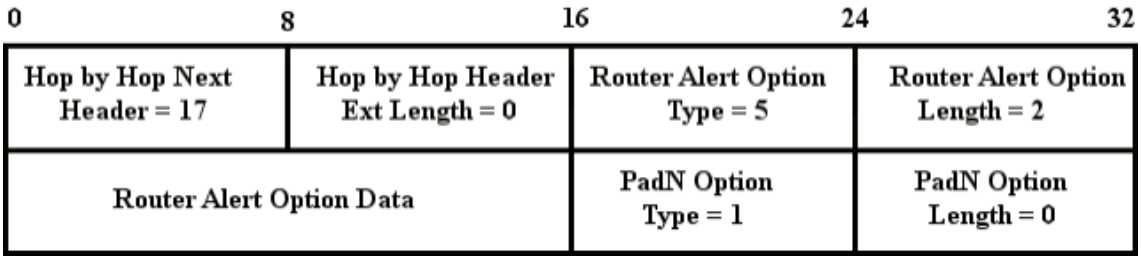

Figura 1.6.9 – Ejemplo de alineamiento de opciones

En la figura 1.6.9 se observa lo siguiente:

- El tamaño total de la cabecera *Hop by Hop* es de 8 bytes, cumpliendo así el requisito de que la cabecera debe tener un tamaño múltiplo de 8 bytes.

- La opción *Router Alert* tiene el requisito de alineamiento de *2n+0*, el cual se cumple al observar que el campo *Option Type* de la opción *Router Alert* está situado a dos bytes del inicio de la cabecera.

- La opción *PadN* ayuda a la cabecera de extensión *Hop by Hop* a tener un tamaño de 8 bytes, rellenando los últimos dos bytes de ésta.

Router Alert

Una de las opciones de *IPv6* tratadas en este proyecto, ya mencionada anteriormente, es la denominada *Router Alert*, definida en la *RFC 2711* [6]. Esta opción tiene el propósito de avisar al nodo receptor de este paquete de que el datagrama recibido debe ser examinado detenidamente. Uno de sus campos determina el motivo por el que el datagrama debe ser procesado por el nodo en cuestión.

La opción tiene un tamaño de 4 bytes y tiene un requisito de alineamiento de *2n+0*. El formato de cabecera se muestra en la figura 1.6.10:

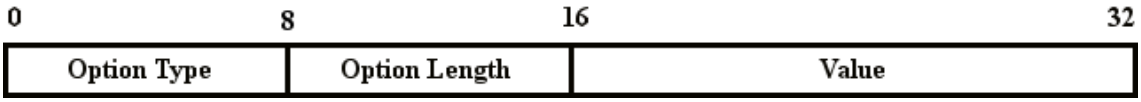

Figura 1.6.10 – Formato de cabecera de la opción *Router Alert*

- ***Option Type* (8 bits):** Indica el número identificador de la opción *Router Alert*. Su valor decimal es 5.

- ***Option Length* (8 bits):** Indica el tamaño, medido en bytes, de la opción *Router Alert*. Su valor decimal es 4.

- ***Value* (16 bits):** Indica el motivo por el que el paquete debe ser examinado detenidamente por el nodo que lo está procesando.

Jumbogram

La siguiente opción, de nombre *Jumbogram* y detallada en la *RFC 2675* [6], es utilizada cuando la carga o *payload* del paquete *IPv6* tiene un tamaño de entre 65,535 y 4,294,967,295 bytes; siendo en este caso el campo *Payload Length* de la cabecera de *IPv6* insuficiente.

La opción tiene un tamaño de 6 bytes y tiene un requisito de alineamiento de *4n+2*. El formato de cabecera se muestra en la figura 1.6.11:

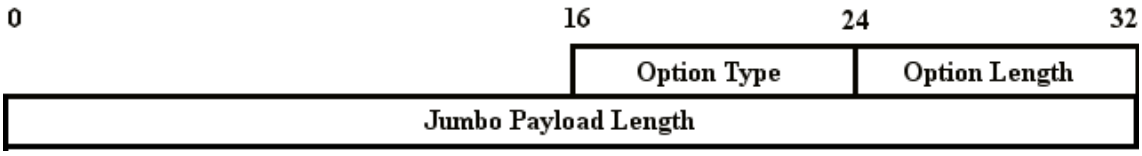

Figura 1.6.11 – Formato de cabecera de la opción *Jumbogram*

- ***Option Type* (8 bits):** Indica el número identificador de la opción *Jumbogram*. Su valor decimal es 194.
- ***Option Length* (8 bits):** Indica el tamaño, medido en bytes, de la opción *Jumbogram*. Su valor decimal es 6.
- ***Jumbo Payload Length* (16 bits):** Indica el tamaño del *payload* del paquete *IPv6*. Al estar presente esta opción en la cabecera, el campo *Payload Length* de la cabecera de *IPv6* debe ser 0.

A la hora de procesar un paquete que contiene una opción *Jumbogram*, un nodo receptor capaz de interpretar esta opción debe comprobar ciertos errores de formato que pueden darse a la hora de recibirlo. En caso de hallar uno de ellos, se descartará el envío del paquete y un mensaje de *ICMPv6* de tipo *Parameter Problem* debe ser enviado al nodo origen, con código 0 y puntero a donde se indique en cada caso, desde el nodo donde el paquete ha sido procesado.

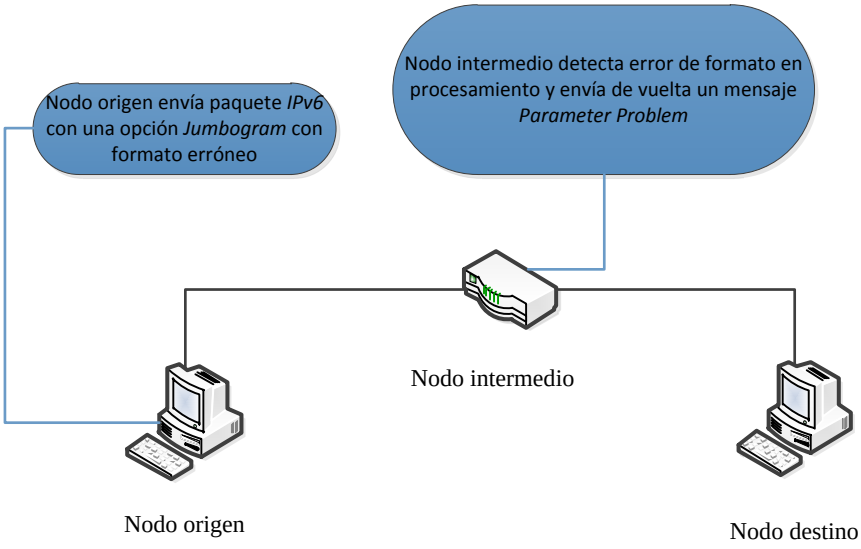

Figura 1.6.12 – Procesamiento de opción *Jumbogram* con formato erróneo

Según la *RFC 2675* [6], pueden distinguirse cuatro errores de formato de cabecera para la opción *Jumbogram*:

Jumbogram caso 1

Tipo de error: *Payload* de *IPv6* igual a 0, *IPv6 Next Header = Hop by Hop* y Opción *Jumbo Payload* no presente; donde todo paquete con *payload* de valor 0 en la cabecera de *IPv6* debe llevar obligatoriamente una opción de tipo *Jumbo Payload*.

El mensaje *ICMP* tendrá un puntero al octeto de más alto orden del *payload* de la cabecera de *IPv6*.

Jumbogram caso 2

Tipo de error: *Payload* de *IPv6* distinto de 0 y Opción *Jumbo Payload* presente; debido a que el *payload* válido del paquete debe ser indicado por la opción de *Jumbogram* y no por la cabecera de *IPv6*.

El mensaje *ICMP* apuntará al campo *Option Type* de la opción *Jumbo Payload*.

Jumbogram caso 3

Tipo de error: Opción *Jumbo Payload* presente y un valor del campo *Jumbo Payload Length* menor que 65,535; lo cual no tendría sentido y haría inútil la presencia de la opción en el paquete.

El mensaje *ICMP* apuntará al campo *Option Type* de la opción Jumbo *Payload*.

Jumbogram caso 4

Tipo de error: Opción *Jumbo Payload* presente junto a una cabecera de extensión de tipo *Fragment*; ya que un paquete que incluya esta opción no puede ser fragmentado.

El mensaje *ICMP* apuntará al octeto de más alto orden de la cabecera *Fragment*.

Calipso

La opción *Calipso*, definida en la *RFC 5570* [2], está diseñada para etiquetar los paquetes transmitidos en una red y aplicar en los nodos de esta red ciertas políticas de control de acceso al tráfico según la clasificación dada a los paquetes.

La opción tiene un tamaño variable entre 18 y 26 bytes; y tiene un requisito de alineamiento de *4n+2*. El formato de cabecera se muestra en la figura 1.6.13:

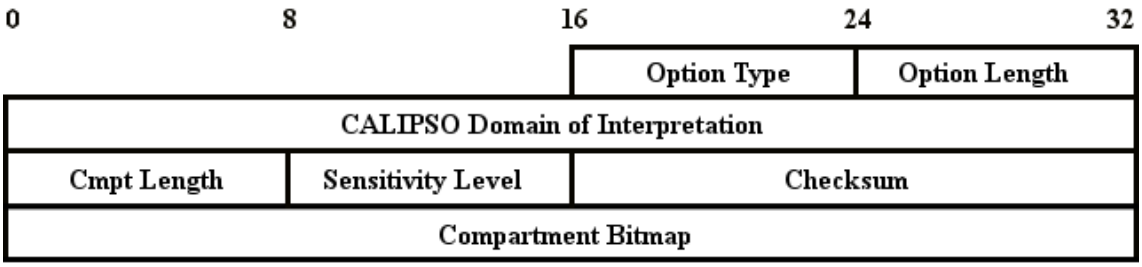

Figura 1.6.13 – Formato de cabecera de la opción *Calipso*

- ***Option Type* (8 bits):** Indica el número identificador de la opción *Calipso*. Su valor decimal es 7.

- ***Option Length* (8 bits):** Indica el tamaño, medido en bytes, de la opción *Calipso*, excluyendo los campos *Option Type* y *Option Length*.

- ***CALIPSO Domain of Interpretation* (32 bits):** Este campo representa el llamado *Dominio de Interpretación* de *Calipso* (*DOI*, de ahora en adelante) e indica las reglas bajo las que un paquete con la opción *Calipso* será tratado.

- ***Compartment Length* (8 bits):** Indica el tamaño del campo *Compartment Bitmap* que aparece posteriormente.

- ***Sensitivity Level* (8 bits):** El valor de este campo indica la sensibilidad relativa de la información que el paquete contiene en el contexto del *DOI* especificado en la cabecera *Calipso*. El valor de este campo no tiene ningún significado en concreto definido.

- ***Checksum* (16 bits)**: El *checksum* de la cabecera se genera alrededor de toda la opción, exceptuando los 16 bits reservados para este *checksum*, los cuales se dejarán a 0 a la hora de calcularlo.

- ***Compartment Bitmap* (de 0 a 64 bits)**: Éste amplía el tamaño del *DOI*, dividiéndolo en tantos compartimentos como bits mida este campo. Por cada bit que mida este campo, se dispone de 256 niveles de sensibilidad adicionales.

Un nodo capaz de procesar la opción *Calipso* debe poder almacenar en cada una de sus interfaces cierta información referente a esta opción. Éstas almacenarán:

- Un conjunto de *DOIs* definidos.

- Un listado de *DOIs* habilitados en la interfaz. Una interfaz puede tener varios *DOIs* definidos en su base de datos, pero no todos tienen por qué estar permitidos para el paso de tráfico por dicha interfaz.

- Cada *DOI* debe tener un rango de valores de niveles de sensibilidad y de *Compartment Bitmap* permitidos.

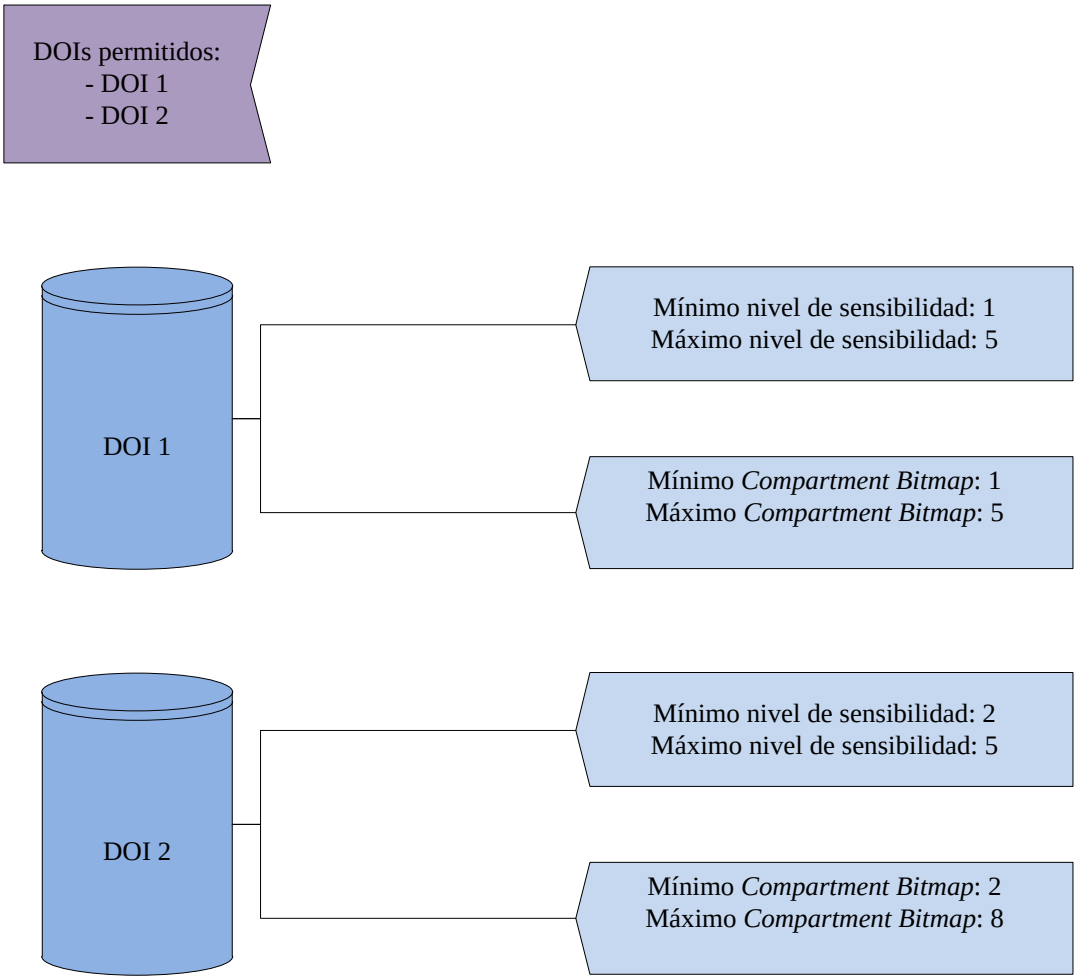

Figura 1.6.14 – Configuración *Calipso* de una sola interfaz

Cuando un paquete es procesado por una interfaz de red configurada para funcionar con *Calipso*, éste es sometido a un *MAC* (*Mandatory Access Control)* o *Control de Acceso Obligatorio*, el cual debe cumplir las siguientes condiciones para que el paquete no sea descartado y pueda continuar el camino hacia el nodo destino:

- El *checksum* contenido en el paquete debe haber sido generado correctamente.
- El valor del campo *DOI* debe ser uno de los *DOIs* reconocidos y permitidos por la interfaz.
- Los valores de los campos *Nivel de sensibilidad* y *Compartment Bitmap* deben estar entre los límites configurados en la interfaz de red para el *DOI* contenido en el paquete en concreto.

Quick Start

La opción *Quick Start* viene definida en la *RFC 4782* [1] y es un mecanismo que puede ser utilizado por protocolos de transporte y diseñado con el propósito de establecer la tasa de transmisión de datos al inicio o punto medio del proceso de transmisión.

Quick Start puede ser utilizado como opción *Hop by Hop* en *IPv6*, aunque no es exclusiva de este protocolo, y debe usarse en conjunto con un protocolo de transporte como *TCP*. Desgraciadamente, la implementación de *TCP* de *NS3* en *IPv6* no está completa y no se ha podido integrar la opción *Quick Start* en este protocolo. En cambio, sí se ha trabajado con *Quick Start* en cuanto a la creación y procesamiento de su cabecera y su integración con la extensión *Hop by Hop* en *NS3*.

La opción tiene un tamaño variable de 8 bytes y no tiene un requisito de alineamiento definido. El formato de cabecera se muestra en la figura 1.6.15:

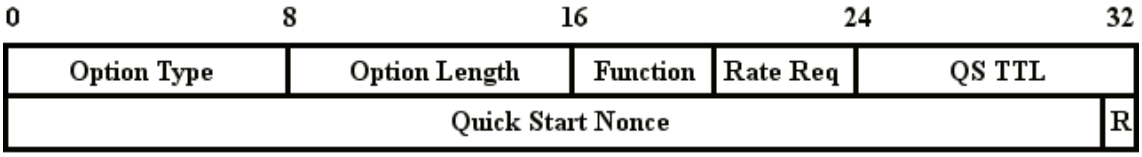

Figura 1.6.15 – Formato de cabecera de la opción *Calipso*

- ***Option Type* (8 bits):** Indica el número identificador de la opción *Quick Start*. Su valor decimal es 38.
- ***Option Length* (8 bits):** Indica el tamaño, medido en bytes, de la opción *Quick Start*, excluyendo los campos *Option Type* y *Option Length*. Su valor decimal es 6.
- ***Function (*4 bits):** Indica si la opción *Quick Start* es un *Rate Request* o un *Report of Approved Rate*. Es un *Rate Request* si el valor de este campo es 0. Será un *Report of Approved Rate* si el valor decimal es 8.
- ***Rate Request* (4 bits):** Indica la tasa de transmisión con la que se desea transmitir los datos a través de la red. Este campo no es utilizado si se trata de un *Report of Approved Rate*.

- ***QS TTL* (4 bits):** Este campo es reducido en una unidad a cada paso por un router y es utilizado por el nodo origen para comprobar si todos los nodos que han recibido este paquete han entendido y aprobado la opción *Quick Start*.

- ***QS Nonce* (30 bits):** Este campo es utilizado como mecanismo de seguridad ante posibles routers mintiendo sobre los valores recibidos en un *Rate Request*.

- ***R* (2 bits):** Dos bits de reserva que el nodo origen debe ignorar y dejar con valor 0.

La siguiente figura resume el funcionamiento de *Quick Start* utilizado en *TCP* para negociar el tamaño inicial de la ventana de congestión de este protocolo:

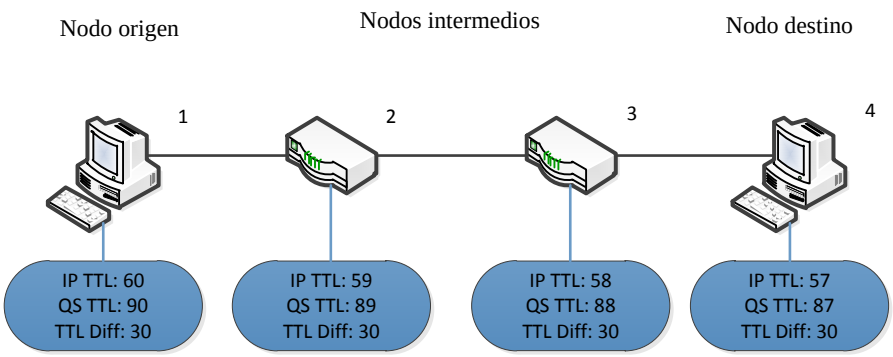

Figura 1.6.16 – Envío de una opción *Quick Start*

Los pasos marcados numéricamente en la figura 1.6.16 son explicados en las siguientes líneas:

1. El nodo origen envía el paquete *SYN* o *SYN/ACK* necesario para establecer una sesión de transporte con el protocolo *TCP* entre los nodos origen y destino. Este paquete contiene una cabecera de extensión *Hop by Hop* y una opción *Quick Start* en su interior con los parámetros de configuración que establecerán el tamaño inicial de la ventana de congestión. Este nodo origen calcula la diferencia entre el campo *TTL* de la cabecera de *IPv6* y el campo *QS TTL* de la opción *Quick Start*, *IP TTL – QS TTL*, y lo almacena para su posterior uso.

2. El primer nodo intermedio recibe el paquete y disminuye en una unidad el campo *TTL* de la cabecera de *IPv6* y el campo *QS TTL* de la opción *Quick Start* como prueba de que aprueba los parámetros indicados por la opción. Si no los aprobara, no disminuiría este último campo.

3. El segundo nodo intermedio recibe el paquete y disminuye en una unidad los mismos campos.

4. El nodo destino recibe el paquete, disminuye ambos campos de nuevo como prueba de que aprueba las condiciones impuestas por la opción *Quick Start* y envía de vuelta al origen ambos parámetros. El nodo origen calcula la diferencia entre ellos, *IP TTL – QS TTL*, y, si es la misma que la calculada en el punto 1, puede confirmar que todos los nodos aprueban la opción y se establece el tamaño de la ventana de congestión deseado.

1.6.1.3 Protocolo ICMPv6

En este proyecto se ha trabajado con algunos mensajes del protocolo *Internet Control Message Protocol version 6* (*ICMPv6*, de ahora en adelante), por lo que será brevemente descrito en este apartado.

ICMPv6, definido en la *RFC 2463* [4], parte como la evolución del protocolo *ICMP* utilizado en *IPv4*, esta vez adaptado para su uso con *IPv6*, del cual depende para su funcionamiento íntegro y todo nodo *IPv6* debe implementarlo. Es utilizado para informar a los nodos de la red de posibles errores producidos en el procesamiento de los paquetes y para otras funciones relacionadas con la capa de *IP*, como puede ser el diagnóstico de la red a través de mensajes de control.

ICMPv6 Echo Request

Un mensaje de control *ICMPv6 Echo Request* es enviado a un nodo destino con la intención de que éste responda con otro mensaje de control al nodo origen, confirmando la conectividad entre ambos nodos.

En este proyecto se ha utilizado este mensaje en la mayoría de los escenarios programados para comprobar el correcto funcionamiento de las opciones de *IPv6* programadas, por lo que es conveniente introducir su cabecera en las siguientes líneas.

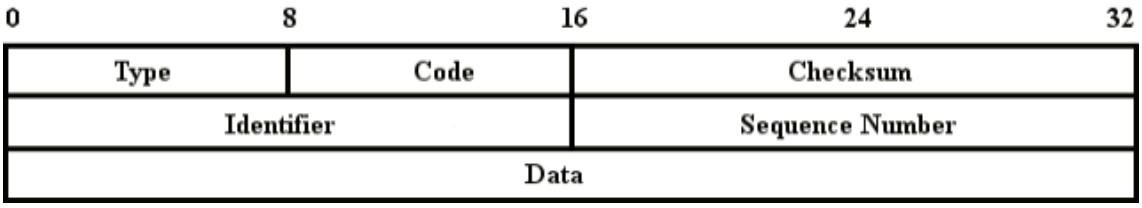

Figura 1.6.17 – Formato de cabecera de un mensaje *ICMPv6 Echo Request*

- ***Type* (8 bits):** Indica el tipo de mensaje *ICMPv6* contenido en la cabecera. Para este caso su valor es 128.
- ***Code* (8 bits):** Dependiendo del tipo de mensaje indicado por el campo *Type*, *Code* puede indicar un subnivel más de identificación. En este caso su valor es 0.
- ***Checksum* (16 bits):** *Checksum* de 16 bits de la cabecera de *ICMPv6*, el cual debe ser calculado según las instrucciones dadas en la *RFC 2463*. No es relevante durante el desarrollo del proyecto.
- ***Identifier (*16 bits):** Identificador del mensaje utilizado para distinguirlo de otros posibles mensajes *ICMPv6 Echo Request*.
- ***Sequence Number* (16 bits):** Número de secuencia utilizado para distinguir este mensaje de otros *ICMPv6 Echo Request*.
- ***Data*:** Cero o más bytes de datos aleatorios.

ICMPv6 Parameter Problem

En el apartado 1.6.1.2 de la memoria se habló de la opción de *IPv6 Jumbogram* y de los errores de formato de cabecera que los nodos debían comprobar en recepción. Cuando uno de los nodos encontraba uno de estos errores de formato, debía, además de descartar el paquete, enviar al nodo de origen del paquete un paquete de tipo *ICMPv6 Parameter Problem* indicando el motivo del problema encontrado.

Este mensaje *Parameter Problem* forma parte del protocolo *ICMPv6* y sirve para informar de este tipo de sucesos. La cabecera de este mensaje se detalla en la figura 1.6.18:

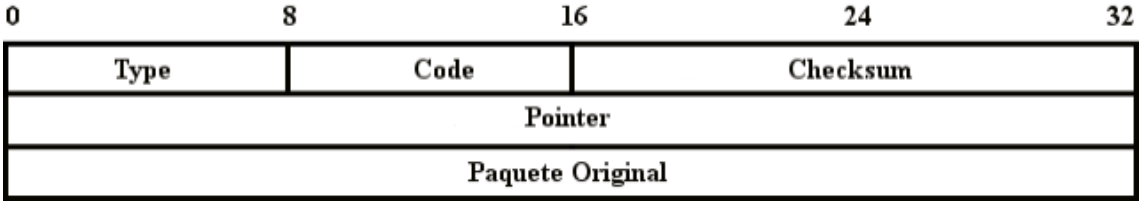

Figura 1.6.18 – Formato de cabecera de un mensaje *ICMPv6 Parameter Problem*

- ***Type* (8 bits):** Indica el tipo de mensaje *ICMPv6* contenido en la cabecera. Su valor es, en este caso, 4.

- ***Code* (8 bits):** Indica el tipo de error de cabecera encontrado. Su valor puede ser 0, 1, o 2, dependiendo de si el error consiste, respectivamente, en un campo erróneo de la cabecera, un valor para el campo *Next header* no reconocido o una opción de *IPv6* no reconocida.

- ***Checksum* (16 bits):** *Checksum* de 16 bits de la cabecera de *ICMPv6*, el cual debe ser calculado según las instrucciones dadas en la *RFC 2463*.

- ***Pointer* (32 bits):** Este puntero indica el byte del paquete original donde el error fue encontrado.

- ***Paquete original*:** El mensaje *ICMPv6 Parameter Problem* transporta en último lugar una copia del paquete original que fue enviado a este nodo receptor y donde se ha encontrado el error de cabecera. El valor del campo *Pointer* apunta a uno de los bytes del paquete original transportado de vuelta en la cabecera.

1.6.2 Network Simulator 3

El software *Network Simulator 3* (*NS3*) es el simulador de redes sobre el que en este proyecto se ha trabajado para añadirle nuevas funcionalidades. Su código es abierto, licenciado bajo la *GNU General Public License version 2,* y es desarrollado con el objetivo principal de ser útil en investigaciones y con fines educativos. La versión de *NS3* utilizada en este proyecto es la versión de desarrollo en la revisión 10226 de su rama de actualizaciones. Esta revisión es posterior a la versión 3.18 de *NS3* y anterior a la salida de la versión 3.19.

Con *NS3*, el usuario puede definir una topología de red de diversos tipos, tales como *Ethernet*, *Wifi* u otros modelos utilizados en la vida real, con la posibilidad de configurar todos los elementos de la red; y sobre esta topología se puede programar el intercambio de datagramas utilizando protocolos reales que han sido re-implementados en el simulador *NS3*, obteniendo unos resultados realistas y equiparables a los que se obtendrían utilizando equipos reales.

Una de las características principales de *NS3* es el hecho de que está programado con la intención de generar grandes cantidades de datos a partir de las simulaciones ejecutadas y sobre los que poder observar el comportamiento de la red. En este proyecto se ha aprovechado la capacidad de *NS3* para mostrar por consola los eventos que se producen durante la ejecución y la capacidad de generar capturas *pcap* analizables por programas como *TCPdump* o *Wireshark,* utilizados ambos para analizar tráfico generado por equipos reales. Las figuras 1.6.19 y 1.6.20 muestran ejemplos de ambas salidas:

```
Packet from 02-06-00:00:00:00:00:03 received on node 1
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x8453318, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x843c2b8, 0x8435318, [redacted] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 32
1:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Router Alert Option received and being processed on node 1
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x843c258, 0x8435318, [redacted] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 32
1:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Router PadN received and being processed on node 1
Ipv6L3Protocol:GetInterfaceForDevice(0x8453318, 0x843aa68)
Ipv6L3Protocol:GetInterfaceForDevice(0x8453318, 0x843aa68)
Ipv6L3Protocol:GetNInterfaces()
Ipv6L3Protocol:GetNAddresses(0x8453318, 0)
Ipv6L3Protocol:GetInterface(0x8453318, 0)
Ipv6L3Protocol:GetAddress(0x8453318, 0, 0)
```

Figura 1.6.19 – Sucesión de eventos en una ejecución de *NS3*

No.	Time	Source	Destination	Protocol	Length
1	0.000000	::	ff02::1	ICMPv6	90
2	0.002000	::	ff02::1	ICMPv6	90
3	0.006637	::	ff02::1	ICMPv6	90
4	0.009378	::	ff02::1	ICMPv6	90
5	1.001000	fe80::200:ff:fe00:1	ff02::2	ICMPv6	74
6	3.993000	fe80::200:ff:fe00:1	ff02::1:ff00:2	ICMPv6	90
7	3.997288	fe80::200:ff:fe00:2	fe80::200:ff:fe00:1	ICMPv6	90
8	3.997288	2001:1::200:ff:fe00:1	2001:2::200:ff:fe00:4	ICMPv6	90
9	8.997287	fe80::200:ff:fe00:2	fe80::200:ff:fe00:1	ICMPv6	90
10	8.997287	fe80::200:ff:fe00:1	fe80::200:ff:fe00:2	ICMPv6	90

```
Frame 8: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00
Internet Protocol Version 6, Src: 2001:1::200:ff:fe00:1 (2001:1::200:ff:fe00:1), Dst: 2001:2
Internet Control Message Protocol v6
  Type: Multicast Listener Report (131)
  Code: 0
  Checksum: 0x3bd4 [correct]
  Maximum Response Delay [ms]: 200
  Reserved: 0000
  Multicast Address: ff02::1:ff00:2 (ff02::1:ff00:2)
```

Figura 1.6.20 – Ejemplo de captura *pcap* generada por *NS3* y analizada con *Wireshark*

1.6.2.1 Orientación a objetos en NS3

El simulador *NS3* está fundamentalmente programado en el lenguaje *C++* y como tal está basado en el uso de objetos.

Es importante introducir ahora el concepto de objeto *NS3*. En *NS3*, los elementos de red se representan mediante objetos, los cuales parten todos de una base común. El simulador dispone de dos clases llamadas *ns3::Object* y *ns3:ObjectBase* de las cuales heredan estos elementos de red para crear a su vez nuevos objetos que representen estas entidades.

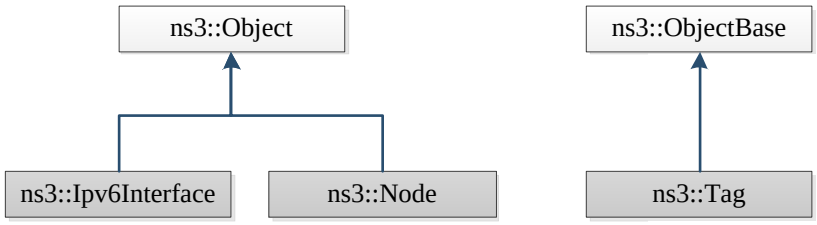

Figura 1.6.21 – Objetos en *NS3*

Así nacen los objetos de *NS3*, de entre los cuales existen objetos para definir elementos de red tales como los nodos, las interfaces de red, los paquetes y las cabeceras. Estos elementos son, pues, subclases de *ns3:Object* o de *ns3:ObjectBase*, según sea la clase de la que hereden.

En *NS3* se puede utilizar cualquier tipo de objeto, pero heredar de las anteriores clases permite obtener unas características especiales en el nuevo objeto creado que son de utilidad en el simulador:

- Se dispone de un conjunto de meta-datos ligados a un objeto, incluyendo la clase base (*NS3:Object* o *NS3:ObjectBase)*, los métodos de la subclase y los atributos de la subclase.
- Un *smart pointer* para cada objeto de *NS3*.
- La posibilidad de manipular los objetos eficientemente en memoria.

1.6.2.2 El paquete y sus cabeceras en NS3

En *NS3*, los paquetes y las cabeceras de *IPv6* son manipulados mediante el uso de objetos de *NS3* llamados *Packet* e *IPv6Header*, respectivamente.

Utilizando el objeto *Packet*, podemos crear un paquete vacío al que poder añadirle cabeceras y ser utilizado en las simulaciones de *NS3*. Cada objeto *Packet* contiene un buffer de datos utilizado para almacenar las cabeceras del paquete, y es en este buffer donde hay que ir añadiendo de una en una cada cabecera que se quiera incrustar en él.

La clase *Packet* contiene métodos para añadir cabeceras, las cuales son tratadas en *NS3* mediante objetos de la clase *Header*. La clase *Header* contiene todos los elementos necesarios para crear una cabecera, desde los valores de los campos de la cabecera como los métodos para asignarlos. Estas cabeceras son añadidas a un objeto *Packet* mediante métodos de esta última clase como, por ejemplo, el método *AddHeader*, el cual añade la cabecera al final del buffer de datos del objeto *Packet*.

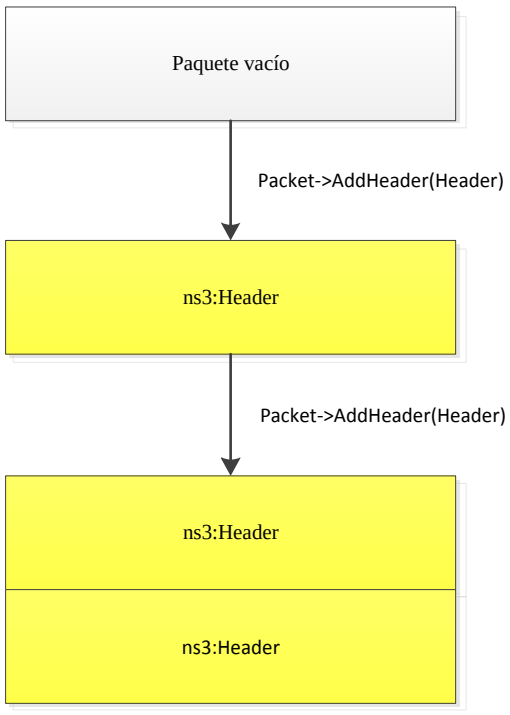

Figura 1.6.22 – Añadido ordenado de cabeceras a un paquete

Cada extensión de *IPv6*, cada opción *Hop by Hop*, la propia cabecera básica de *IPv6* y, en definitiva, cualquier protocolo que requiera añadir cabeceras a un paquete tiene su propio objeto para manipularlas, el cual hereda de la clase de *NS3 Header*, añadiendo a su

vez los métodos y los atributos propios de cada cabecera, según los campos que esta contenga.

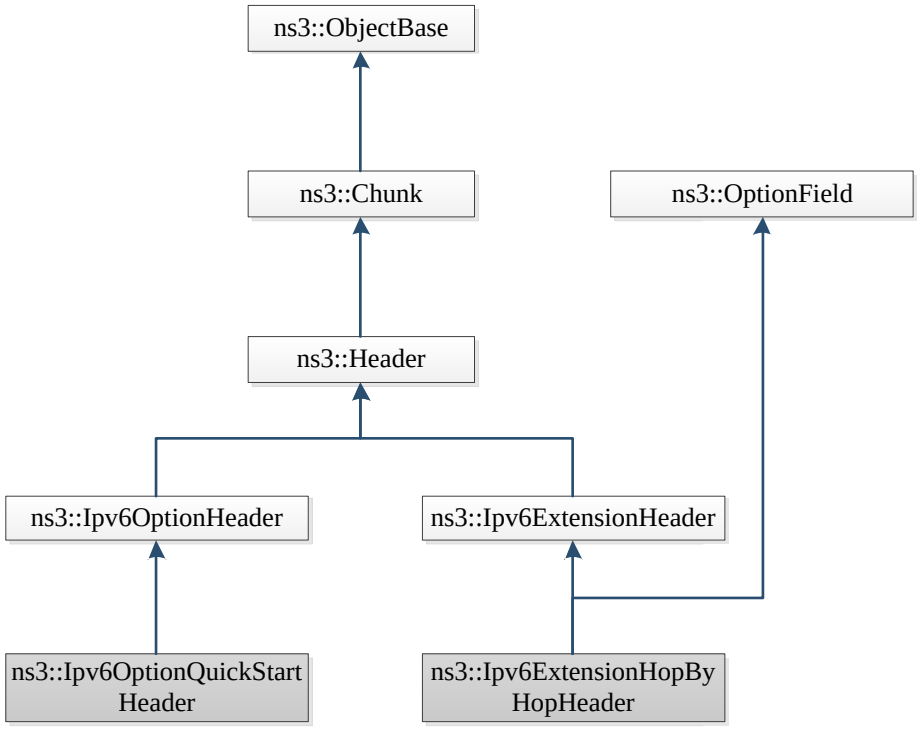

Figura 1.6.23 – Objetos de cabeceras en *NS3*

Cada cabecera debe ser añadida individualmente al paquete en orden descendente, empezando por las capas superiores y terminando por el nivel 2 de la torre de protocolos de *IP*:

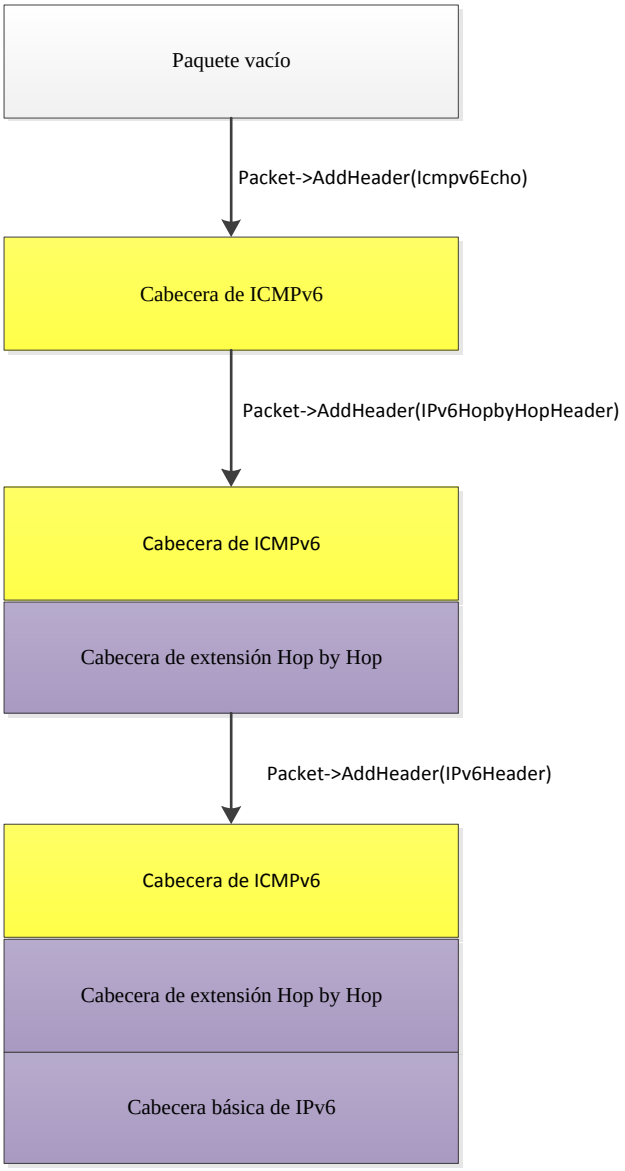

Figura 1.6.24 – Añadido de cabeceras a un paquete *IPv6*

Los paquetes contienen una funcionalidad adicional que ha sido aprovechada en este proyecto: el *Packet Tag*. El *Packet Tag* es un objeto de *NS3* creado a partir de la clase *Tag* que sirve para añadir información adicional a un objeto *Packet* sin alterar de modo alguno las cabeceras que éste lleva incorporadas.

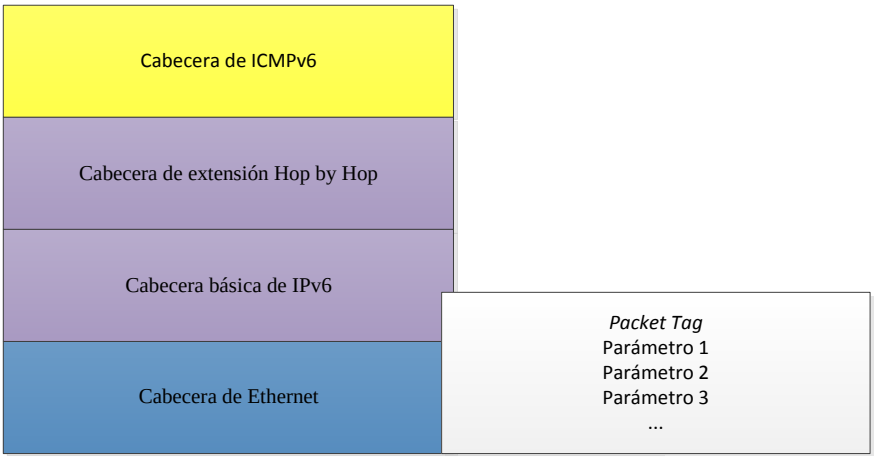

Figura 1.6.25 – Paquete con un *Packet Tag*

De este modo, un paquete puede transportar, además de las cabeceras, información que puede ser de utilidad a su paso por los distintos módulos de *NS3*. El programador puede crear nuevos *Tags* heredando de la clase principal *ns3:Tag*, a los que puede añadir los parámetros que considere oportunos. Además, el paquete puede llevar distintos tipos de *Tags* consigo, y éstos podrán ser leídos, modificados y eliminados en cualquier punto del simulador.

Los *Packet Tags* han sido utilizados en este proyecto para la generación de la cabecera de extensión *Hop by Hop* y de las cabeceras de sus opciones. Las cabeceras de estas opciones pueden ser modificadas en ruta y, gracias a los *Packet Tags*, se ha implementado esta característica en el simulador *NS3*.

1.6.2.3 Módulos principales de NS3

Las siguientes líneas explican parte del funcionamiento de *NS3* y se hace un recorrido por los módulos y funciones que han sido relevantes durante el desarrollo del proyecto.

NS3 contiene, entre otras, una implementación de *IPv6* independiente de cualquier otro software y totalmente funcional por sí misma; y pretende ser tan válida como lo pueden ser las implementaciones de *IPv6* de *Linux* o de *FreeBSD*. *NS3* tiene, por tanto, que trabajar sobre todos los niveles de la torre de protocolos de *IP*, tal y como se muestra en la figura 1.6.26:

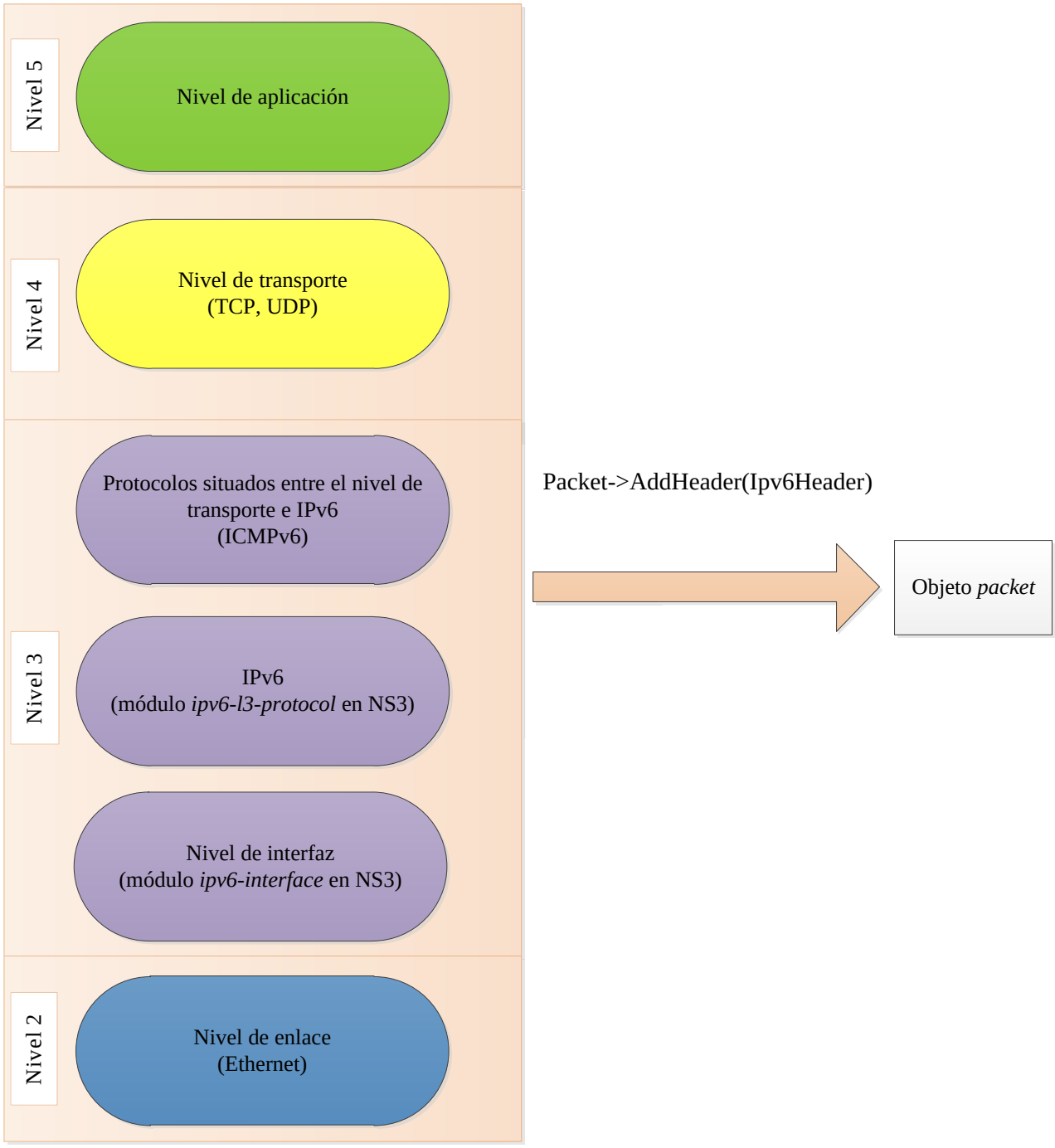

Figura 1.6.26 – Torre de protocolos tratada en este proyecto en *NS3*

El origen del paquete comienza en las capas superiores de la torre de protocolos de la figura 1.6.26, donde un objeto de *NS3* de la clase *packet* es creado, representando este

objeto un paquete, en principio vacío, al que cada capa le añade ordenadamente sus cabeceras correspondientes antes de ser enviado a un nodo distinto.

Lo que en la realidad sería el nivel 1 o nivel físico se obvia en el simulador, ya que la transmisión física del paquete es inexistente en un simulador. En una transmisión en *NS3*, la referencia del objeto *packet* es pasada como argumento a los métodos de recepción de paquetes del nodo receptor.

Aunque *NS3* tiene un gran número de módulos en su código, en este apartado son sólo nombrados y descritos aquellos que tienen el protagonismo a la hora de tratar paquetes que contienen opciones *Hop by Hop*. La figura 1.6.27 recoge estos módulos:

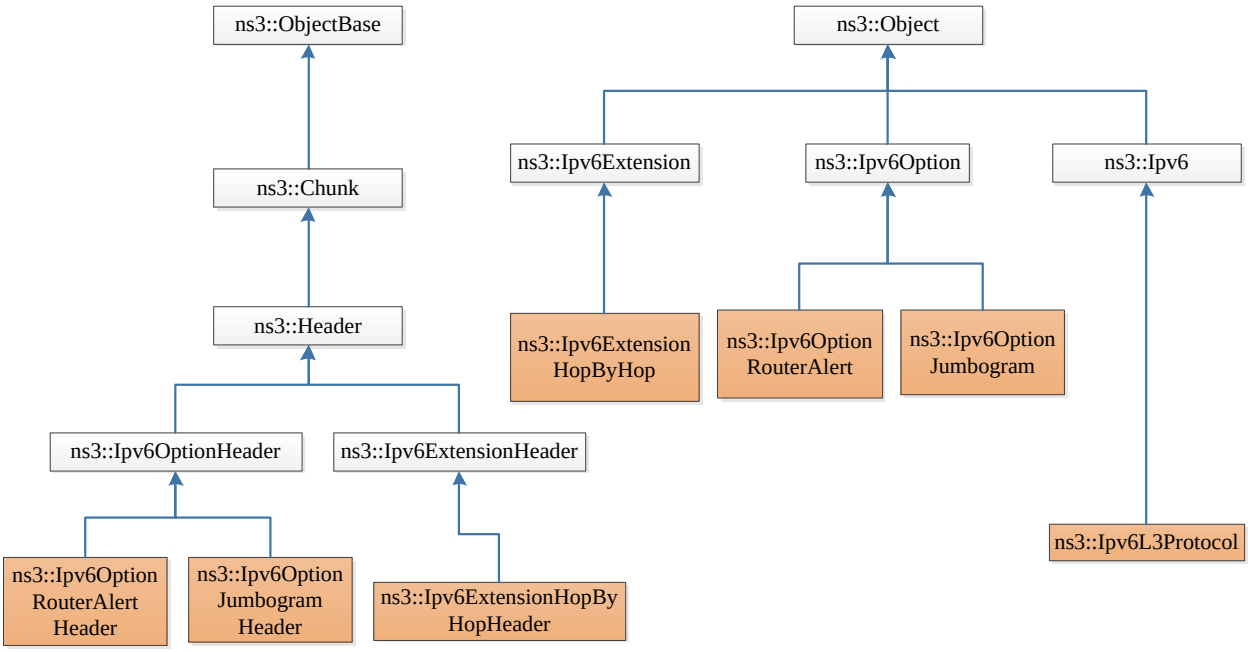

Figura 1.6.27 – Módulos principales en el uso de un paquete con extensión *Hop by Hop*

El módulo ***ipv6-l3-protocol*** es aquel en el que el paquete es tratado a nivel de IP. Las siguientes funciones han sido tratadas en este proyecto:

- ***Send*:** Llama a la función *BuildHdr*, explicada posteriormente, para añadir los campos de la cabecera básica de *IPv6* al paquete que va a ser enviado.

- ***Receive*:** Es invocada cuando un nodo recibe un paquete. Se encarga de revisar la cabecera básica de *IPv6* y llama a las funciones encargadas de procesar la cabecera de extensión *Hop by Hop*. Por último, decide si continuar con el procesamiento del paquete en capas superiores o, por el contrario, se reenvía el paquete hacia otro nodo.

- ***SendRealOut*:** Selecciona la interfaz por la que el paquete será enviado al siguiente nodo.

- ***BuildHdr*:** Añade los campos de la cabecera básica de *IPv6* al paquete.

- ***IpForward*:** Asigna la ruta adecuada al paquete para reenviarlo por el nodo adecuado.

- ***LocalDeliver*:** Es invocada cuando un paquete llega al nodo destino. Se encarga de examinar la cabecera básica de *IPv6* y todas las cabeceras de extensión. Tras ello, envía el paquete al protocolo superior a *IPv6*.

El módulo ***ipv6-interface*** trata el paquete a nivel de interfaz de red. La siguiente función ha sido tratada en el proyecto:

- ***Send*:** Una vez seleccionada la interfaz adecuada en el módulo *ipv6-l3-protocol*, esta función envía el paquete al nivel 2 de la torre de protocolos, donde será tratado a nivel de direcciones físicas o *MAC*.

El módulo ***ipv6-extension*** es el encargado de examinar y procesar las cabeceras de extensión de *IPv6*. Sus funciones más relevantes son:

- ***GetExtensionNumber*:** Devuelve el número identificador de la cabecera de extensión.

- ***Process*:** Procesa la cabecera de extensión y realiza todas las acciones pertinentes en cuanto a dicha extensión.

- ***ProcessOptions*:** Es invocada al procesar las cabeceras de extensión *Hop by Hop* y *Destination*. Se encarga de examinar las opciones presentes en la cabecera de extensión y de llamar a las funciones necesarias para el procesado individual de cada opción.

El módulo ***ipv6-option*** es el encargado de procesar las opciones de *IPv6*. Sus funciones más relevantes son las siguientes:

- ***GetOptionNumber*:** Devuelve el número identificador de la opción tratada.
- ***Process*:** Procesa la opción de *IPv6* y realiza todas las acciones pertinentes en cuanto a dicha opción.

El módulo ***ipv6-option-header*** es el encargado de tomar como parámetros los valores de los campos de la cabecera de una opción y serializarlos, es decir, convertirlos a un flujo de bytes que serán añadidos a un paquete.

1.6.2.4 Recorrido de un paquete IPv6 en NS3

Este proyecto ha realizado cambios en cuanto al recorrido del paquete por los módulos de *NS3* en el nivel 3 de la torre de protocolos de *IP*. Es conveniente entender el estado original de este proceso en *NS3* antes de realizar las modificaciones.

Para empezar, el siguiente diagrama muestra el recorrido del paquete durante su envío en un nodo:

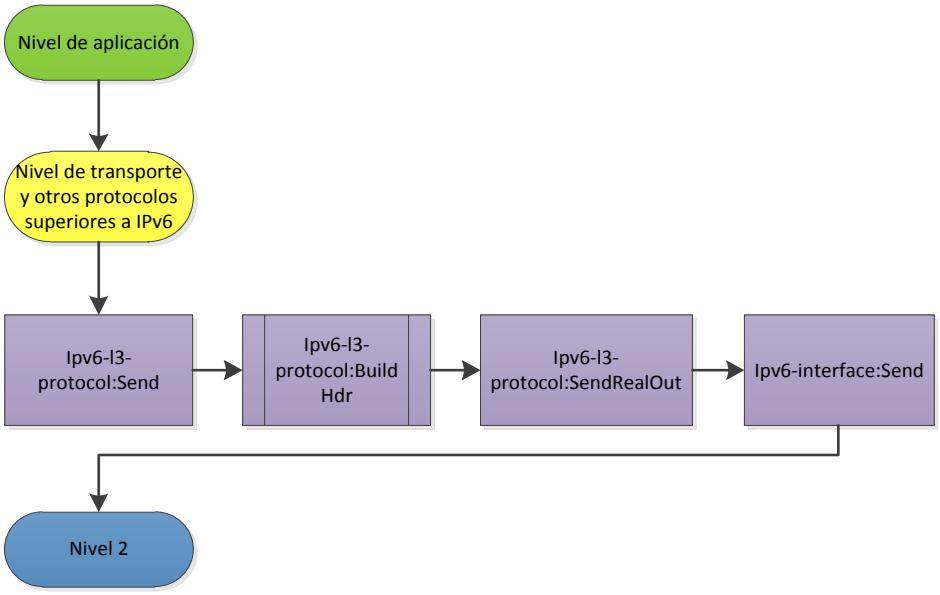

Figura 1.6.28 – Envío de un paquete *IPv6* en *NS3*

El siguiente diagrama muestra el recorrido del paquete durante su recepción en un nodo:

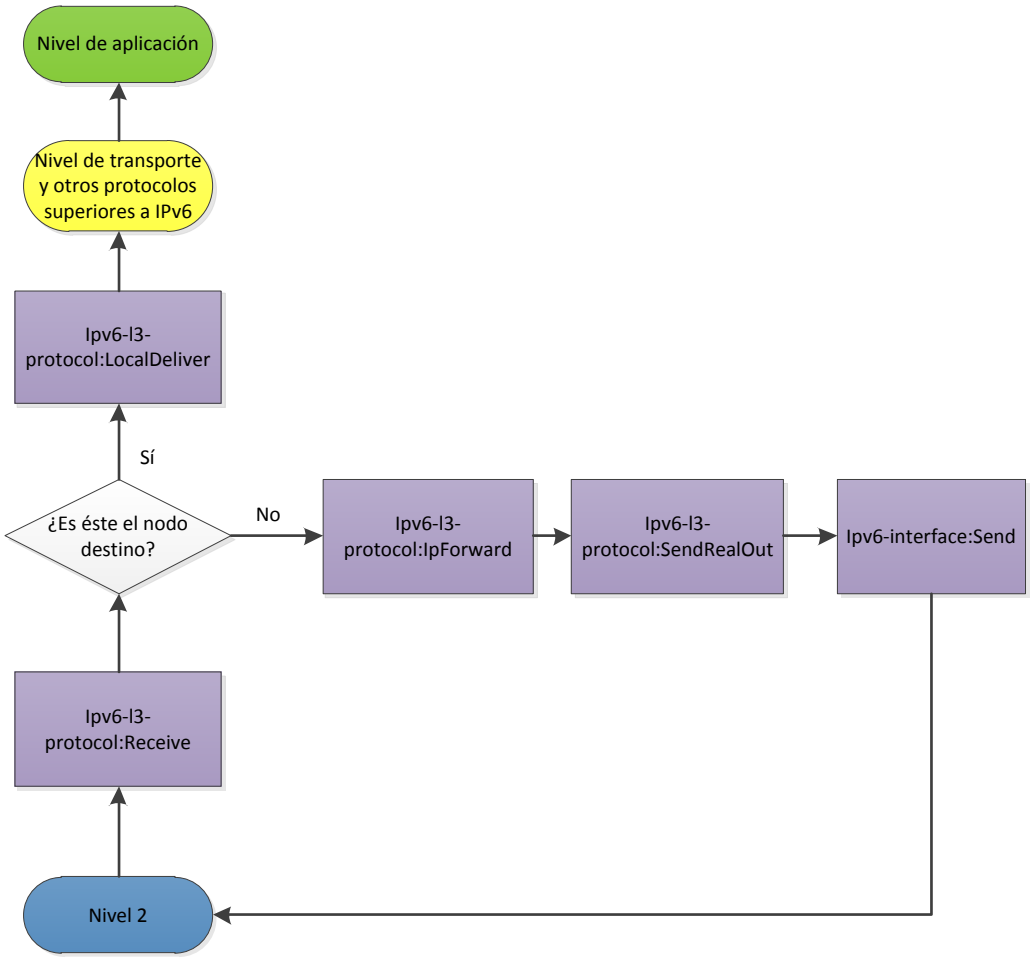

Figura 1.6.29 – Recepción de un paquete *IPv6* en *NS3*

1.6.2.5 Registro de mensajes en NS3

Anteriormente se hizo referencia al registro de eventos que ofrece el simulador *NS3*, el cual es utilizado en este proyecto para comprobar el funcionamiento de las características aquí programadas.

Este registro es llevado a cabo en *NS3* por un módulo de *logging* que muestra, a la salida de la ejecución de una simulación, mensajes programados en el código con información sobre lo que está sucediendo en un momento determinado.

Este módulo de registro tiene una jerarquía de niveles, de manera que un mensaje pertenece a un nivel concreto, permitiendo así mostrar solamente por pantalla aquellos niveles que el programador desee. Estos niveles se muestran en la figura 1.6.30:

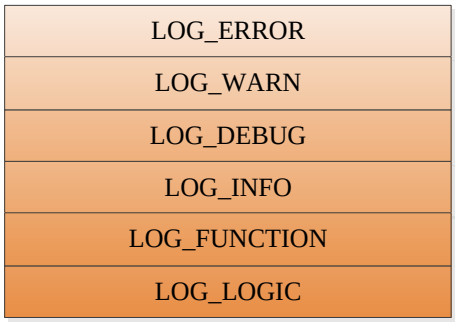

Figura 1.6.30 – Niveles de registro en *NS3*

A partir de estos niveles, una aplicación puede permitir mostrar sólo los mensajes de un determinado nivel. No obstante, existe la posibilidad de mostrar, con una sola instrucción, todos los niveles a la vez; o un nivel determinado y todos los niveles superiores a él; como por ejemplo, habilitar el nivel *LOG_DEBUG* y los dos superiores a él: *LOG_WARN* y *LOG_ERROR*.

La figura 1.6.19 mostraba la salida de una ejecución a todos los niveles de registro. Como ejemplo, la siguiente figura 1.6.31 muestra un extracto de esa salida, haciendo referencia a un mensaje a nivel de *LOG_FUNCTION*:

```
Ipv6Option:Process(0x843c2b8, 0x8435318, 99 (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 32
1:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe00:0004, 0)
```

Figura 1.6.31 – Mensaje de nivel *LOG_FUNCTION*

El mensaje de la anterior figura indica que se ha producido una llamada al método *Process* del módulo *Ipv6Option*, confirmando el reconocimiento de una opción de *IPv6* y su procesamiento.

La siguiente figura muestra otro ejemplo de la misma salida, esta vez de un mensaje perteneciente al nivel *LOG_LOGIC*:

```
Ipv6 Router PadN received and being processed on node 1
```

Figura 1.6.32 – Mensaje de nivel *LOG_LOGIC*

El anterior mensaje indica la llegada de un paquete *IPv6* con una opción *PadN* en su cabecera de extensión al nodo 1, el cual está procesando dicha opción.

1.7 Estado del arte

Mientras que la implementación de *IPv6* continúa extendiéndose en dispositivos como routers o en los principales sistemas operativos, las opciones *Hop by Hop* de *IPv6* no suelen tenerse demasiado en cuenta debido, probablemente, a su carácter experimental y a veces muy específico como para ser necesarias sus implementaciones.

Las opciones de relleno de bits *PadN y Pad1* deben poder ser interpretadas por toda implementación de *IPv6* según la *RFC 2460* [4], por lo que estas opciones son las más extendidas.

La opción *Router Alert* es bastante común, ya que se utiliza junto a protocolos de uso extendido como *Multicast Listener Discovery* o *Resource Reservation Protocol*. La opción *Jumbogram* puede verse también en las implementaciones de *IPv6* de *Linux* y de *FreeBSD*, aunque su uso no es tan común, siendo inútil en la red de *Internet* al estar enfocado a trabajar con *MTUs* de gran tamaño que no tienen cabida en dicha red.

En cambio, las opciones *Quick Start* y *Calipso* no están presentes en ninguna parte del *kernel* o núcleo de los sistemas operativos citados en el anterior párrafo. Ambos están enfocados a su uso fuera de *Internet*, en redes privadas con fines específicos, y sus implementaciones no son nada comunes, siendo éstas aún experimentales. Independientemente de lo anterior, estas opciones han sido estudiadas e implementadas en este proyecto con el fin de poder ser utilizadas en el futuro por cualquier usuario interesado en hacer uso o estudio de ellas.

1.8 Alcance

El objetivo principal del proyecto es el de mejorar las características actuales del programa *Network Simulator 3*, software de código abierto licenciado bajo la versión 2 de la *GNU General Public License*, lo cual significa que se dispone de acceso a todo el código del programa y puede ser modificado sin límite alguno.

Para ello, se ha revisado la implementación del protocolo *IPv6* del programa y se ha realizado una serie de cambios en el código con el objetivo de cumplir los objetivos del proyecto.

No es objeto de este proyecto implementar los protocolos que hagan uso de las opciones implementadas, estando éste enfocado principalmente en *Hop by Hop* y sus opciones.

1.9 Desarrollo

1.9.1 Introducción

En este apartado se detallan los cambios introducidos por el proyecto en el código de *NS3* y las aplicaciones programadas para demostrar el funcionamiento de las nuevas funcionalidades añadidas.

El desarrollo del proyecto comprende:

- La programación de los aquí denominados *Hop by Hop Tags*.
- La programación de las opciones *Router Alert*, *Jumbogram*, *Calipso* y *Quick Start*.
- La programación de unos escenarios que simulen situaciones que validen el funcionamiento correcto de las opciones.

1.9.2 Programación de los Hop by Hop Tags

NS3 no cuenta con una forma práctica y cómoda de añadir y modificar opciones de *Hop by Hop* a un paquete. Debido a que las opciones pueden ser modificadas en los nodos intermedios durante el trayecto del paquete desde el origen a su destino, es necesario implementar un sistema por el que la cabecera de extensión *Hop by Hop* fuera eliminada del paquete durante la recepción y fuera sustituida por una nueva con los nuevos valores de los campos en las opciones.

Para lograr este objetivo se ha utilizado una herramienta propia del simulador *NS3*: el *Packet Tag*. Un *Packet Tag* es un conjunto de información que puede adherirse al paquete y ser leído en cualquier punto del simulador, sin interferir de ningún modo con las cabeceras que el paquete contiene. El uso de los Packet Tags para este fin es adecuado debido a la diversidad de opciones de IPv6¸ las cuales son procesadas cada una de forma distinta. Los Packet Tags y su versatilidad ayudan en este cometido, ya que la información de los Tags es fácilmente accesible desde cualquier módulo de NS3 sin tener que hacer cambio alguno en la estructura de archivos del simulador.

Dispondremos de un *Packet Tag* específico para cada opción de *IPv6 Hop by Hop*, a excepción de las opciones *Pad1* y *PadN*, las cuales son generadas automáticamente por *NS3* y no es necesario adaptarlas a este sistema.

La figura 1.9.1 muestra la idea del sistema de los *Hop by Hop Tags*. Representa el paso de un paquete por dos nodos, uno emisor y otro receptor:

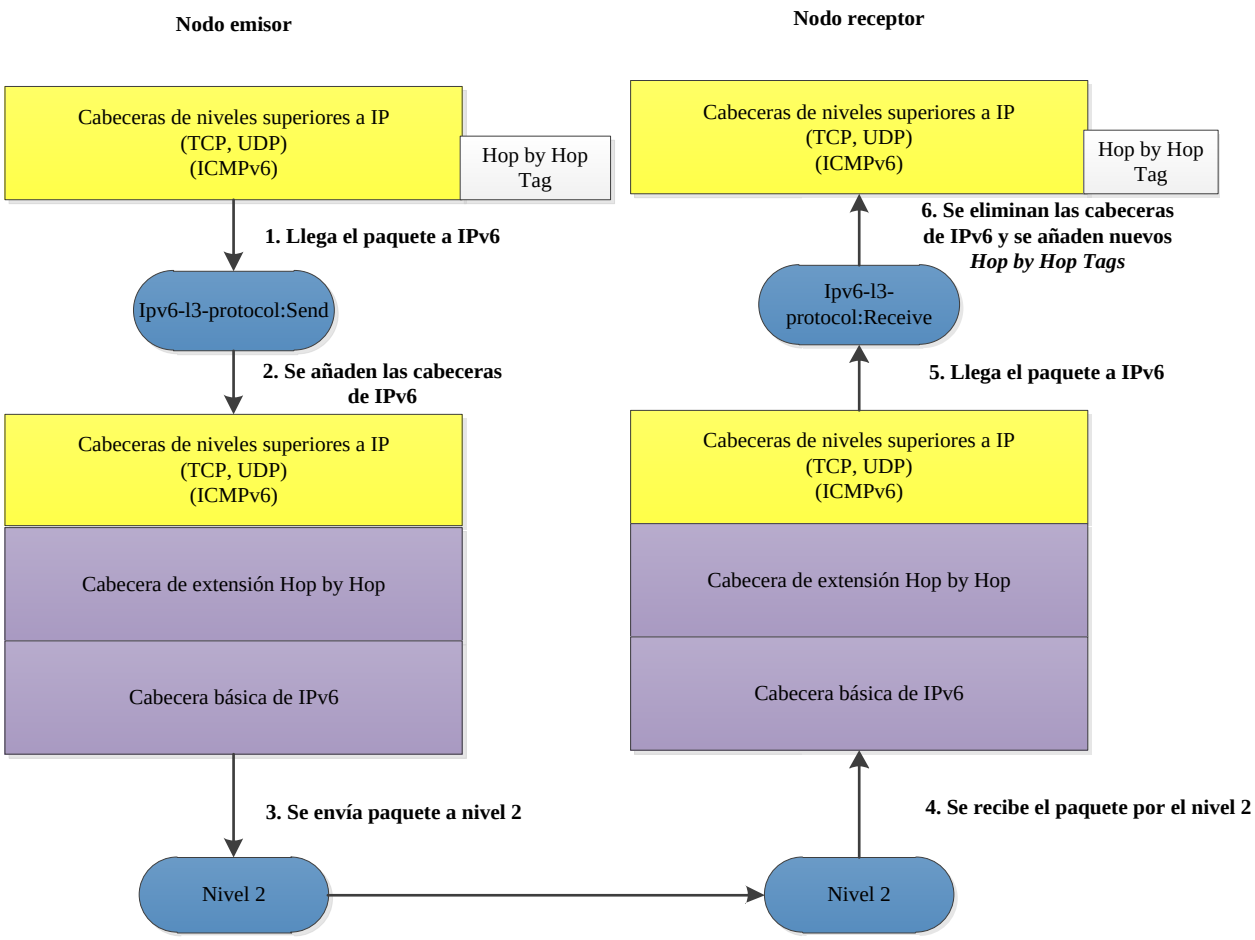

Figura 1.9.1 – Añadido y eliminado de *Hop by Hop Tags* a un paquete

Los pasos que sigue el paquete en la figura 1.9.1 se detallan a continuación:

1. Un paquete llega a la capa de *IPv6* de *NS3* para ser enviado a otro nodo. Hasta ahora no contiene ninguna cabecera de *IPv6*; tan sólo contiene cabeceras de protocolos superiores. El paquete lleva adherido un *Hop by Hop Tag*.

2. El módulo *ipv6-l3-protocol* detecta la presencia de un *Hop by Hop Tag*. Éste toma la información contenida en él para formar la cabecera de extensión *Hop by Hop* y la opción contenida en el *Tag*. Posteriormente, *ipv6-l3-protocol* añade la cabecera básica de *IPv6*. El *Hop by Hop Tag* ha sido eliminado y ya no acompaña al paquete.

3. El paquete ya tiene todas las cabeceras de nivel de *IP* y ya está listo para ser enviado al nivel 2.

4. El nodo receptor recibe el paquete por el nivel 2.

5. El paquete llega intacto a nivel de *IP*.

6. El módulo *ipv6-l3-protocol* elimina la cabecera básica de *IPv6* y la cabecera de extensión *Hop by Hop* del paquete. Durante el procesamiento de la opción u opciones encontradas en el paquete, se añade un *Hop by Hop Tag* para cada opción, almacenando el *Tag* los nuevos campos que serán añadidos. El paquete está listo para repetir el mismo proceso a la hora de ser reenviado hacia el siguiente nodo.

No todas las opciones cambian durante el trayecto del paquete. Sin embargo, todos los paquetes serán tratados de esta forma, contribuyendo a la homogeneidad del proceso a la hora de manipular un paquete con la extensión *Hop by Hop*.

Cada *Hop by Hop Tag* es una clase que hereda los métodos de otra clase ya existente en *NS3*: la clase *Tag*. La clase *Hop by Hop Tag* implementa, además, los métodos que almacenan los valores de los campos de cabecera de las opciones de *IPv6* y un método que crea la cabecera de la opción a partir de los valores anteriores. La figura 1.9.2 muestra la estructura de una clase *Hop by Hop* genérica, expuesta aquí como ejemplo:

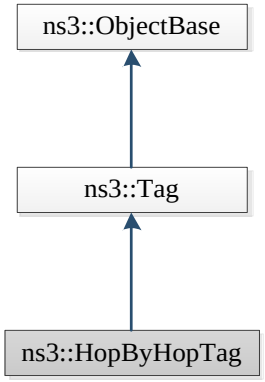

Figura 1.9.2 – Clase *HopbyhopTag*

Los siguientes métodos, pertenecientes a esta clase, han sido tratados durante el desarrollo del proyecto:

- ***Serialize:*** Serializa los parámetros en un buffer de datos contenido en el *Tag*. Todos los parámetros deben ser serializados en este buffer para poder ser transportados con el paquete.

- ***Deserialize:*** Lee el contenido serializado y asigna los valores leídos a los datos miembro del *Packet Tag*; en este caso, a la variable de ejemplo *parameter*.

- ***GetParameter:*** Devuelve el dato miembro *parameter*. Se crearán tantos métodos de este tipo como campos tenga la opción.

- ***SetParameter:*** Asigna al dato miembro *parameter* el valor pasado como parámetro a este método. Se crearán tantos métodos de este tipo como campos tenga la opción.

- ***Process:*** Este método recoge toda la información contenida en el *Tag* y creará una cabecera *Ipv6OptionHeader* con los datos obtenidos. El método devuelve la cabecera como resultado.

Tras programar la clase *Hop by Hop Tag*, se ha realizado una serie de cambios en el módulo *ipv6-l3-protocol*, tanto en el método *ipv6-l3-protocol::Send* como en *ipv6-l3-protocol::Receive*. Estos cambios han sido introducidos para realizar la búsqueda automática de *Hop by Hop Tags* en un paquete y añadir a dicho paquete las cabeceras de la extensión *Hop by Hop* y las opciones encontradas.

Para ello ha sido programado un nuevo método en el módulo *ipv6-l3-protocol* llamado *ipv6-l3-protocol::CheckHopbyhopTags*, el cual examina el paquete que se le ha pasado como argumento en búsqueda de cualquier *Hop by Hop Tag* perteneciente a una opción *Hop by Hop* que éste pueda contener. Los *Hop by Hop Tags* que este método busca en un paquete son:

- *Ipv6OptionRouterAlertTag*
- *Ipv6OptionJumboTag*
- *Ipv6OptionCalipsoTag*
- *Ipv6OptionQuickStartTag*

Los *Tags* anteriores son descritos en los apartados sucesivos al actual, donde se habla de los cambios producidos para cada opción *Hop by Hop* programada en el proyecto.

El diagrama de la figura 1.9.3 muestra el proceso de llamadas a funciones realizadas cuando el método *ipv6-l3-protocol::CheckHopbyhopTags* recibe un paquete:

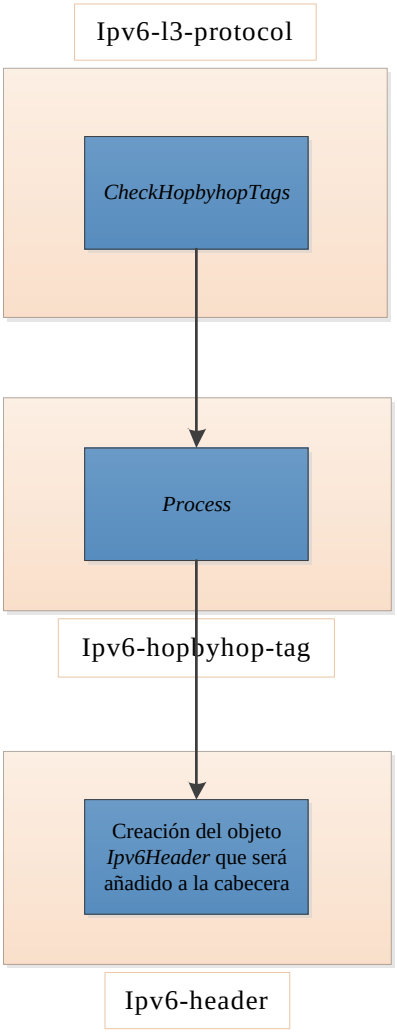

Figura 1.9.3 – Proceso de llamadas al utilizar *ipv6-l3-protocol::CheckHopbyhopTags*

Cuando el método *ipv6-l3-protocol::CheckHopbyhopTags* encuentra uno de los *Tags* definidos para las opciones de *IPv6*, éste realiza una llamada al método *Process* del que dispone el *Tag* encontrado. En *Process*, se leen los parámetros almacenados en el *Tag* y se crea la cabecera de la opción que será añadida al paquete utilizando la clase *ipv6-header*.

En cuanto al momento en el que *ipv6-l3-protocol:CheckHopbyhopTags* debe ser utilizado durante el paso de un paquete por la capa de *IPv6*, éste debe ser invocado en dos ocasiones.

La primera de ellas debe realizarse justo antes de que la capa de *IP* añada la cabecera básica de *IPv6* al paquete, ya que el lugar de la cabecera de extensión *Hop by Hop* dentro

de un paquete va siempre, sin excepción, junto a la cabecera básica. La figura 1.9.4 indica la posición del método dentro de la capa de *IP* durante el proceso de envío de un paquete:

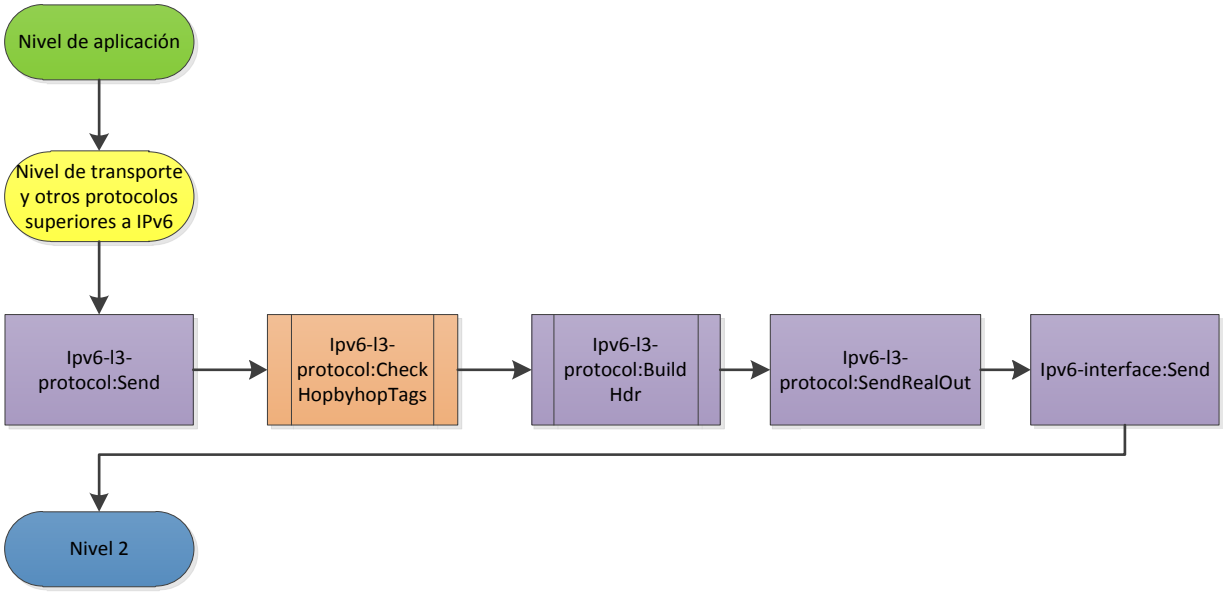

Figura 1.9.4 – Envío de un paquete *IPv6* en *NS3*

La segunda de ellas se realiza después de producirse el procesamiento de la cabecera de extensión *Hop by Hop* al recibir un paquete. Al procesar la extensión *Hop by Hop*, ésta es eliminada del paquete, momento justo en el que una llamada al método *ipv6-l3-protocol::CheckHopbyhopTags* agrega de nuevo la cabecera *Hop by Hop*, con nuevos datos si se da el caso de que han cambiado en ruta. La figura 1.9.5 indica el lugar correspondiente de esta llamada dentro del proceso de recepción de un paquete:

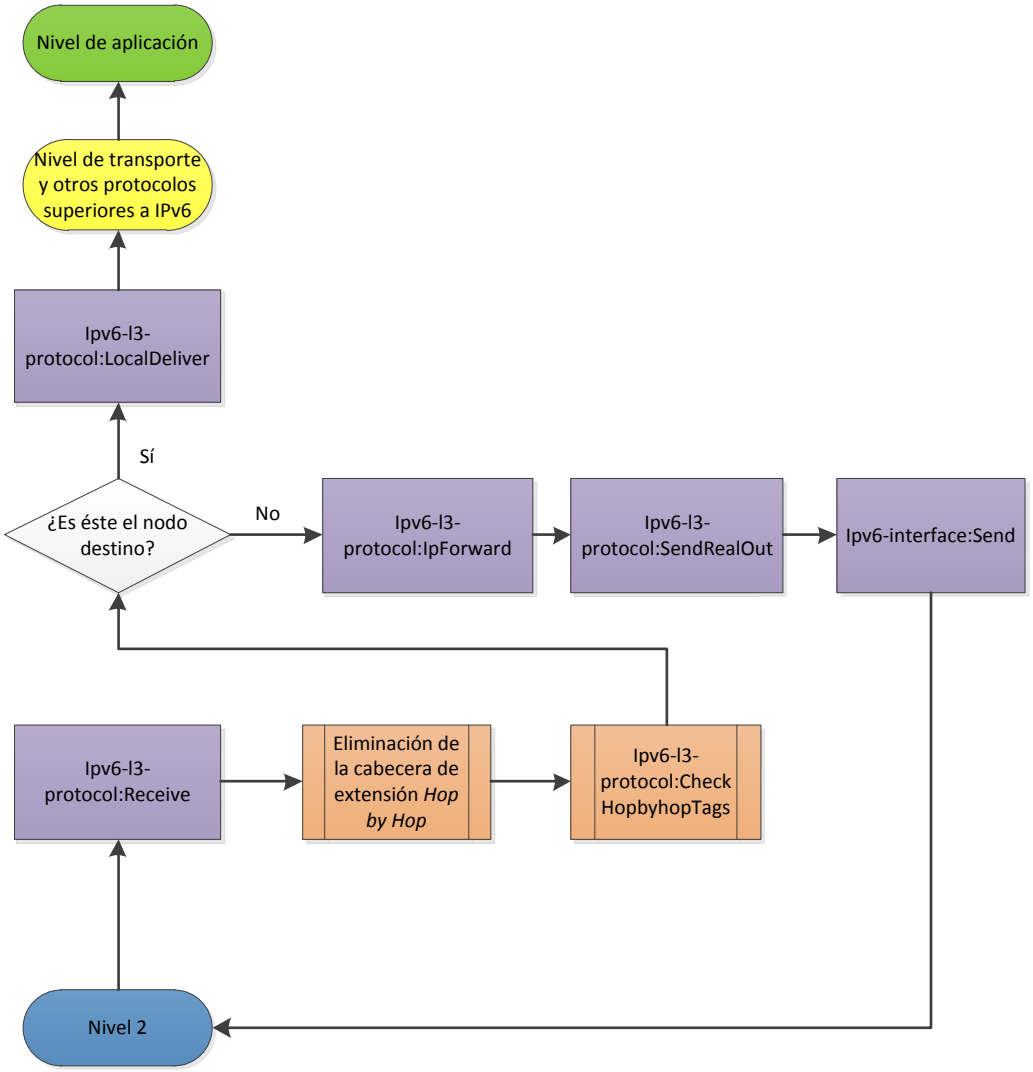

Figura 1.9.5 – Recepción de un paquete *IPv6* en *NS3*

1.9.3 Programación de la opción Router Alert

La opción de *IPv6 Router Alert* se encuentra ya implementada en *NS3* en los siguientes aspectos:

- Los métodos para crear una cabecera de opción *Router Alert* se encuentran disponibles en el módulo *ipv6-option-header*.
- La opción es reconocida por el módulo *ipv6-option* y es procesada correctamente.

Puesto que el propósito de la opción *Router Alert* es meramente informativo, no ha sido necesario añadir ninguna funcionalidad adicional especial para dicha opción. Los cambios que han sido realizados para esta opción han sido:

- Creación de un *Hop by Hop* Tag para *Router Alert* llamado *Ipv6OptionRouterAlertTag*.
- Añadido de compatibilidad de la opción con el sistema de procesamiento de *Hop by Hop Tags*.
- Creación de un mensaje informativo para el *LOG* de *NS3* avisando del procesamiento de la opción en el nodo donde se recibe el paquete.

Hop by Hop Tag de la opción Router Alert

La clase *Ipv6OptionRouterAlertTag*, añadida al módulo *ipv6-hopbyhopt-tag*, tiene la siguiente estructura:

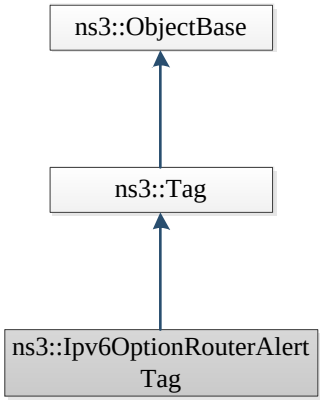

Figura 1.9.6 – Clase Ipv6RouterAlertTag

1.9.4 Programación de la opción Jumbogram

La opción *Jumbogram* se encuentra ya implementada en los siguientes aspectos:

- Los métodos para crear una cabecera de opción *Router Alert* se encuentran disponibles en el módulo *ipv6-option-header*.

- La opción es reconocida por el módulo *ipv6-option* y es procesada correctamente.

Los cambios que han sido introducidos para esta opción han sido:

- Creación de un *Hop by Hop* Tag para *Jumbogram* llamado *Ipv6OptionJumboTag*.
- Añadido de compatibilidad de la opción con el sistema de procesamiento de *Hop by Hop Tags*.
- Creación de un mensaje informativo para el *LOG* de *NS3* avisando del procesamiento de la opción en el nodo donde se recibe el paquete.
- Añadido al procesado de la opción la comprobación de los errores de formato del paquete vistos en el apartado 1.6.1.3 de la Memoria.
- Modificación del código de *NS3* para poder utilizar *MTUs* de 32 bits.

Hop by Hop Tag de la opción Jumbogram

La clase *Ipv6OptionJumbogramTag*, añadida al módulo *ipv6-hopbyhopt-tag*, contiene los mismos métodos descritos en la clase *Hop by Hop Tag* genérica adaptados al único parámetro que la opción contiene, *Jumbo Payload Length*. Esta clase tiene la estructura de la figura 1.9.7:

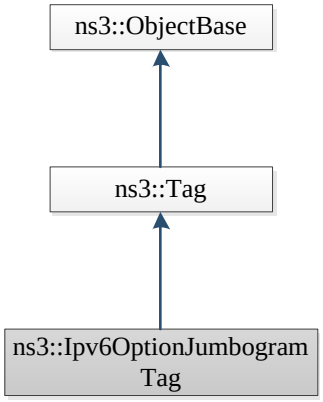

Figura 1.9.7 – Clase Ipv6*Jumbogram*Tag

Antes de hablar del procesamiento que ha sido programado para la opción *Jumbogram*, es necesario describir la clase diseñada específicamente para dicho procesamiento. Esta clase ha sido denominada *Ipv6Plen0Tag* y su propósito es el de avisar con su presencia de que un paquete contiene el valor 0 en el campo *Payload Length* en la cabecera de *IPv6*. Su estructura se muestra en la figura 1.9.8 :

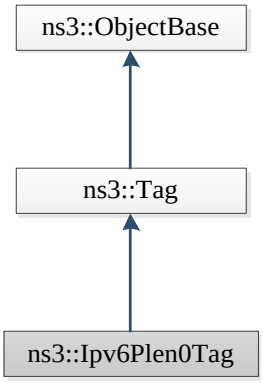

Figura 1.9.8 – Clase *Ipv6Plen0Tag*

Procesamiento de la opción Jumbogram

Una vez descritas ambas clases, a continuación, en la figura 1.9.9, se presenta el diagrama de procesamiento de un paquete que contiene la opción *Jumbogram*:

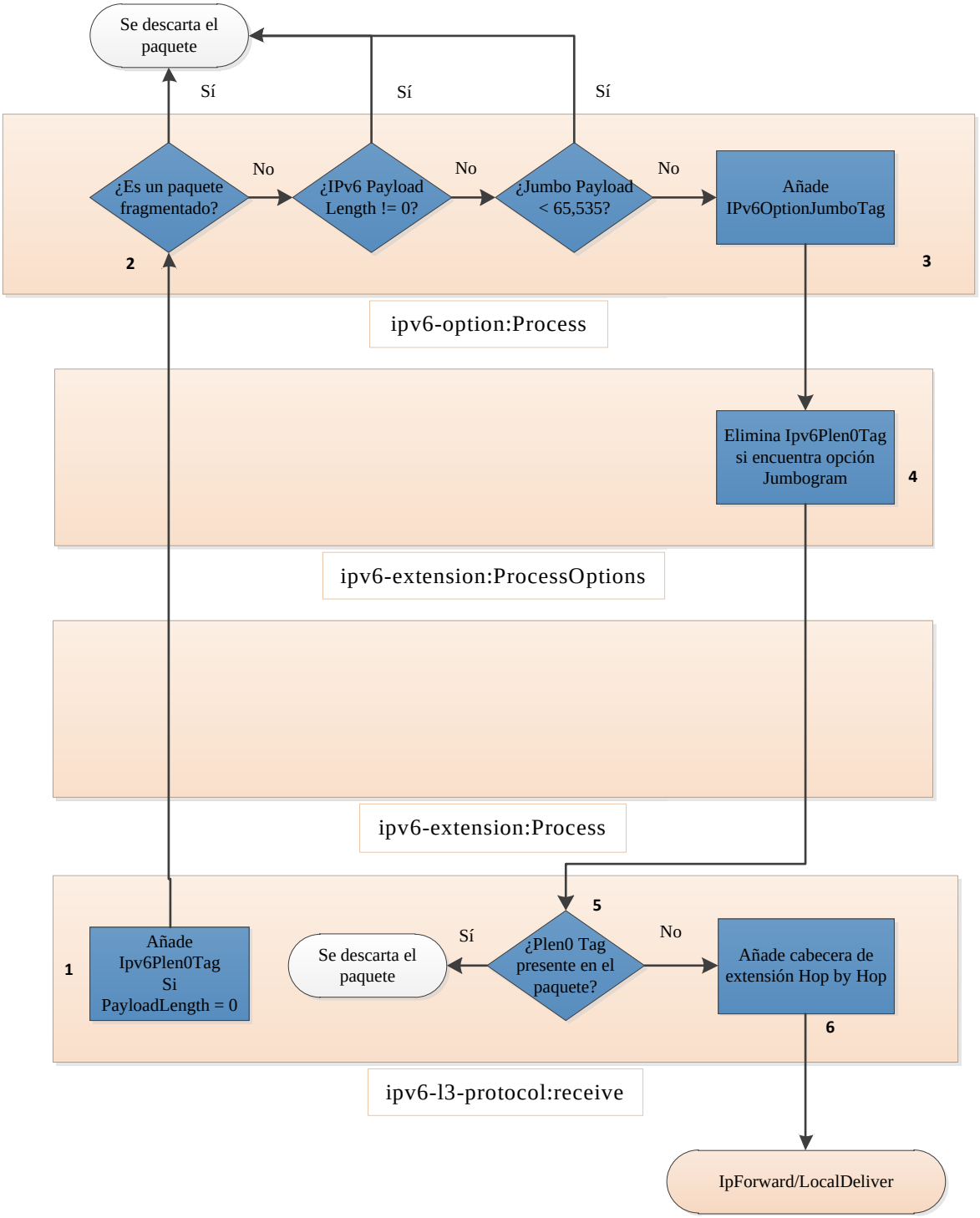

Figura 1.9.9 – Procesamiento de la opción *Jumbogram*

Los pasos marcados en la figura 1.9.9 se describen en las siguientes líneas:

1. El método *ipv6-l3-protocol:Receive* añade al paquete recibido un *Packet Tag* de tipo *Ipv6Plen0Tag* en el caso de que el campo *Payload Length* de la cabecera

de *IPv6* sea 0. Este *Tag* no contiene información almacenada alguna; simplemente servirá como un dato auxiliar más adelante en el procesamiento.

Tras ello, *ipv6-l3-protocol:Receive* realiza la llamada a *ipv6-extension:Process* para procesar la extensión *Hop by Hop*. Éste último realiza una llamada al método *ipv6-extension:ProcessOptions*, el cual examina la cabecera de extensión en busca de opciones de *IPv6* y encuentra la opción *Jumbogram*.

2. Al encontrar una opción *Jumbogram* en el paquete, el anterior método *ipv6-extension:ProcessOptions* realiza una llamada al método I*pv6OptionJumbogram:Process*. Aquí se realizan las tres comprobaciones de formato mostradas en el diagrama y se descarta el paquete en el caso de encontrar cualquiera de ellas.

3. Se añade un nuevo *Ipv6OptionJumboTag* para el futuro añadido de la opción al paquete.

4. Se elimina del paquete el antes mencionado *Ipv6Plen0Tag* si una opción *Jumbogram* es encontrada en la cabecera del paquete.

5. Se realiza la última comprobación de formato de cabecera de la opción *Jumbogram*, referente a la ausencia de opción *Jumbogram* en el paquete cuando el campo *Payload Length* de *IPv6* es 0.

6. Se elimina la cabecera de extensión *Hop by Hop* del paquete y se procede a la búsqueda de *Hop by Hop Tags* en éste para añadir una nueva cabecera de extensión. Tras ello, el paquete continúa su camino hacia capas superiores en el nodo o hacia su destino a través de otros nodos.

Adaptación de la MTU en NS3

El último cambio realizado sobre *NS3* referente a la opción *Jumbogram* es aquel que permite al simulador trabajar con una *MTU* de 32 bits, ya que la versión 3.18 de *NS3* utilizada en este proyecto no permite esta característica.

Los cambios para este cometido son bastante simples. *NS3* almacena en sus distintos módulos el valor de la *MTU* en variables de tipo enteros sin signo de 16 bits *(uint16_t).* Cambiar todas estas variables por enteros sin signo de 32 bits *(uint32_t)* y los métodos que trabajan con ellas permitirá a *NS3* trabajar con *MTUs* de 32 bits sin ningún problema, permitiendo crear y enviar paquetes con *payloads* mayores de 65,535 bytes. Los módulos modificados han sido los siguientes:

- *bridge-net-device*
- *csma-net-device*
- *emu-net-device*
- *fd-net-device*
- *fd-net-device*
- *loopback-net-device*
- *error-net-device*
- *lte-net-device*
- *mesh-point-device*
- *net-device*
- *simple-net-device*
- *point-to-point-net-device*
- *aloha-noack-net-device*
- *non-communicating-net-device*
- *tap-bridge*
- *uan-net-device*
- *virtual-net-device*
- *wifi-net-device*

- *wimax-net-device*

Todos estos módulos deben ser arreglados para poder compilar el simulador *NS3* y utilizar *MTUs* de 32 bits de longitud en cualquier ámbito del programa.

1.9.5 Programación de la opción Calipso

La opción *Calipso* no se encuentra programada en ningún ámbito en *NS3*, por lo que ha habido que partir de cero en este caso. Los cambios introducidos en el simulador respecto a esta opción han sido los siguientes:

- Añadido de los métodos de creación de la cabecera de la opción *Calipso* en el módulo *ipv6-option-header*.
- Añadido de los métodos de procesamiento en el módulo *ipv6-option* para la opción *Calipso*.
- Creación de un *Hop by Hop* Tag para Calipso llamado Ipv6OptionCalipsoTag.
- Dada a las interfaces de *IPv6* la capacidad de realizar controles de acceso basados en *Calipso*.
- Añadido de compatibilidad de la opción con el sistema de procesamiento de *Hop by Hop Tags*.
- Creación de un mensaje informativo para el *LOG* de *NS3* avisando del procesamiento de la opción en el nodo donde se recibe el paquete.

Cabecera de la opción Calipso

La creación de la cabecera de la opción *Calipso* se realiza a semejanza de las opciones *Router Alert* y *Jumbogram*, las cuales estaban ya programadas en *NS3*. Para ello ha sido necesario crear la clase llamada *Ipv6OptionCalipsoHeader* que herede los métodos de la

clase *Ipv6OptionHeader*. También ha sido necesario añadir los métodos y variables privadas referentes a cada campo de la opción *Calipso*. Su estructura se muestra en la figura 1.9.10:

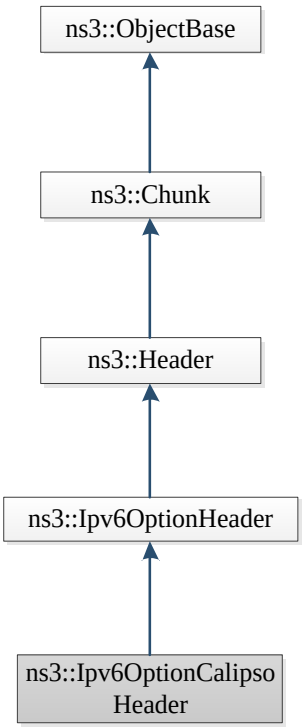

Figura 1.9.10 – Clase Ipv6OptionCalipsoHeader

Hop by Hop Tag de la opción Calipso

La clase *Ipv6OptionCalipsoTag*, añadida al módulo *ipv6-hopbyhopt-tag*, contiene los mismos métodos descritos en la clase *Hop by Hop Tag* genérica más aquellos referentes a los campos de la cabecera *Calipso*. Esta clase tiene la estructura de la figura 1.9.11:

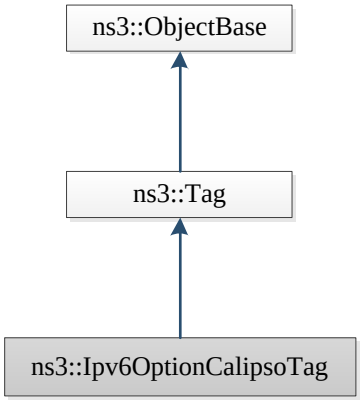

Figura 1.9.11 – Clase Ipv6OptionCalipsoTag

La clase anterior contiene un método adicional exclusivo para esta opción: *Send*. Este método aparecerá más adelante, cuando se hable del procesamiento de la opción, y es utilizado en el control de acceso de *Calipso*.

Cambios a nivel de interfaz

Otro de los cambios introducidos en *NS3* ha sido realizado a nivel de interfaz de red. El control de acceso de *Calipso* requiere que las interfaces de red almacenen cierta información relativa a la configuración de *Calipso*, por lo que la clase *ipv6-interface* contiene ahora los datos miembro y métodos que aparecen tras la figura 1.9.12.

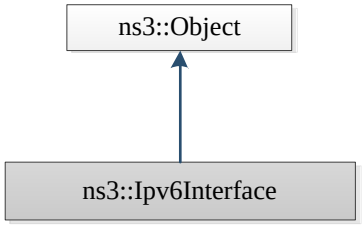

Figura 1.9.12 – Estructura de la clase *Ipv6Interface*

Cada interfaz almacena, con los cambios introducidos en este proyecto, los dos siguientes parámetros:

- ***m_knownDoiList***: Lista de Dominios de Interpretación reconocidos por la interfaz, formada esta lista por estructuras *doiStruct*.
- ***m_permittedDoiList***: Lista de Dominios de Interpretación permitidos por la interfaz, formada por los números identificadores de los Dominios de Interpretación.

En el anterior párrafo ha aparecido un término desconocido: *doiStruct*. Un *doiStruct* es una estructura definida en el módulo *ipv6-interface* para almacenar los siguientes datos:

- El número identificador del Dominio de Interpretación.

- El máximo nivel de sensibilidad permitido.
- El mínimo nivel de sensibilidad permitido.
- El máximo nivel de *Compartment Bitmap* permitido.
- El mínimo nivel de *Compartment Bitmap* permitido.

El código que define la estructura *doiStruct* es el que aparece en la figura 1.9.13:

```
struct doiStruct
{
  uint32_t doi;
  uint8_t maxLevelRange;
  uint8_t minLevelRange;
  uint8_t maxCompRange;
  uint8_t minCompRange;
};
```

Figura 1.9.13 – Estructura *doiStruct*

Los métodos añadidos al módulo *ipv6-interface* permiten manipular los anteriores parámetros y configurar las características de *Calipso* en una interfaz de red:

- ***SetMaxLevelRange***: Asigna el valor máximo permitido para el nivel de sensibilidad.
- ***GetMaxLevelRange***: Obtiene el valor máximo permitido para el nivel de sensibilidad.
- ***SetMinLevelRange***: Asigna el valor mínimo permitido para el nivel de sensibilidad.
- ***GetMinLevelRange***: Obtiene el valor mínimo permitido para el nivel de sensibilidad.
- ***SetMaxCompRange***: Asigna el valor máximo permitido para el nivel de *Compartment Bitmap*.
- ***GetMaxCompRange***: Obtiene el valor máximo permitido para el nivel de *Compartment Bitmap*.

- ***SetMinCompRange***: Asigna el valor mínimo permitido para el nivel de *Compartment Bitmap*.
- ***GetMinCompRange***: Obtiene el valor mínimo permitido para el nivel de *Compartment Bitmap*.
- ***AddDoiToKnownList***: Añade a la lista *m_knownDoiList* una nueva estructura *doiStruct*.
- **EraseDoiFromKnownList**: Elimina una estructura *doiStruct* de la lista *m_knownDoiList*.
- **GetDoiPermittedDoiList**: Devuelve la lista *m_permittedDoiList*.
- ***AddDoiToPermittedList***: Añade a la lista *m_permittedDoiList* un nuevo Dominio de Interpretación.
- ***EraseDoiFromPermittedList***: Elimina un Dominio de Interpretación de la lista *m_permittedDoiList*.
- ***GetIndexForDoi***: Devuelve la posición de una estructura *doiStruct* dentro de la lista *m_knownDoiList* según el valor del Dominio de Interpretación que se le dé como argumento.
- ***isDoiInKnownList***: Comprueba la presencia de un Dominio de Interpretación en la lista *m_knownDoiList*.

Procesamiento de la opción Calipso

Durante el proceso de transmisión de un paquete que contiene la opción *Calipso*, el paquete debe ser revisado por las interfaces de salida o de entrada del nodo que transmite o recibe el paquete, respectivamente. Durante esta revisión, se realiza la comprobación de una serie de condiciones basadas en los parámetros configurados en cada interfaz y los campos del paquete que se envía o se recibe. Si alguna de estas condiciones no se cumple, el paquete será descartado y no se continuará su envío o procesamiento. Este control de acceso tiene las siguientes condiciones, realizadas en el orden en el que aparecen:

1. Se calcula el *checksum* del paquete enviado o recibido y se compara con el *checksum* que el paquete incluye en su cabecera.

2. El valor del campo del Dominio de Interpretación del paquete debe estar incluido en el listado de Dominios de Interpretación conocidos por la interfaz de envío o de recibo.

3. El valor del campo del Dominio de Interpretación del paquete debe estar incluido en el listado de Dominios de Interpretación aceptados por la interfaz que procese en ese momento el paquete.

4. El valor del Nivel de Sensibilidad contenido en el paquete debe encontrarse entre los límites mínimo y máximo permitidos para la interfaz en uso y Dominio de Interpretación indicado por el paquete.

5. El valor del *Compartment Bitmap* contenido en el paquete debe encontrarse entre los límites mínimo y máximo configurados para la interfaz en uso y Dominio de Interpretación indicado por el paquete.

Estas cinco condiciones que una opción *Calipso* debe cumplir para que un paquete pueda seguir su trayecto se realizan en dos momentos diferentes durante el paso del paquete por la capa de *IPv6*. El diagrama de la figura 1.9.14 sitúa las funciones llamadas para esta labor en las interfaces de red de tres nodos que transmiten un paquete desde el nodo origen hacia el nodo destino:

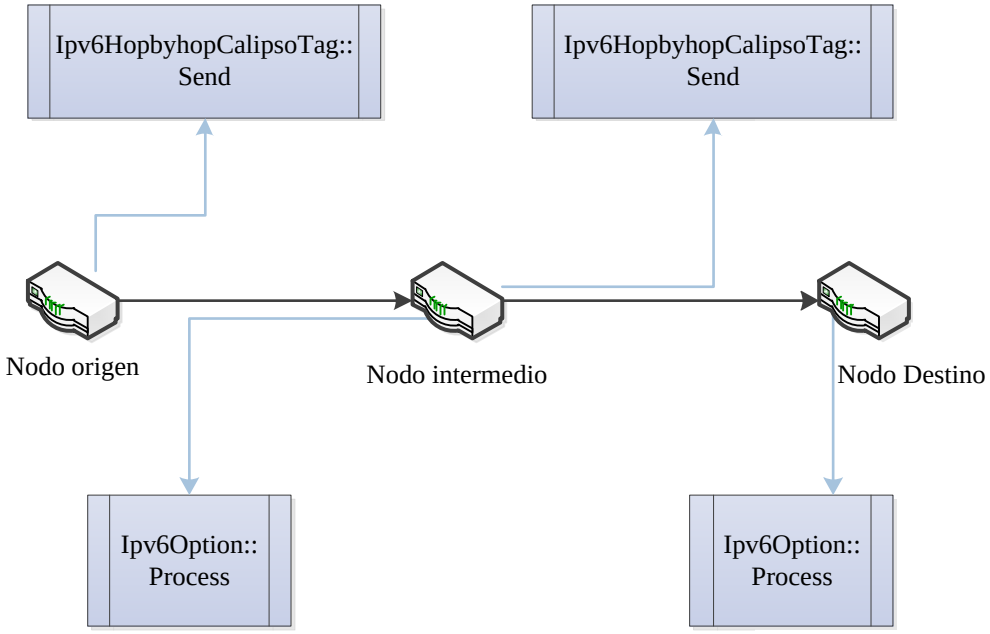

Figura 1.9.14 – Diagrama de llamadas a funciones en procesamiento de Calipso

A continuación se describe cómo funciona el procesamiento de la opción *Calipso* tanto en una interfaz de entrada como de salida.

En una interfaz de entrada, es decir, al recibir un nodo un paquete con la cabecera de extensión *Hop by Hop* y la opción *Calipso*, se desarrolla el proceso de la figura 1.9.15 en la capa de *IPv6*:

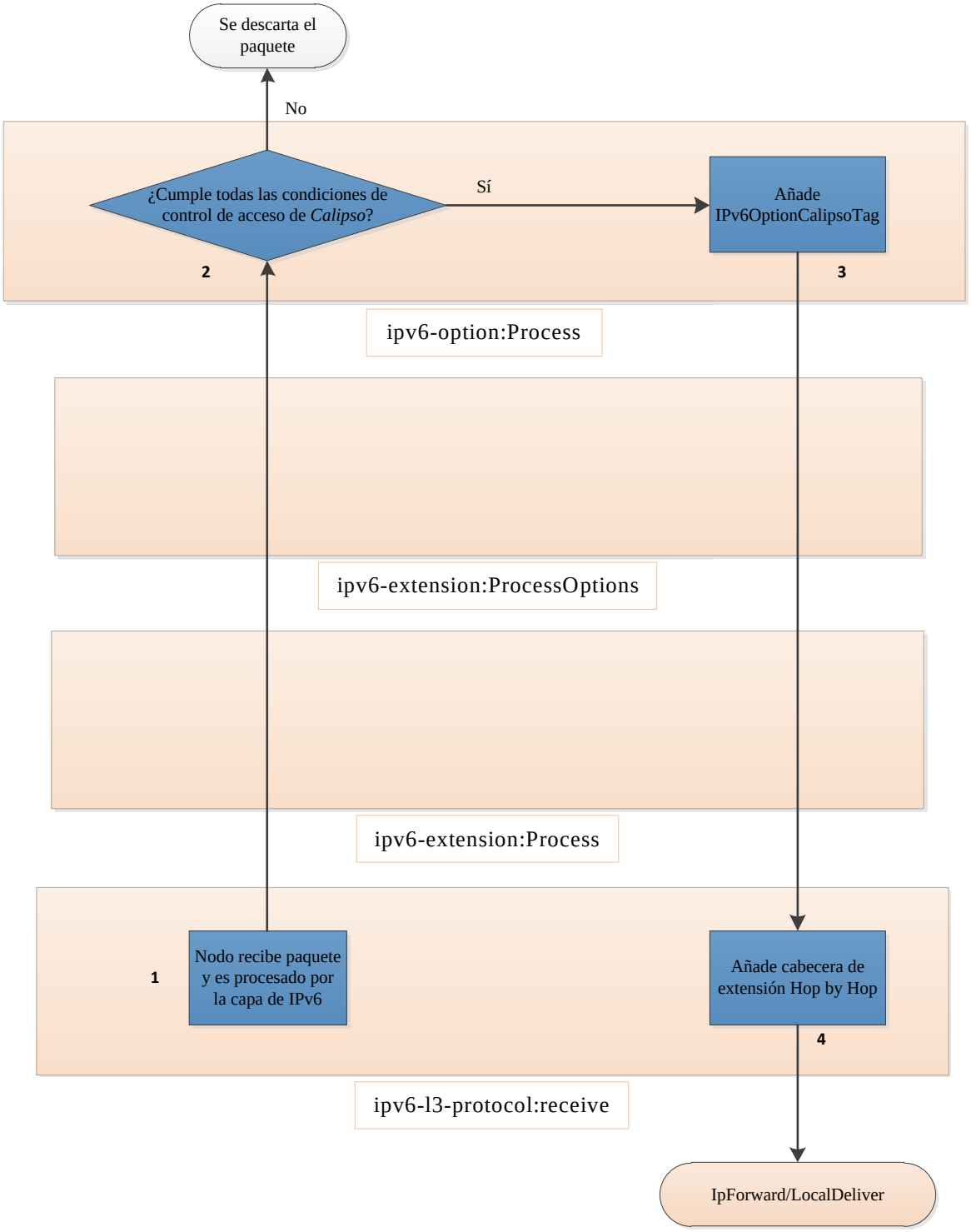

Figura 1.9.15 – Procesamiento de la opción *Calipso* al recibir un paquete

Los pasos marcados en la figura 1.9.15 se describen en las siguientes líneas:

1. El método *ipv6-l3-protocol:Receive* recibe un paquete con una opción *Calipso*. Tras ello, *ipv6-l3-protocol:Receive* realiza la llamada a *ipv6-extension:Process*

para procesar la extensión *Hop by Hop*. Éste último realiza una llamada al método *ipv6-extension:ProcessOptions*, el cual examina la cabecera de extensión en busca de opciones de *IPv6* y encuentra la opción *Calipso*.

2. Al encontrar una opción *Calipso* en el paquete, el anterior método *ipv6-extension:ProcessOptions* realiza una llamada al método I*pv6OptionCalipso:Process*. Es aquí donde se comprueban las cinco condiciones que el paquete debe cumplir para poder seguir su ruta. En caso de no cumplir alguna de las condiciones, el paquete es descartado.

3. Se añade un nuevo *Ipv6OptionCalipsoTag* para el futuro añadido de la opción al paquete. Este *Tag* contiene como parámetros los mismos valores que el paquete procesado, ya que esta opción no cambia en ruta.

4. Se elimina la cabecera de extensión *Hop by Hop* del paquete y se procede a la búsqueda de *Hop by Hop Tags* en éste para añadir una nueva cabecera de extensión. Con esto, el procesamiento de la cabecera de extensión *Hop by Hop* en la interfaz de entrada del nodo ha finalizado y el paquete continúa su camino hacia capas superiores en el nodo o hacia su destino a través de otros nodos.

El control de acceso de *Calipso* debe realizarse también en las interfaces de salida de los nodos antes de enviar un paquete hacia un nodo distinto. Para ello se evita el uso del método *ipv6-option::Process* como en el caso anterior, el cual requeriría innecesariamente examinar la cabecera de extensión *Hop by Hop* en su totalidad en busca de la opción *Calipso*. En su lugar, se utiliza el método *Ipv6OptionCalipsoTag::Send*, el cual comprueba los mismos cinco requisitos del control de acceso de *Calipso* que se producen en la recepción.

El diagrama de la figura 1.9.16 muestra el proceso a seguir en el envío de un paquete con la opción *Calipso*:

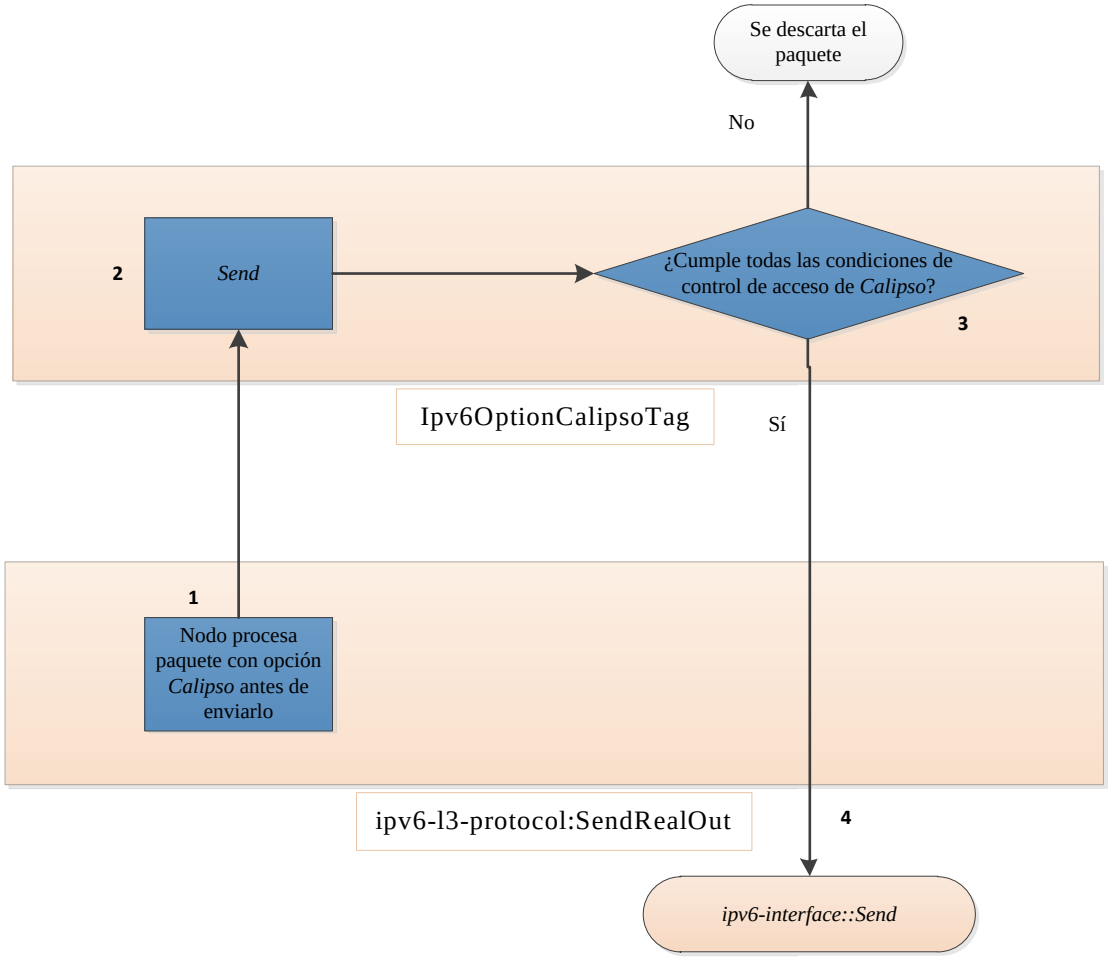

Figura 1.9.16 – Procesamiento de la opción *Calipso* al enviar un paquete

Los puntos marcados en la figura 1.9.16 se detallan a continuación:

1. El método *ipv6-l3-protocol:SendRealOut* recibe un paquete para ser enviado y comprueba la existencia de un *Hop by Hop Tag* de la opción *Calipso* dentro del paquete.

2. Si existe este *Tag*, se utilizará el método *Ipv6OptionCalipsoTag::Send* para realizar el control de acceso de *Calipso*. Este procesamiento no examina la cabecera del paquete para encontrar los valores *Calipso* contenidos en el paquete, ya que puede obtenerlos directamente del *Hop by Hop Tag*, el cual contiene estos mismos valores almacenados.

3. Si no cumple los cinco requisitos del control de acceso, el paquete es descartado y no se realizará su envío.

4. Si los cumple, se continúa con el proceso habitual de envío del paquete.

1.9.6 Programación de la opción Quick Start

La opción *Quick Start* no se encuentra implementada en el simulador *NS3*, por lo que hay que partir de cero en este caso.

Mientras lo ideal habría sido integrar la opción *Quick Start* con la implementación de TCP en *IPv6*, esta idea ha sido descartada, pues dicha implementación no está completa y el objetivo de este proyecto no es el de programar el protocolo *TCP* para *IPv6*.

Sí se ha procedido, en cambio, a programar las siguientes características sobre *Quick Start*:

- Añadido de los métodos de creación de la cabecera de la opción *Quick Start* en el módulo *ipv6-option-header*.
- Añadido de los métodos de procesamiento en el módulo *ipv6-option* para la opción *Quick Start*.
- Creación de un *Hop by Hop* Tag para Quick Start llamado *Ipv6OptionQuickStartTag*.
- Programación de la reducción del campo *QSTTL* del paquete en cada paso por un nodo.
- Añadido de compatibilidad de la opción con el sistema de procesamiento de *Hop by Hop Tags*.
- Creación de un mensaje informativo para el *LOG* de *NS3* avisando del procesamiento de la opción en el nodo donde se recibe el paquete.

Cabecera de la opción Quick Start

Para la creación de cabeceras con la opción *Quick Start* ha sido añadida una clase nueva al módulo *ipv6-option-header*. Esta clase ha sido llamada *Ipv6QuickStartHeader*, la cual implementa los métodos de la clase *Ipv6OptionHeader* y añade los métodos y variables privadas utilizados para la asignación de cada campo de la opción *Quick Start*. La figura 1.9.17 representa esta clase:

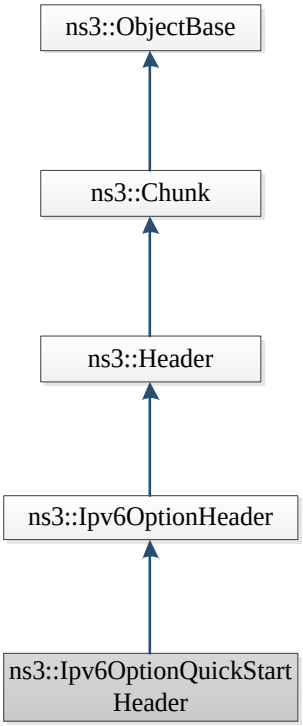

Figura 1.9.17 – Clase Ipv6OptionQuickStartHeader

Hop by Hop Tag de la opción Quick Start

El *Hop by Hop Tag* programado para la opción *Quick Start* es llamado *Ipv6OptionQuickStartTag* y tiene los mismos métodos que el *Hop by Hop Tag* genérico puesto como ejemplo en el apartado 1.9.2 de la Memoria. Se muestra a continuación en la figura 1.9.18:

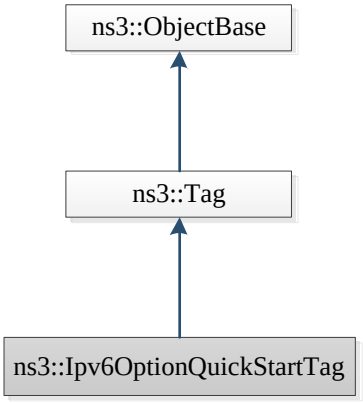

Figura 1.9.18 – Clase Ipv6OptionQuickStartTag

Procesamiento de la opción Quick Start

En cuanto al procesamiento de la opción *Quick Start* al ser recibido en un nodo, el diagrama de la figura 1.9.19 muestra el paso del paquete por la capa de *IPv6*:

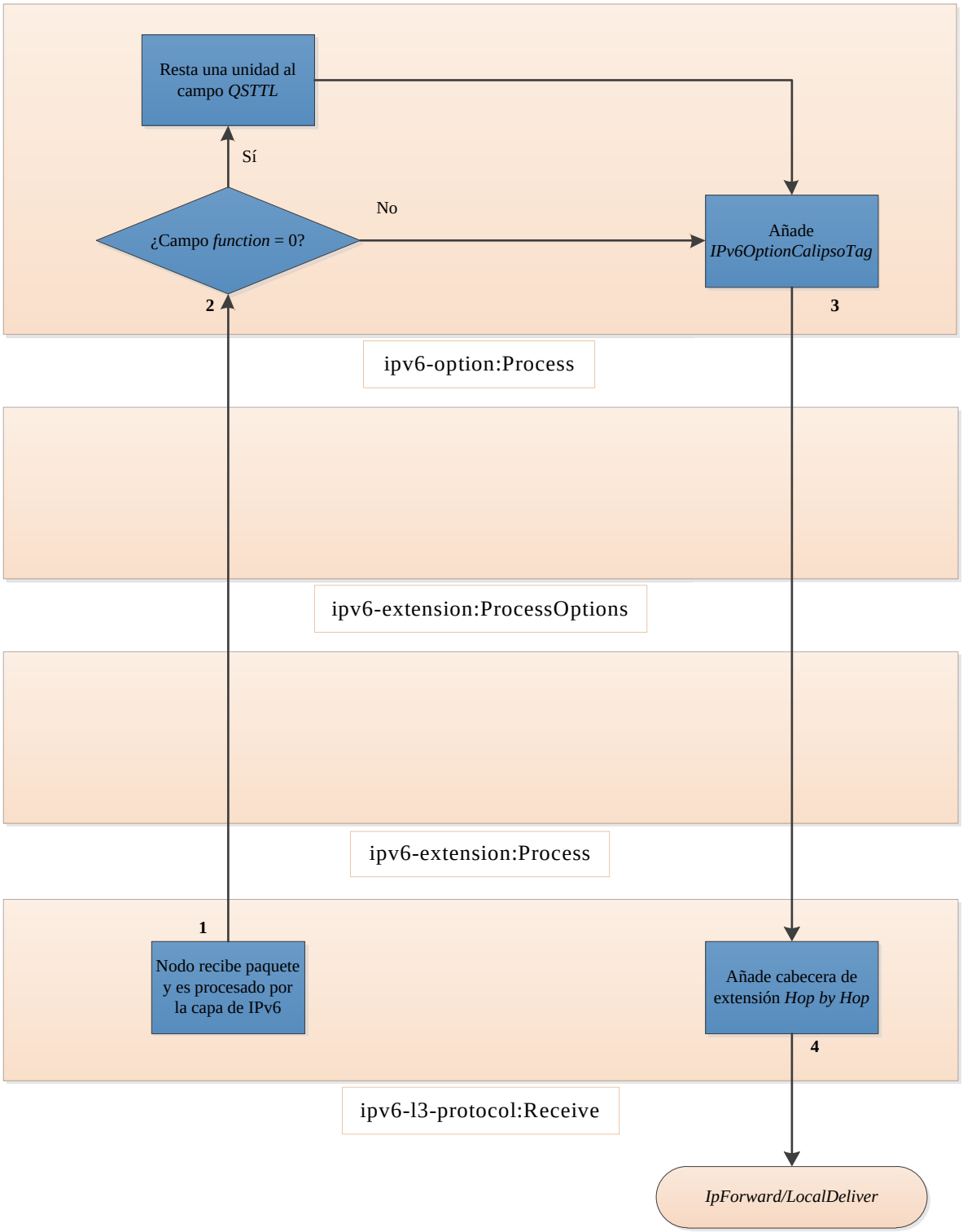

Figura 1.9.19 – Procesamiento de la opción *Quick Start* al recibir un paquete

El procesamiento de la opción *Quick Start* mostrado en la figura 1.9.19 se resume en los siguientes pasos:

1. El método *ipv6-l3-protocol:Receive* recibe un paquete con una opción *Quick Start*. Tras ello, *ipv6-l3-protocol:Receive* realiza la llamada a *ipv6-*

extension:Process para procesar la extensión *Hop by Hop*. Éste último realiza una llamada al método *ipv6-extension:ProcessOptions*, el cual examina la cabecera de extensión en busca de opciones de *IPv6* y encuentra la opción *Quick Start*.

2. Al encontrar una opción *Quick Start* en el paquete, el anterior método *ipv6-extension:ProcessOptions* realiza una llamada al método I*pv6OptionQuickStart:Process*.

3. Se añade un nuevo *Ipv6OptionCalipsoTag* para el futuro añadido de la opción al paquete. Este *Tag* contiene como parámetros los mismos valores que el paquete procesado salvo una excepción. Si la opción *Quick Start* es de tipo *Rate Request*, es decir, el campo *Function* tiene el valor 0, al campo *QSTTL* se le resta una unidad.

4. Se elimina la cabecera de extensión *Hop by Hop* del paquete y se le añade la nueva cabecera, esta vez con el campo *QSTTL* modificado. El procesamiento ha finalizado y el paquete se envía a capas superiores o se reenvía al siguiente nodo, según sea el nodo el destinatario del paquete o no.

1.9.7 Programación de los escenarios

En este proyecto se ha desarrollado una serie de escenarios que simulan topologías de red y envíos de paquetes con el objetivo de probar las nuevas características implementadas y validar el comportamiento de la extensión *Hop by Hop* y sus opciones en *NS3*.

Para cada opción se ha dispuesto de un fichero de código individual, por lo que han sido desarrollados cuatro ficheros de escenarios, uno para cada opción desarrollada:

- *routeralert.cc*
- *jumbogram.cc*
- *quickstart.cc*
- *calipso.cc*

Cada escenario define una topología de red y la configuración de sus componentes (routers, hosts, enlaces, direcciones *IP*, etc), para los cuales se programa el envío de paquetes que contienen una extensión *Hop by Hop.*

Todos estos ficheros son ejecutables y dan como resultado de su ejecución los siguientes componentes:

- La salida de consola del programa, la cual informa de los eventos y llamadas a funciones que se generan durante la ejecución del escenario.
- Capturas de tráfico con formato *pcap*, las cuales son compatibles con software de análisis de tráfico como *TCPdump* o *Wireshark.*
- Ficheros de traza con información sobre los paquetes enviados y recibidos por cada nodo durante la simulación.

Los cuatro escenarios programados siguen la misma estructura de código, formada principalmente por dos clases (figura 1.9.20):

- Método principal *main.*
- Clase de *NS3* tipo *Application.*

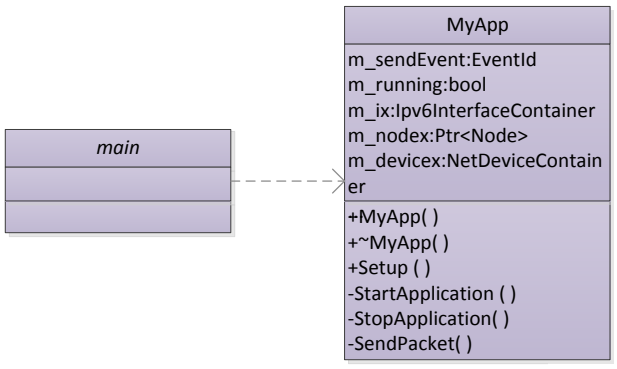

Figura 1.9.20 –Estructura de un escenario

El método principal *main* se encarga de los siguientes puntos:

- La creación de los nodos.
- La creación de los enlaces entre nodos dos a dos.
- La configuración del enlace *CSMA* por el que se simulará la transmisión y la adición de los nodos a esta red.
- La configuración e instalación del *stack* de *IPv6*.
- La configuración de las interfaces de los nodos y la asignación de una dirección IP a cada una.
- Configuración de rutas de la red de *IPv6*.
- Configuración de la aplicación que enviará el paquete y su instalación en un nodo para poder ser ejecutada durante la simulación.

Mientras el método *main* se encarga de la configuración de la topología, la creación del paquete y su envío se realiza en una clase aparte, la cual puede ser configurada desde el método *main* mediante el paso de argumentos. Esta clase se llama *MyApp* y podemos ejecutarla en el momento que queramos de la simulación, ya sea nada más empezar la simulación o con un retraso de varios segundos o minutos.

La clase *MyApp* contiene una serie de variables almacenadas que serán las que guarden los datos de los nodos e interfaces que actúan en el envío del paquete. Contiene también los siguientes métodos:

- ***MyApp*** y ~***MyApp***: Constructores del objeto de tipo A*pplication*.
- ***Setup***: Asigna a las variables privadas los valores que desde *main* se hayan configurado.
- ***StartApplicaton***: Todo el código que hay en este método es ejecutado al poner en marcha la aplicación *MyApp* en la simulación.
- ***StopApplication***: Detiene la ejecución de la aplicación *MyApp* en la simulación.
- ***SendPacket***: Este método es llamado por *StartApplication*, por lo que es ejecutado durante la simulación. Contiene el código que crea el paquete o los paquetes que

serán enviados durante la simulación y especifica por qué interfaces y con qué rutas serán enviados.

Los siguientes apartados describen los cuatro escenarios programados. Se describen las topologías utilizadas, los objetivos con los que se programa cada escenario y los paquetes que son enviados en cada uno de ellos.

La interpretación de los resultados de la ejecución de estos escenarios se muestra en el apartado *1.10* de la Memoria.

1.9.8 Programación del escenario para la opción Router Alert

Este escenario simula una situación de uso de la opción *Hop by Hop* de *IPv6 Router Alert*. *Router Alert* es utilizado por protocolos como *Multicast Listener Discovery* o *Resource Reservation Protocol*. Sin embargo, éstos no están aún implementados en *NS3* y no podemos validar la opción con ellos.

En una implementación *MLD*, el mensaje *Multicast Listener Report* debe contener una opción *IPv6 Router Alert* en el paquete, por lo que esta opción será probada, como ejemplo, con este mensaje, el cual ha sido añadido al módulo de *ICMPv6* del simulador en este proyecto y podría ser de guía en una futura implementación de *MLD*.

El siguiente esquema muestra la topología de red simulada en este escenario:

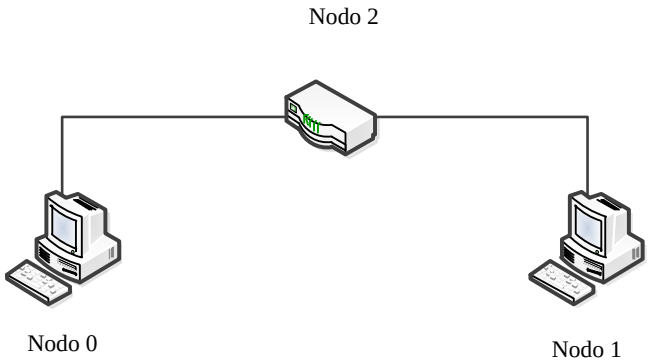

Figura 1.9.21 –Escenario *Router Alert,* esquema general

La red está formada por un router, con dos interfaces de red, y dos hosts, cada uno conectado a una de las interfaces del router. Hay un total de 2 subredes en toda la red, con dos direcciones *IPv6* asignadas a cada una.

Se muestran a continuación detalles del anterior esquema, indicando las direcciones asignadas a cada red y cada interfaz:

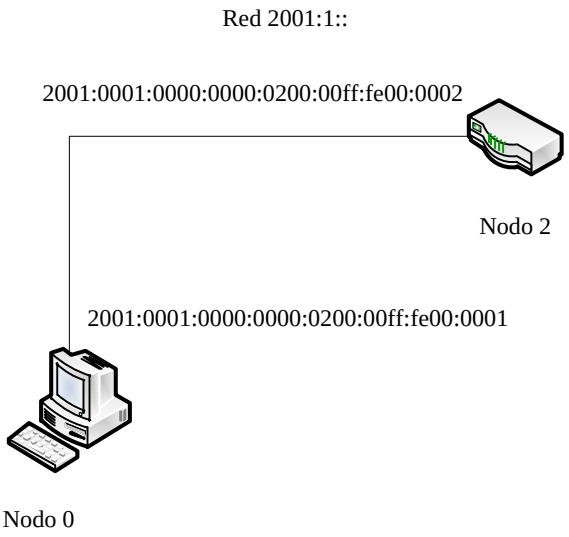

Figura 1.9.22 – Subred 2001:1::, Escenario *Router Alert*

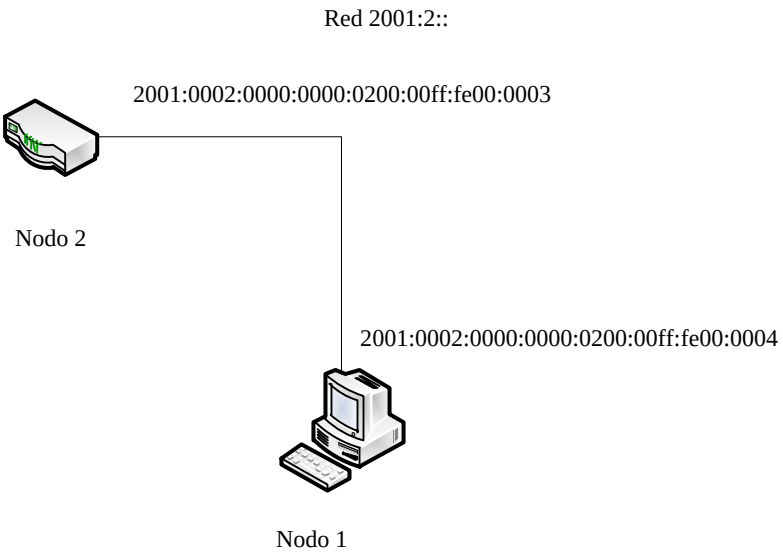

Figura 1.9.23 – Subred 2001:2::, Escenario *Router Alert*

El escenario consistirá en el envío de un paquete desde el nodo 0 con destino al nodo 1. El paquete contendrá en su interior una cabecera *Hop by Hop* y una opción *IPv6 Router Alert*. El paquete será, a nivel de transporte, un mensaje *ICMPv6 Multicast Listener Report*, perteneciente al protocolo *Multicast Listener Discovery*, aún no implementado en el simulador.

Los valores de los campos de la opción *Router Alert* vienen reflejados en la siguiente tabla:

Campo	Valor
Option Type	5
Option Length	2
***Router Alert* Value**	0, indicativo de pertenecer a un paquete *Multicast Listener Discover*

Tabla 1.9.1 – Paquete enviado en escenario Router Alert

De este escenario se espera:

- La generación correcta de la opción *Router Alert* en el paquete a partir del añadido de un *tag HopbyhopTag* en el módulo de *ICMPv6*.
- La correcta recepción y el correcto procesamiento del paquete y la opción *Router Alert* en los nodos intermedio y destino.

- El correcto alineamiento de la opción y añadido de bits de relleno en la cabecera *Hop by Hop*.

1.9.9 Programación del escenario para la opción Jumbogram

Para la opción *Jumbogram* ha sido programado un escenario en el que se pone a prueba el funcionamiento de esta opción ante cinco situaciones diferentes. Estas situaciones utilizan la misma topología en cada caso, y en ellas se produce el envío de un paquete con una opción *Jumbogram* con parámetros diferentes en cada situación. Se comprueba en cada una la reacción de los nodos ante diferentes errores de formato en la cabecera de la opción *Jumbogram*.

1.9.9.1 Paquete válido

La topología utilizada ha sido la misma usada en el escenario programado para la opción *Router Alert*, indicada en las figuras 1.9.21, 1.9.22 y 1.9.23, aunque en este caso conviene señalar la *MTU* de los enlaces:

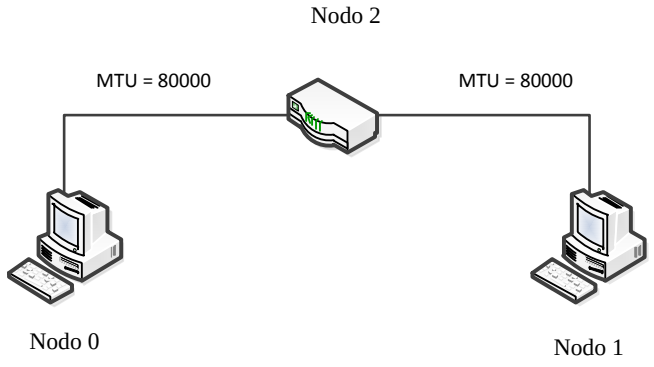

Figura 1.9.24 –Escenario *Jumbogram* (paquete válido), esquema general

La primera simulación con respecto a la opción de *Jumbogram* comprobará:

- La generación correcta de las cabeceras en el paquete.

- El correcto alineamiento de la opción dentro de la cabecera.

- La correcta recepción del paquete en los nodos intermedios y en el nodo destino.

- El añadido automático de la cabecera *Hop by Hop* a partir del T*ag Hop by Hop* diseñado en este proyecto.

En este escenario, el nodo 0 envía un paquete ICMPv6 tipo *echo* (*ping*) *request* (128) al nodo del otro extremo de la red, el nodo 1. La aplicación generadora de este paquete añadirá una cabecera de *Hop by Hop* con una opción de *Jumbogram*. El paquete deberá ser correctamente procesado por cada nodo, por lo que la cabecera de la opción *Jumbogram* no debe contener errores de formato. Por lo tanto, se partirá de que:

- El campo *payload* de la cabecera *IPv6* deberá ser 0.

- El paquete tendrá un tamaño mayor que 65,535 bytes, siendo éste de 70,056 bytes.

- La red tiene una *MTU* lo suficientemente grande como para no fragmentar el paquete, siendo ésta de tamaño 80,000.

1.9.9.2 Payload distinto distinto de 0

Se presenta aquí un escenario similar al del apartado 1.9.8.1, con la diferencia de que el campo *payload* de la cabecera de *IPv6* será distinto de 0 y deberá provocar un error en la transmisión del paquete. Se espera que:

- El paquete llegue al nodo 2 y sea procesado.

- El paquete sea descartado en este nodo.

- El nodo 2 envíe un mensaje *ICMPv6 Parameter Problem* adecuado para esta situación.

1.9.9.3 Payload menor que 65,535

Este escenario presenta la misma topología y situación que en el apartado 1.9.8.1 con la diferencia de que el paquete tendrá un tamaño de 7056 bytes, un valor menor que 65,535

La característica especial de este escenario es la del paquete teniendo un *payload* real menor a 65,535 bytes, lo cual haría innecesaria la presencia de la opción *Jumbogram* en el paquete. Se espera que:

- El paquete llegue al nodo 2 y sea procesado.
- El paquete será descartado en este nodo.
- El nodo 2 enviará un mensaje *ICMPv6 Parameter Problem* con un puntero al *payload* de la opción *Jumbogram*.

1.9.9.4 Jumbogram not present

Este escenario utiliza la misma topología de los anteriores apartados (figura 1.9.24). Como en los casos anteriores, consistirá en transmitir un mensaje *ICMPv6 request* desde el nodo 0 hasta el nodo 1.

Este último caso busca provocar el error de formato que indica la ausencia de una opción *Jumbogram* en el paquete. El paquete tiene, por tanto, las siguientes características:

- El campo *payload* de la cabecera *IPv6* es 0.
- El paquete tiene un tamaño mayor que 65,535 bytes, siendo éste de 70,056 bytes.
- La red tiene una *MTU* lo suficientemente grande como para no fragmentar el paquete, siendo ésta de tamaño 80,000.
- El paquete no contiene una extensión *Hop by Hop* y, por lo tanto, tampoco contiene una opción *Jumbogram*.

Un paquete con un *payload* superior a 65,535 y el campo de *IPv6 Payload Length* igual a 0 debe contener obligatoriamente una opción *Jumbogram* en las cabeceras de extensión. La ausencia de ésta provoca un error en el nodo que recibe tal paquete.

En este escenario se espera que:

- El paquete se reciba con éxito en el nodo 2.
- Se compruebe si el paquete contiene una opción *Jumbogram* en su interior.
- Se descarte el paquete en el nodo 2 y éste envíe un mensaje *ICMPv6 Parameter Problem* al nodo origen indicando el motivo del error.

1.9.9.5 Paquete fragmentado

Este escenario utiliza la misma topología de los casos anteriores, con la diferencia de que la MTU de los enlaces es ahora de 65,535, lo cual provoca la fragmentación de los paquetes mayores de este tamaño. El esquema general de la topología se muestra en la figura 1.9.25:

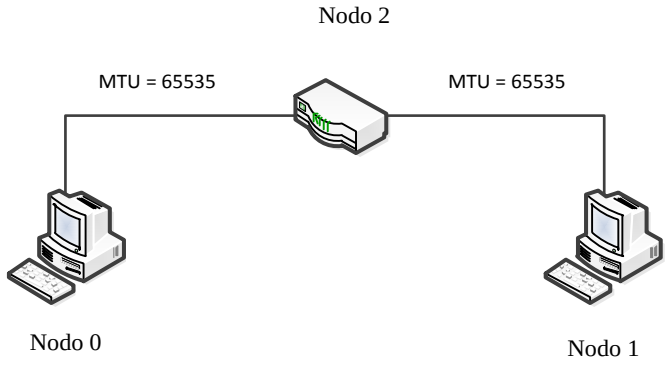

Figura 1.9.25 –Escenario *Jumbogram* (paquete fragmentado), esquema general

Al igual que en los casos anteriores, se produce el envío de un paquete *ICMPv6 Request* desde el nodo 0 con destino al nodo 1, conteniendo este paquete una cabecera de extensión *Hop by Hop* incluyendo la opción *Jumbogram* en su cabecera.

El escenario presenta las siguientes características:

- El campo *Payload Length* de la cabecera *IPv6* es 0.
- El paquete tiene un tamaño mayor que 65,535 bytes, siendo éste de 70,056 bytes, ya que se busca provocar la fragmentación del paquete.
- La *MTU* de los enlaces tiene un tamaño de 65,535.

Se espera que:

- *NS3* añada automáticamente la cabecera de extensión *Fragment* posteriormente a la cabecera de *Hop by Hop*.
- No se fragmente la cabecera *Hop by hop* y aparezca ésta en cada fragmento.
- Se reensamble con éxito el paquete a su llegada al nodo 2.
- Se procese el paquete en el nodo 2 y sea descartado al detectar la presencia simultánea de una opción *Jumbogram* y una extensión *Fragment*.

1.9.10 Programación del escenario para la opción Calipso

En el escenario mostrado en este apartado se pone a prueba el funcionamiento del sistema de procesamiento de la opción de *Ipv6 Calipso* programado en este proyecto.

La topología de red empleada en este escenario se muestra en la siguiente figura:

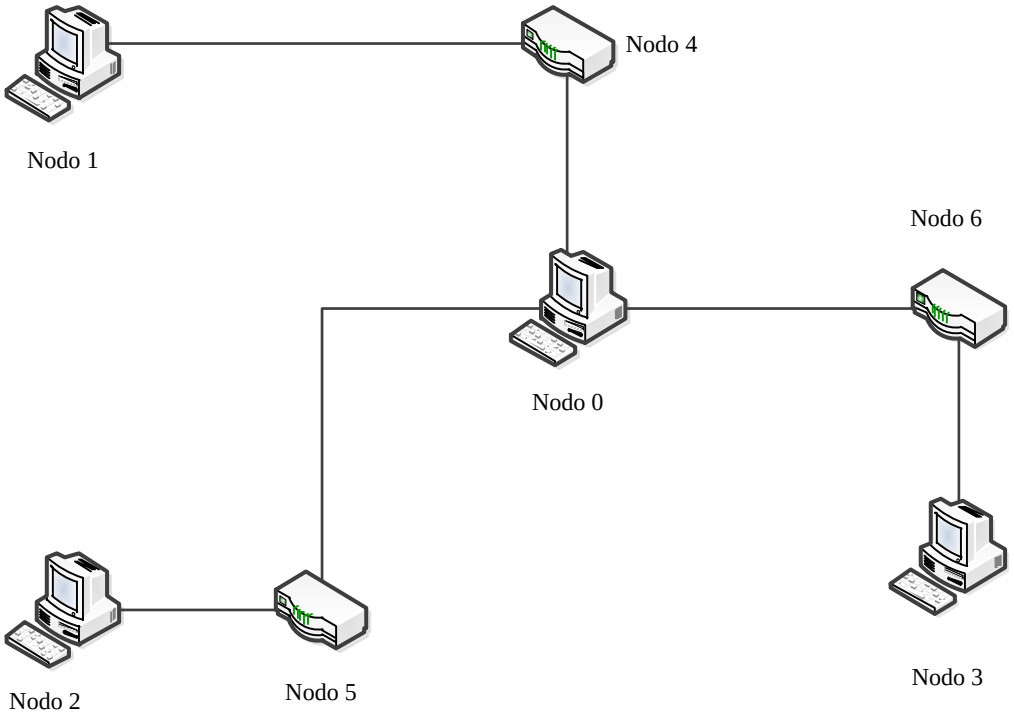

Figura 1.9.26 – Red completa, Escenario Calipso

La red está formada por siete nodos; cuatro de los cuales son hosts sin capacidad para enrutar y los otros tres son routers. Hay un total de seis subredes y doce direcciones IP asignadas entre ellas para cada interfaz de red presente en la topología. Estas direcciones se detallan en los siguientes esquemas, mostrando secciones de la figura 1.9.26:

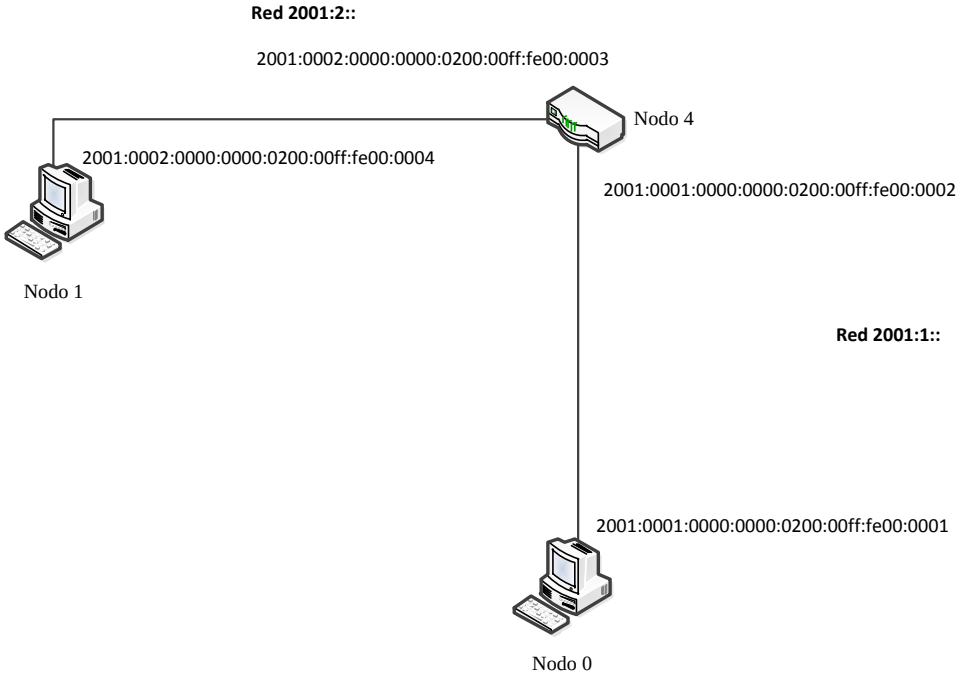

Figura 1.9.27 – Subredes 2001:1:: y 2001:2:: , Escenario Calipso

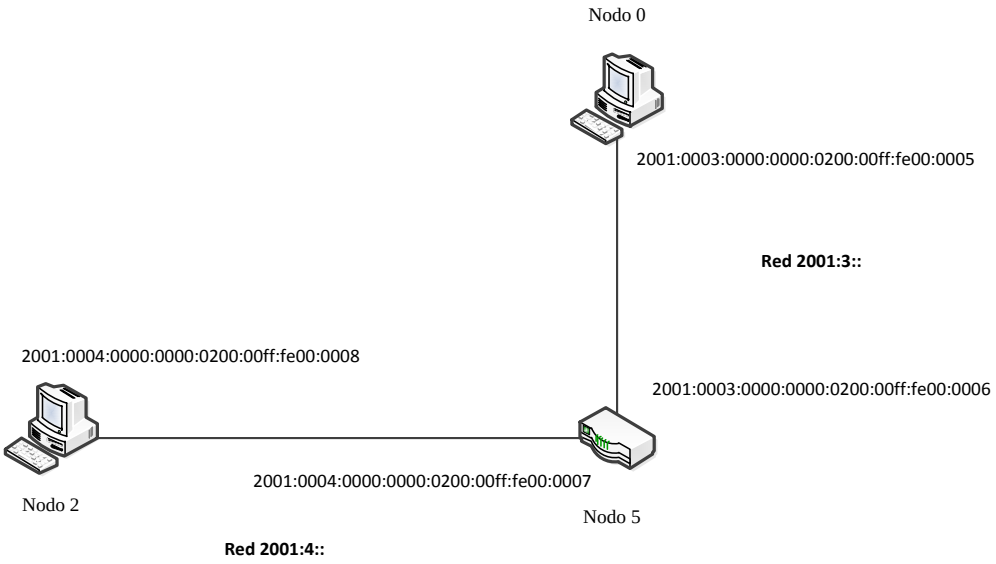

Figura 1.9.28 – Subredes 2001:3:: y 2001:4:: , Escenario Calipso

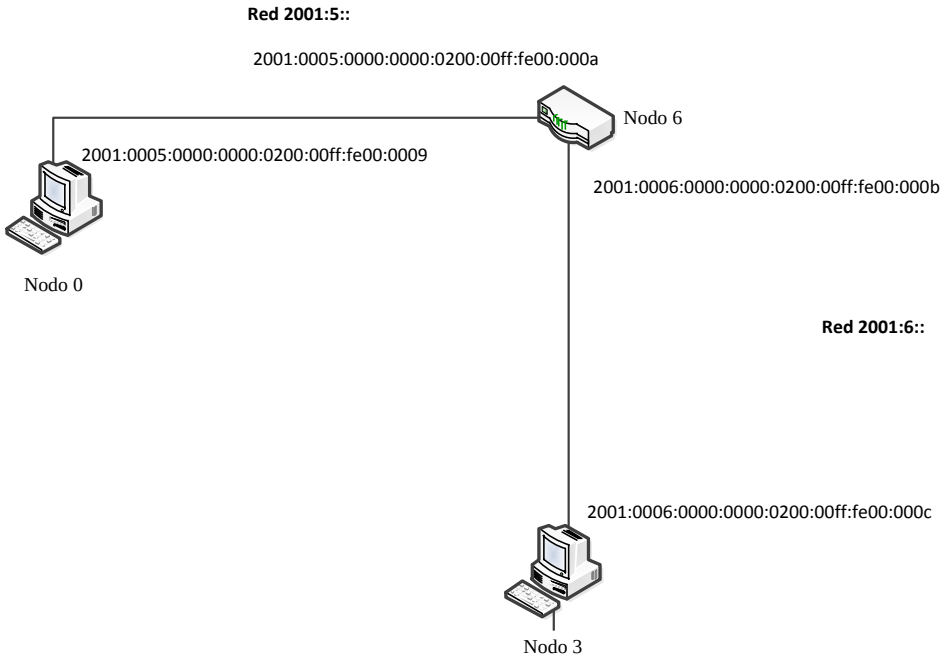

Figura 1.9.29 – Subredes 2001:5:: y 2001:6:: , Escenario Calipso

El escenario consiste en el envío de un paquete *ICMPv6 Echo Request* desde el nodo 0 a tres distintos destinos (nodos 1, 2 y 3), los cuales tendrán diferentes respuestas en su camino según la configuración que las interfaces de red dispongan sobre la opción *Calipso*.

Este paquete contiene una cabecera de extensión *Hop by Hop* con la opción *Calipso*, la cual incluye los campos de la siguiente tabla:

Campo	Valor
Length	12
Calipso Domain of Interpretation	400
Compartment Length	4
Sensitivity Level	5
Checksum	(Generado automáticamente)
Compartment Bitmap	5

Tabla 1.9.2 – Paquete enviado en escenario Calipso

El método *main* del escenario *calipso.cc* configura los parámetros de *Calipso* todas las interfaces de red presentes en la topología. Partimos, en un principio, de que todas las interfaces tienen la siguiente configuración:

- Tienen el dominio 400 como un *Dominio de Interpretación* conocido.
- Tienen el dominio 400 como un *Dominio de Interpretación* conocido.
- Los niveles mínimo y máximo para la sensibilidad permitidos en el *Dominio* 400 son 2 y 8, respectivamente.
- Los niveles mínimo y máximo para el *Compartment Bitmap* permitidos son 4 y 9, respectivamente.

El primero de los tres envíos del paquete se realiza desde el nodo 0 al nodo 1. Se espera que el paquete llegue sin problema a su destino, ya que los valores de los campos de la cabecera de la opción *Calipso* cumplen las condiciones de paso en todas las interfaces de red que encuentra su paso.

El segundo de los tres envíos tiene como origen el nodo 0 y como destino, el nodo 2, mostrados en la figura 1.9.30. Para este caso, la configuración *Calipso* de la interfaz de entrada del nodo 2 ha sufrido la siguiente modificación:

- Los niveles mínimo y máximo permitidos para la sensibilidad son ahora 7 y 8.

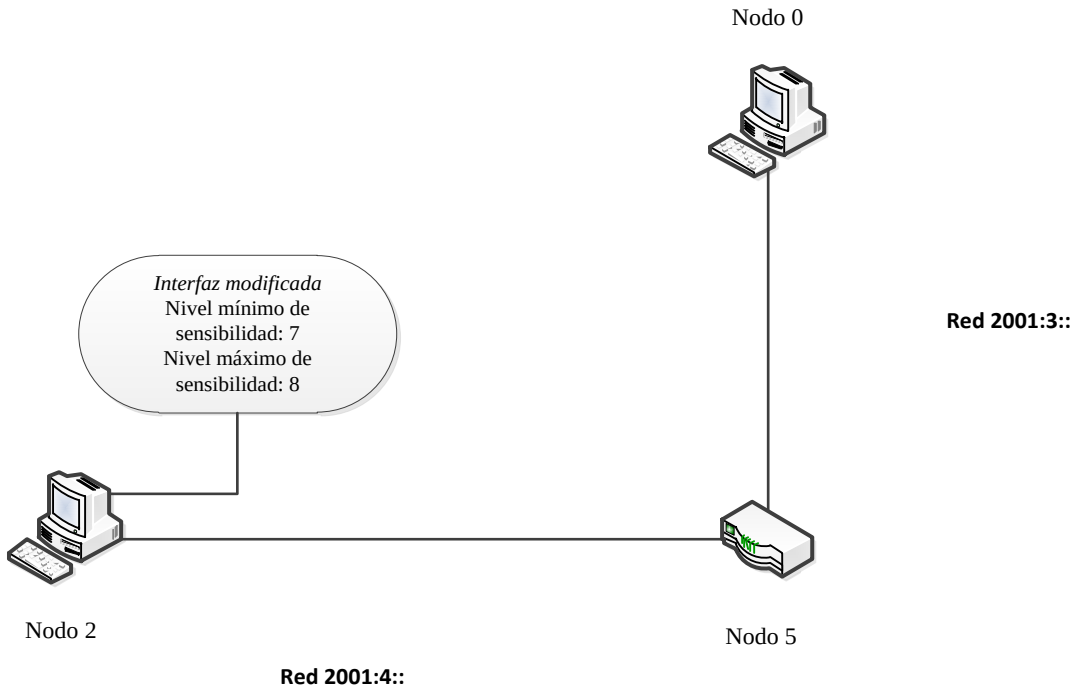

Figura 1.9.30 – Escenario Calipso, primera variación

Se espera que el paquete sea descartado al llegar al nodo 2, al tener el campo *Sensitivity Level* el valor 5, no comprendido entre los límites 7 y 8.

El tercero de los tres envíos tiene ahora como destino el nodo 3, mostrado en la figura 1.9.31, y el mismo origen que los envíos anteriores. Para este caso, se ha modificado la configuración *Calipso* de la interfaz de salida del nodo 6 con el siguiente cambio:

- El *Dominio de Interpretación* 400 no se encuentra ahora entre la lista de *Dominios de Interpretación* permitidos para esta interfaz.

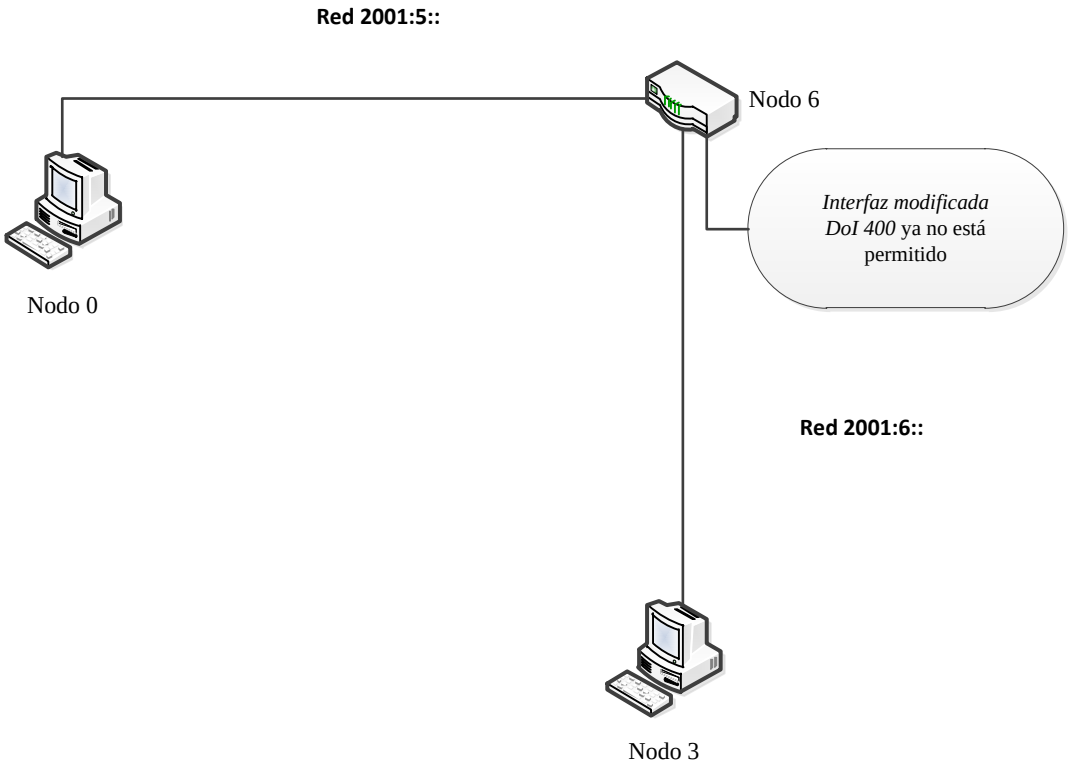

Figura 1.9.31 – Escenario Calipso, segunda variación

Se espera en este caso que el paquete sea descartado en la interfaz de salida a su paso por el nodo 6.

1.9.11 Programación del escenario para la opción Quick Start

Este apartado describe el escenario programado para poner a prueba el funcionamiento de la opción *Quick Start*. Para ello se utilizará la misma topología empleada en el apartado 1.9.8 y mostrada en las figuras 1.9.21, 1.9.22 y 1.9.23.

La opción *Quick Start* es utilizada por el protocolo *TCP* como método alternativo al *Slow Start* normalmente usado, pero debido a la falta de una completa implementación de *TCP* en la versión 6 de *IP* en *NS3*, no podemos probar la opción utilizando *TCP* de forma adecuada.

Podemos, sin embargo, validar el formato y la correcta generación de la opción en *IPv6* y aprovechar para probar la modificación de opciones *Hop by Hop* en ruta, una de las motivaciones principales de utilizar tags para la generación automática de opciones en los paquetes.

En este escenario se envía un paquete *ICMPv6 Echo Request* desde el nodo 0 con destino al nodo 1, esta vez llevando consigo una opción *Quick Start* en la cabecera de la extensión *Hop by Hop*. La siguiente tabla muestra los valores asignados a los campos de la opción *Quick Start* en el nodo origen de la transmisión:

Campo	Valor	Comentario
Length	6	Como se indica en la *RFC* de *Quick Start*.
Function	0	Necesario para que la opción sea considerada *Quick Start Request*.
Rate	2	Indiferente para el escenario.
QS TTL	8	Este campo irá siendo disminuido en una unidad en cada nodo.
QS Nonce	2	Indiferente para el escenario.
Reserved	0	No utilizado, debe ser 0.

Tabla 1.9.3 – Paquete enviado en escenario Quick Start

La opción *Quick Start* de este paquete tiene el valor 0 en el campo *Function*, haciendo que este paquete sea considerado de tipo *Quick Start Request*, lo cual conlleva que a cada paso por un nodo, el campo *QS TTL* de la cabecera sea decrementado en una unidad, como se muestra en la figura 1.9.32:

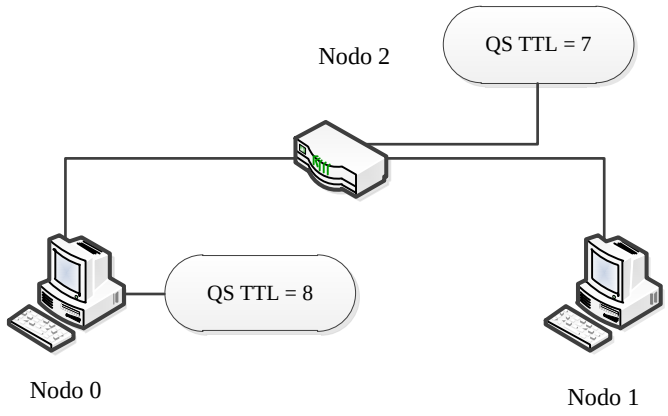

Figura 1.9.32 – Escenario *Quick Start*, modificación de opción en ruta

De esta simulación se espera, pues:

- La generación correcta de la opción *Quick Start* en el paquete enviado a partir del añadido de un *HopbyhopTag*.
- El correcto alineamiento de la opción dentro de la cabecera.
- El añadido correcto de bits de relleno junto a la opción.
- La modificación automática de la opción *Quick Start* en el procesamiento de la extensión *Hop by Hop* a su paso por cada nodo, donde el campo *QSTTL* será decrementado en una unidad.

1.10 Resultados finales

En este apartado se muestran los resultados de ejecutar las aplicaciones de los escenarios programados para cada opción *Hop by Hop*, mostrando el correcto funcionamiento de las nuevas características programadas.

Los resultados se muestran, además de con sus correspondientes explicaciones, con capturas del tráfico en interfaces de interés para cada ejemplo, la salida de la ejecución del programa y las conclusiones pertinentes.

1.10.1 Resultados del escenario Router Alert

La ejecución del escenario *routeralert.cc* da como salida lo mostrado en las figuras 1.10.1 y 1.10.2, las cuales confirman la llegada y procesamiento correctos de la opción *Router Alert* en los nodos 1 y 2:

```
Ipv6L3Protocol:Receive(0x918a3a8, 0x9189068, 0x91880d0, 34525, 02-06-00:00:00:00:00:01, 02-06-00:00:00:0
0:00:02, 0)
Packet from 02-06-00:00:00:00:00:01 received on node 2
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x918a3a8, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x9188910, 0x91cc2a0, [..] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
2 Next Header 0 Hop Limit 64 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe
00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Router Alert Option received and being processed on node 2
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x91888b0, 0x91cc2a0, [..] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
2 Next Header 0 Hop Limit 64 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe
00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Router PadN received and being processed on node 2
```

Figura 1.10.1 – Paquete con *Router Alert* recibido en nodo 2

```
Ipv6L3Protocol:Receive(0x91a2318, 0x9189a68, 0x91cba80, 34525, 02-06-00:00:00:00:00:03, 02-06-00:00:00:0
0:00:04, 0)
Packet from 02-06-00:00:00:00:00:03 received on node 1
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x91a2318, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x918b2b8, 0x9184318, [..] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
2 Next Header 0 Hop Limit 63 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe
00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Router Alert Option received and being processed on node 1
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x918b258, 0x9184318, [..] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
2 Next Header 0 Hop Limit 63 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe
00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Router PadN received and being processed on node 1
```

Figura 1.10.2 – Paquete con *Router Alert* recibido en nodo 3

La figura 1.10.3 muestra la captura de la cabecera básica de *IPv6* del paquete enviado desde el nodo 0:

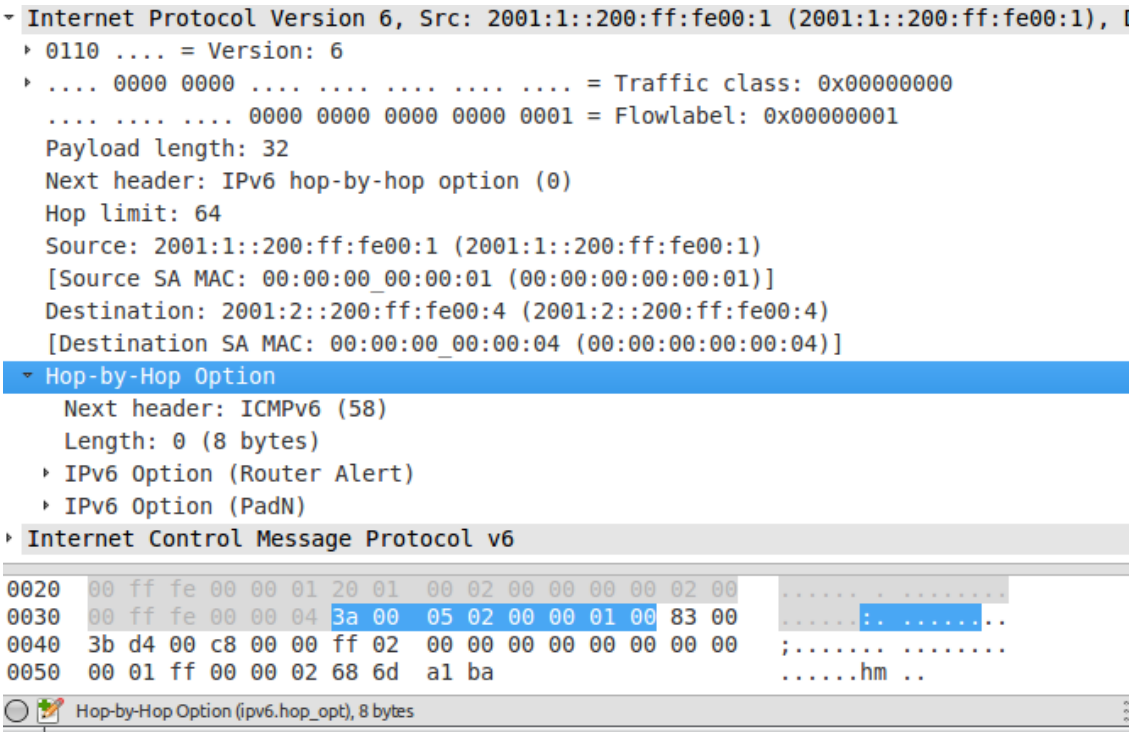

Figura 1.10.3 – Paquete con *Router Alert*, vista general (nodo 0)

La cabecera *Hop by Hop* ocupa un total de 8 bytes, cumpliendo con la norma de la *RFC 2460* sobre el formato de la cabecera de extensión. Ésta contiene el protocolo correcto en el campo *Next Header (ICMPv6)* y contiene las opciones *Router Alert* y *PadN*, esta última para conseguir ocupar ese total de 8 bytes.

La siguiente captura, mostrada en la figura 1.10.4, analiza la opción *Router Alert* contenida en el paquete:

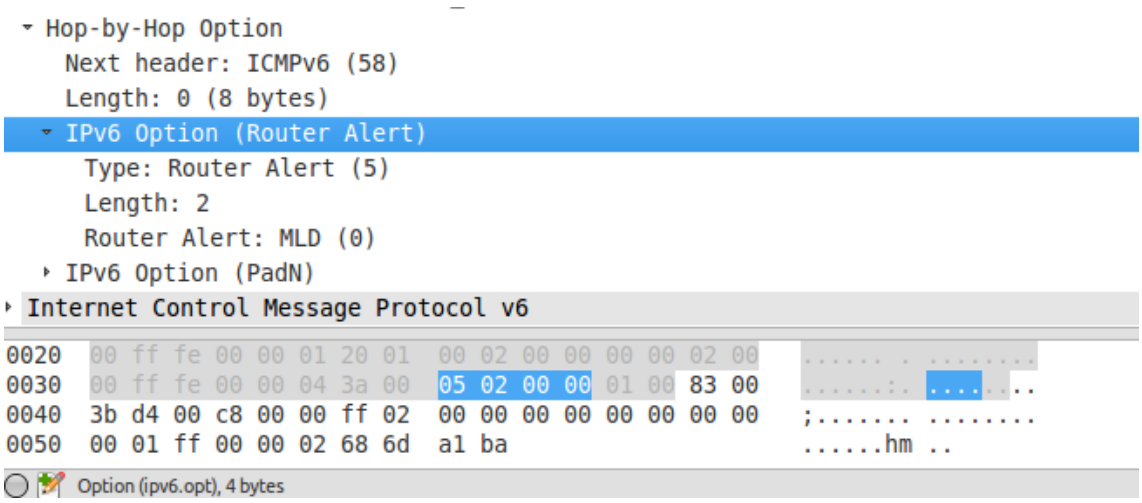

Figura 1.10.4 – Paquete con *Router Alert*, detalle *Router Alert* (nodo 0)

La opción ocupa un total de 4 bytes dentro del paquete y tiene asignado el valor 0 en el campo *Router Alert*, haciendo referencia a la presencia de un mensaje *MLD* dentro del paquete.

En la figura 1.10.5 podemos ver en detalle la opción *PadN* del paquete:

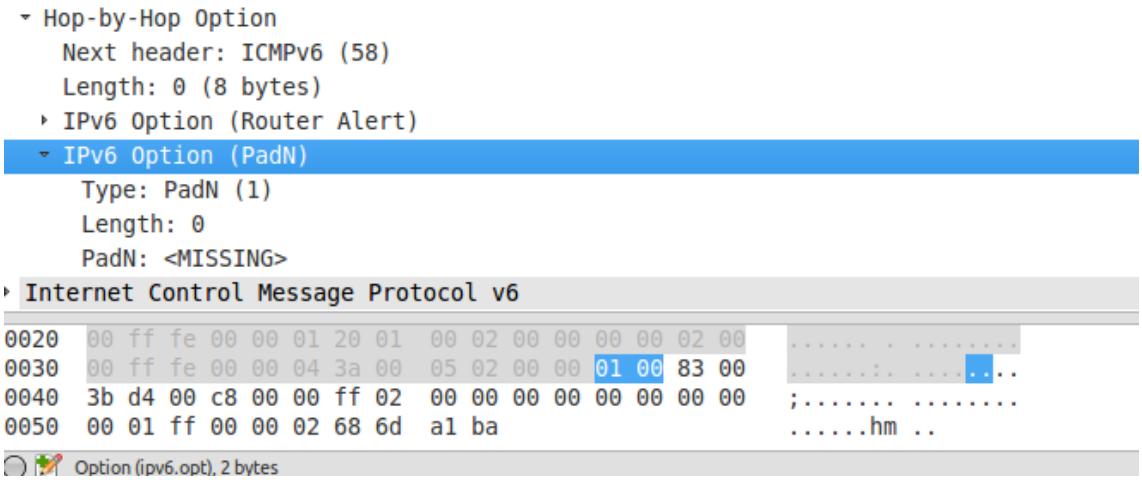

Figura 1.10.5 – Paquete con *Router Alert*, detalle *PadN* (nodo 0)

La opción ocupa un total de 2 bytes que, sumados a los 2 primeros bytes de la cabecera de extensión *Hop by Hop* y a los 4 bytes que forman la opción *Router Alert*, se llega al total de los 8 bytes necesarios para cumplir con la norma de la *RCF 2460*:

2 bytes (*Hop by Hop*) + 4 bytes (*Router Alert*) + 2 bytes (*PadN*) = 8 bytes

Por último, la siguiente captura muestra en detalle el mensaje *Multicast Listener Report*, formado gracias al código añadido en este proyecto para ser utilizado en este escenario:

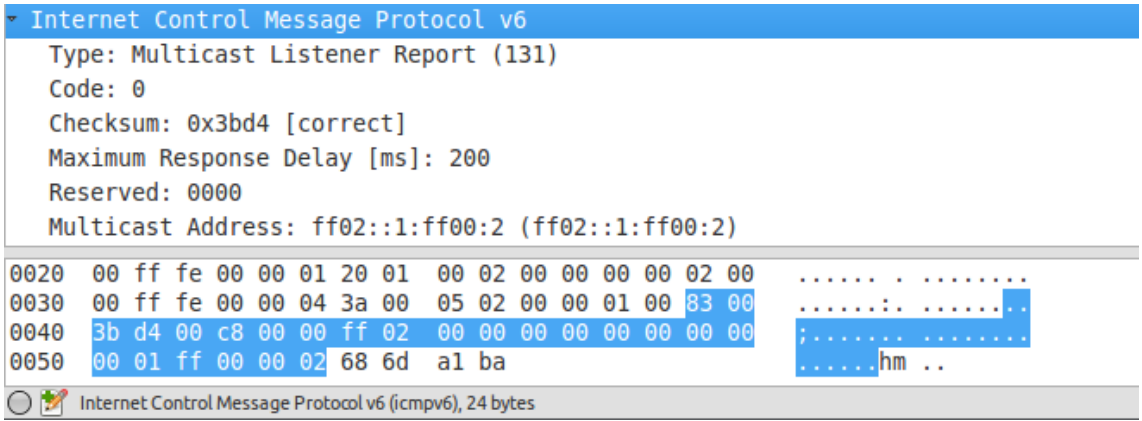

Figura 1.10.6 – Paquete con *Router Alert,* detalle Multicast Listener Report (nodo 0)

1.10.2 Resultados del escenario Jumbogram

Para la opción *Jumbogram* han sido realizadas cinco ejecuciones del programa, en las que, utilizando en todas la misma topología de red, en cada una el envío del paquete varía ligeramente. Se subdivide entonces este apartado en cinco subapartados que vienen a continuación.

1.10.2.1 Escenario Jumbogram con paquete válido

En esta primera situación se envía un paquete que cumple todas las normas de formato de un paquete con una opción *Jumbogram*, por lo que el paquete debe llegar correctamente a su destino.

Las siguientes capturas muestran puntos clave de la salida del sistema al ejecutar el programa:

```
Ipv6L3Protocol:Receive(0x9c3f438, 0x9c3e0f8, 0x9c813a0, 34525, 02-06-00:00:00:00:00:01, 02-06-00:00:00:0
0:00:02, 0)
Packet from 02-06-00:00:00:00:00:01 received on node 2
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9c3f438, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x9c3d958, 0x9c3adf0, 00 07 (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 0
 Next Header 0 Hop Limit 255 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe
00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Jumbogram Option received and being processed on node 2
```

Figura 1.10.7 – Paquete con *Jumbogram* recibido en nodo 2

```
Ipv6L3Protocol:Receive(0x9c57430, 0x9c3eaf8, 0x9c8a0a8, 34525, 02-06-00:00:00:00:00:03, 02-06-00:00:00:0
0:00:04, 0)
Packet from 02-06-00:00:00:00:00:03 received on node 1
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9c57430, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x9c40318, 0x9c3adf0, 00 07 (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 0
 Next Header 0 Hop Limit 254 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe
00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Jumbogram Option received and being processed on node 1
```

Figura 1.10.8 – Paquete con *Jumbogram* recibido en nodo 1

Como se esperaba, el paquete es recibido y procesado en cada nodo durante su trayecto hasta el destino sin provocar error alguno, ya que no ha sido detectado ningún error de formato en su cabecera.

A continuación se analiza con más detalle el contenido del paquete enviado:

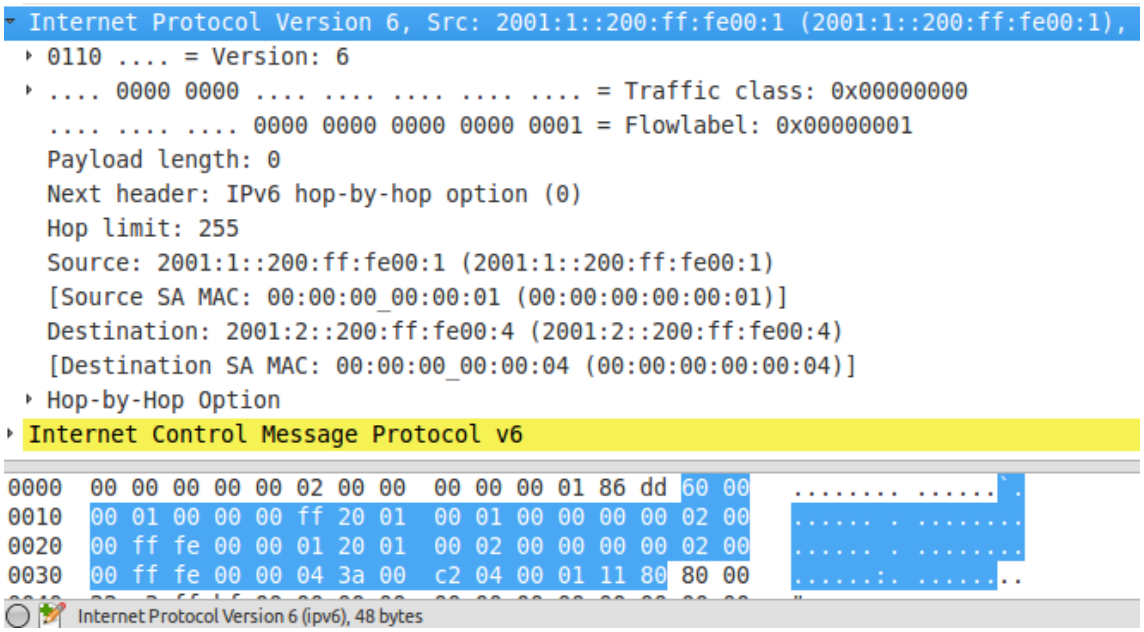

Figura 1.10.9 – Paquete con *Jumbogram*, vista general (nodo 0)

Se puede apreciar que el campo *Payload length* de *IPv6* es 0 y que hay una cabecera *Hop by Hop* en su interior. Esta cabecera se analiza con más detalle en la siguiente figura:

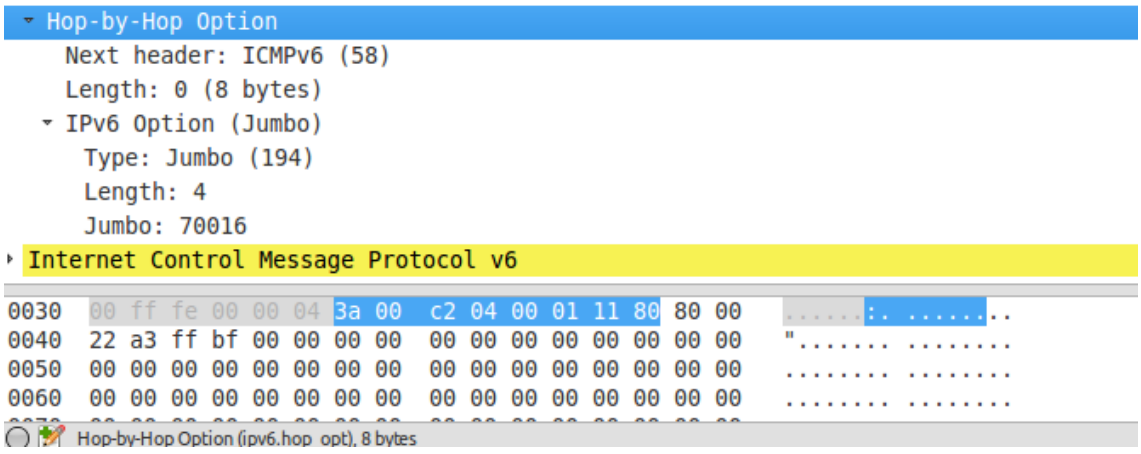

Figura 1.10.10 – Paquete con *Jumbogram*, detalle *Jumbogram* (nodo 0)

La opción *Jumbogram*, junto a los dos bytes iniciales de la extensión *Hop by Hop*, ocupan un total de 8 bytes, cumpliendo la norma sobre el formato de la cabecera de *Hop by Hop* de la *RFC 2460*. No ha hecho falta en este caso añadir relleno de bits con las opciones *PadN* o *Pad1*.

2 bytes (*Hop by Hop*) + 6 bytes (*Jumbogram*) = 8 bytes

1.10.2.2 Escenario Jumbogram con IPv6 Payload Length distinto de 0

Este es el primero de los escenarios donde se provoca un error en la transmisión del paquete con la opción *Jumbogram*, en la que el campo de la cabecera básica de *IPv6* no tiene un valor de 0, al contrario del requisito de los paquetes *Jumbogram* en el que este campo debe valer 0.

A continuación se muestra un extracto de la salida a la ejecución del escenario, en el que se confirma la llegada del paquete al nodo 2:

```
Ipv6L3Protocol:Receive(0x9c30438, 0x9c2f0f8, 0x9c723a0, 34525, 02-06-00:00:00:00:00:01, 02-06-00:00:00:0
0:00:02, 0)
Packet from 02-06-00:00:00:00:00:01 received on node 2
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9c30438, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x9c2e958, 0x9c2bdf0, [] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 4
480 Next Header 0 Hop Limit 255 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff
:fe00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Jumbogram Option received and being processed on node 2
```

Figura 1.10.11 – Paquete con *Jumbogram* recibido en nodo 2

Tras llegar el paquete al nodo 2 y reconocer la opción *Jumbogram*, se detecta el error de formato y un mensaje lo indica por consola:

```
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9c30438, 58)
IPv6 Payload Length != 0 and Jumbo Payload option present. Dropping packet
```

Figura 1.10.12 – Error de formato en *Jumbogram*

Al procesar el paquete, se detecta que el campo *Payload Length* de la cabecera de *IPv6* no es 0 y se marca el paquete para ser descartado.

El campo *Payload length* de la cabecera de *IPv6* es, en efecto, distinto de 0:

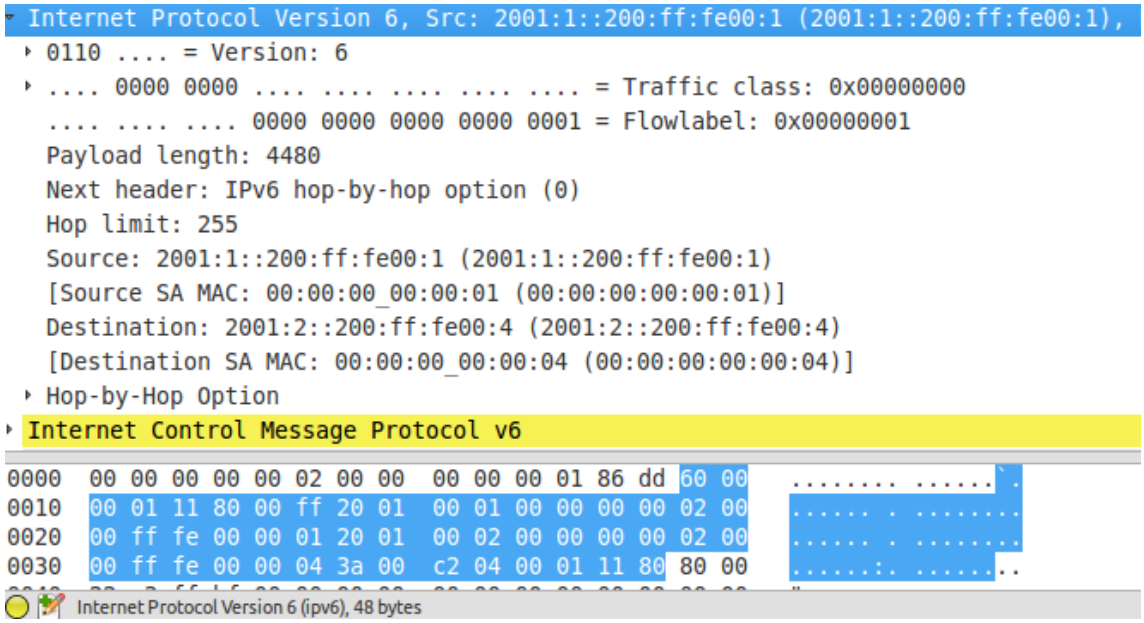

Figura 1.10.13 – Paquete con *Jumbogram*, vista general (nodo 0)

En la figura inferior se muestra una captura del paquete *ICMPv6 Parameter Problem* que el nodo 2 envía al nodo 0 para informar del descarte del paquete anterior:

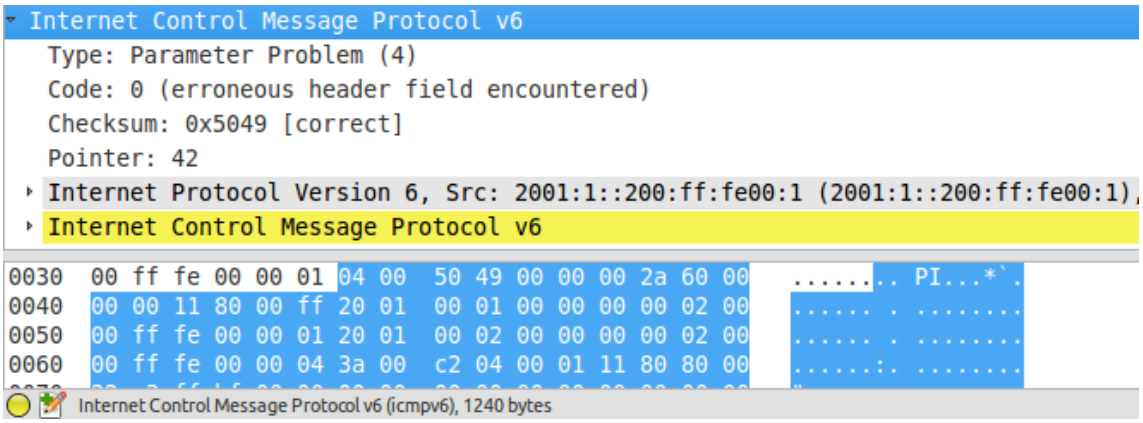

Figura 1.10.14 – Paquete con Parameter Problem (nodo 0)

El campo *pointer* tiene un valor de 42 y apunta al tipo de opción *Hop by Hop*, tal y como se indica en la *RFC 2675*.

1.10.2.3 Escenario Jumbogram con Jumbogram Payload Length menor que 65,535

En este escenario se provoca otro de los errores de formato de la cabecera de la opción *Jumbogram*.

En este caso, el paquete que se envía no es un paquete *Jumbogram*, ya que su tamaño es menor de 65,535 bytes y, por lo tanto, el valor del campo *Jumbogram Payload* es menor que 65,535 y no se corresponde con el *payload* de 32 bits de longitud que debería llevar.

Las siguientes capturas muestran la salida a la ejecución del programa, confirmando la llegada del paquete al nodo intermedio 2 reconociendo la opción *Jumbogram* en el paquete:

```
Ipv6L3Protocol:Receive(0x9f72438, 0x9f710f8, 0x9fbd0a8, 34525, 02-06-00:00:00:00:00:01, 02-06-00:00:00:0
0:00:02, 0)
Packet from 02-06-00:00:00:00:00:01 received on node 2
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9f72438, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x9f70958, 0x9fbd260, 00 02 (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 0
 Next Header 0 Hop Limit 255 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe
00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Jumbogram Option received and being processed on node 2
```

Figura 1.10.15 – Paquete con *Jumbogram* recibido en nodo 2

Al procesar la opción, se detecta el error de formato:

```
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9f72438, 58)
Jumbo Payload option present and Jumbo Payload Length = 7016< 65,536. Dropping packet
```

Figura 1.10.16 – Error de formato en *Jumbogram*

El paquete tiene un *payload* de 7016 bytes, inferior a 65,535, por lo que se marca el paquete para su posterior descarte.

La siguiente captura muestra el paquete *ICMPv6 Parameter Problem* que el nodo 2 envía al nodo origen 0 al detectar el error:

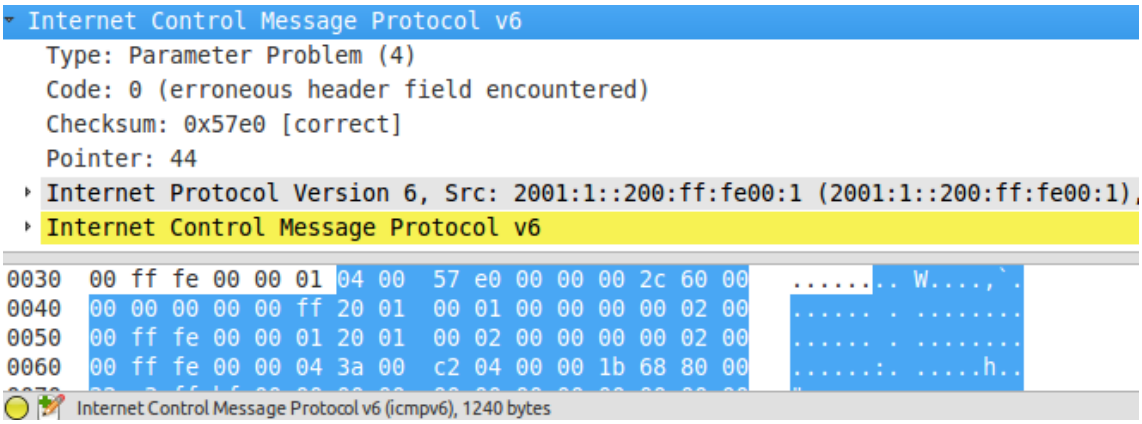

Figura 1.10.17 – Paquete con Parameter Problem (nodo 0)

El campo *pointer* del mensaje tiene un valor de 44, correspondiente al byte número 44 del paquete original, el cual hace referencia al campo *Jumbo payload* de la opción *Jumbogram*.

1.10.2.4 Escenario Jumbogram con opción Jumbogram ausente

En este escenario se envía un paquete con un *payload* mayor que 65,535 bytes desde el nodo 0 hacia el nodo 1, con la peculiaridad de que éste no lleva una cabecera de extensión *Hop by Hop* con la opción *Jumbogram* en su interior. La ausencia de la opción *Jumbogram* en este paquete provoca un error en la llegada al nodo intermedio 1, el cual detecta esta particularidad y descarta el paquete.

La salida de la ejecución del programa muestra la detección del error:

```
Ipv6L3Protocol:Receive(0x9c53438, 0x9c520f8, 0x9c94ab0, 34525, 02-06-00:00:00:00:00:01, 02-06-00:00:00:0
0:00:02, 0)
Packet from 02-06-00:00:00:00:00:01 received on node 2
IPv6 Payload Length = 0 and Jumbo Payload not present. Dropping packet
```

Figura 1.10.18 – Error de formato en *Jumbogram*

Se detecta que el campo *Payload Length* de la cabecera básica de *IPv6* es 0 y que no hay opción *Jumbogram* alguna en el paquete. Por este motivo se marca el paquete para su descarte y no llegará a su destino.

Se analiza en la siguiente figura el paquete generado en el escenario para su envío por la red:

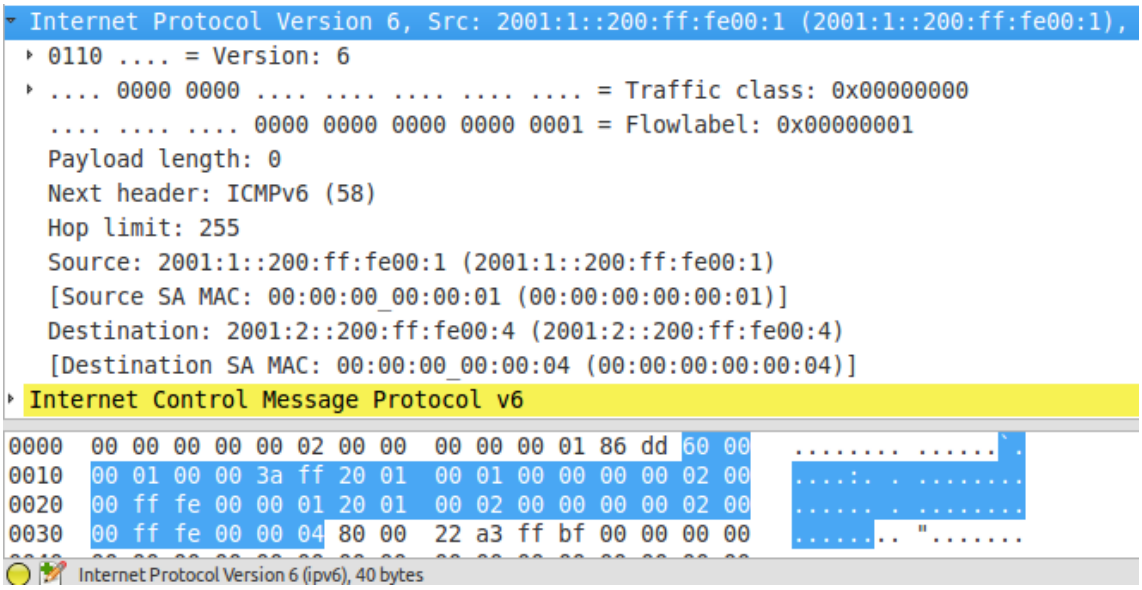

Figura 1.10.19 – Paquete sin *Jumbogram* (nodo 0)

El campo *Payload length* de la cabecera de *IPv6* tiene el valor 0 y ésta carece de la opción *Jumbogram* que debería llevar en este caso. Esto provoca que, en el siguiente nodo intermedio, se descarte el paquete y envíe de vuelta al origen el siguiente mensaje *ICMPv6 Parameter Problem*:

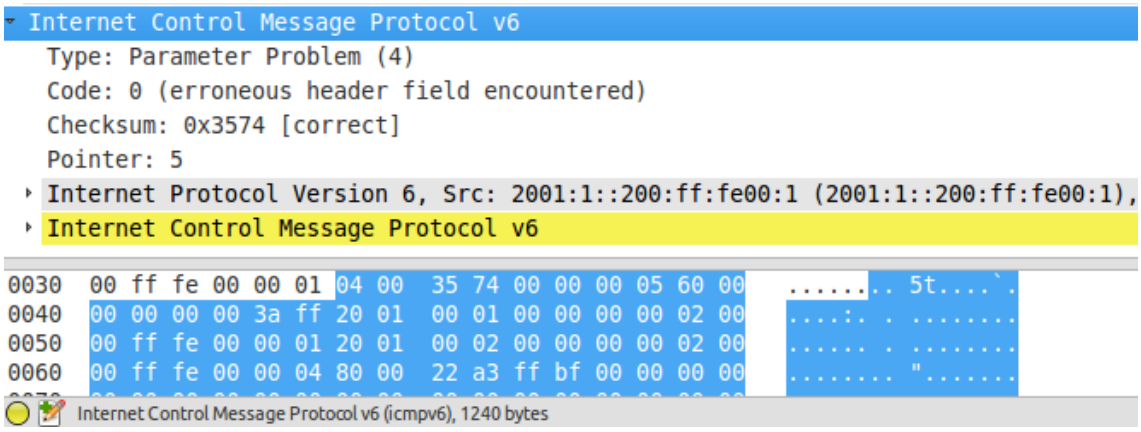

Figura 1.10.20 – Paquete con Parameter Problem (nodo 0)

El puntero del mensaje indica el quinto octeto de la cabecera *IPv6* del paquete original, señalando el valor nulo del campo *Payload Length*.

1.10.2.5 Escenario Jumbogram con paquete fragmentado

Este último escenario relativo a la opción *Jumbogram* envía un paquete fragmentado en dos partes debido a que la *MTU* del enlace es inferior al tamaño del paquete.

La siguiente imagen muestra la salida de la ejecución del escenario:

```
Ipv6L3Protocol:Receive(0x9c9a438, 0x9c990f8, 0x9ce51f8, 34525, 02-06-00:00:00:00:00:01, 02-06-00:00:00:0
0:00:02, 0)
Packet from 02-06-00:00:00:00:00:01 received on node 2
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9c9a438, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x9c98958, 0x9cdbab0, 09 02 (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 4
552 Next Header 0 Hop Limit 64 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:
fe00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Jumbogram Option received and being processed on node 2
```

Figura 1.10.21 – Paquete con *Jumbogram* recibido en nodo 2

El paquete ha llegado con éxito al nodo intermedio 2. En su procesamiento, detecta la presencia conjunta de una opción *Jumbogram* y una cabecera de extensión *Fragment*, indicando que el paquete está fragmentado.

```
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9c9a438, 58)
Jumbogram Option and Fragment Extension present. Dropping packet.
```

Figura 1.10.22 – Error de formato en *Jumbogram*

El procesador detecta la incongruencia y marca el paquete para ser descartado.

Las siguientes capturas muestran ambos fragmentos del paquete:

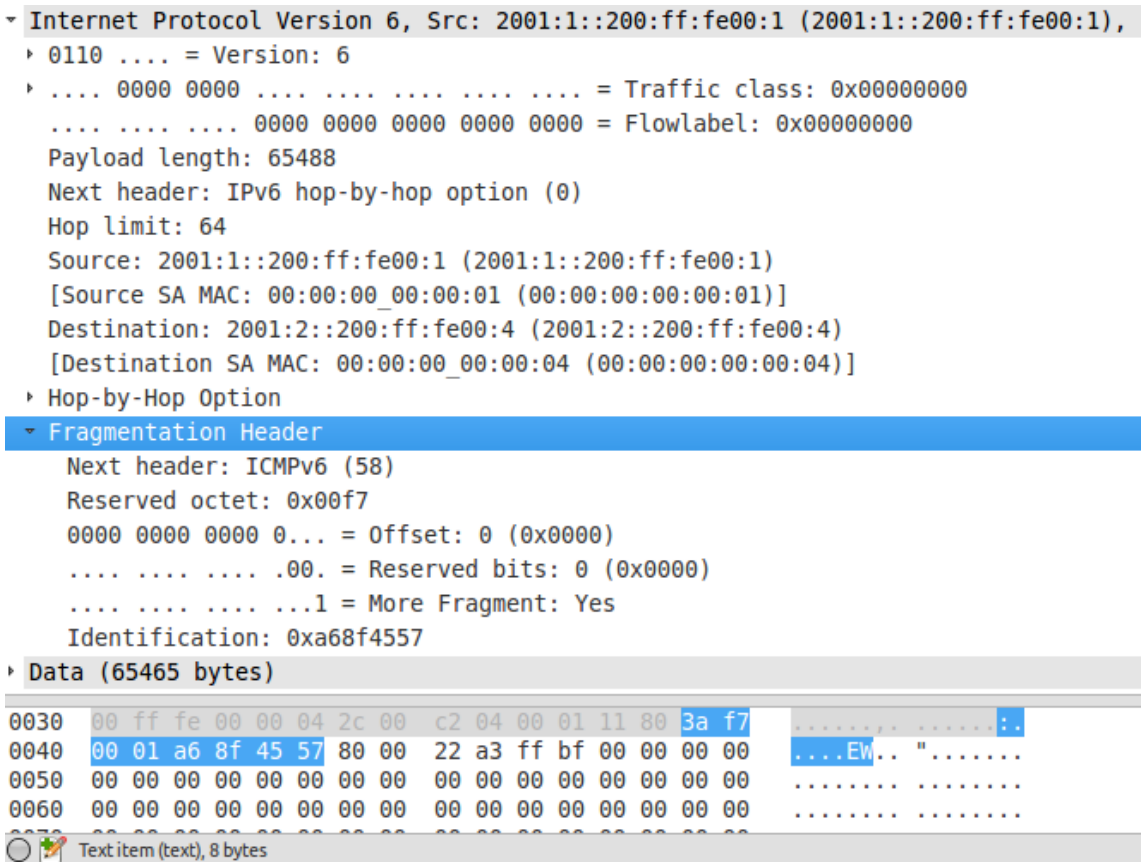

Figura 1.10.23 – Paquete con *Jumbogram* y Fragment, fragmento 1 (nodo 0)

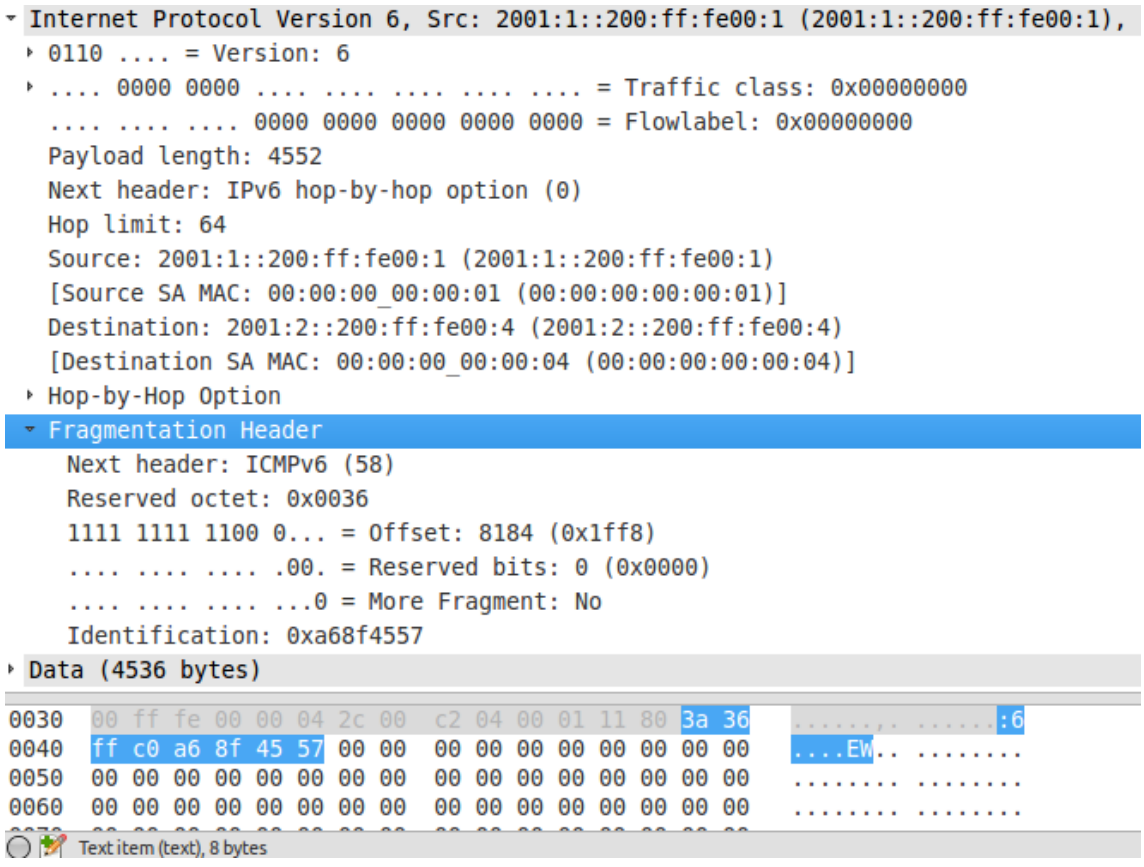

Figura 1.10.24 – Paquete con *Jumbogram* y Fragment, fragmento 2 (nodo 0)

Se puede apreciar que la extensión *Fragmentation Header* ha sido añadida automáticamente por el simulador y de forma correcta, respetando la extensión *Hop by Hop* al no fragmentarla y dejarla intacta en ambos fragmentos, tal y como se especifica en la *RFC 2460*.

La siguiente captura muestra uno de los dos mensajes *Parameter Problem* recibidos en el nodo 0:

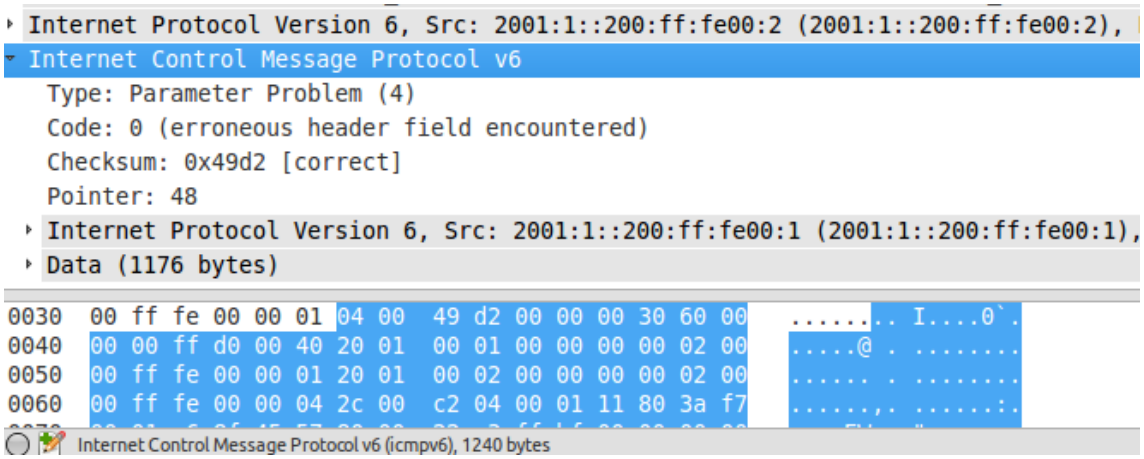

Figura 1.10.25 – Paquete con Parameter Problem (nodo 0)

Al descartar el paquete, el nodo 2 envía un mensaje *ICMPv6 Parameter Problem*, apuntando esta vez al octeto 48, haciendo referencia al inicio de la cabecera de la extensión *Fragment* del paquete original.

1.10.3 Resultados del escenario Calipso

Este apartado muestra los resultados de ejecutar el fichero *calipso.cc*, el cual comprueba el funcionamiento de la opción *Calipso* dentro de *NS3*.

En este escenario se produce el envío de un mismo paquete con la opción *Calipso* desde el mismo origen a tres distintos destinos.

En primer lugar se muestran en las figura 1.10.26 y 1.10.27 las capturas del paquete enviado generadas en la ejecución del escenario:

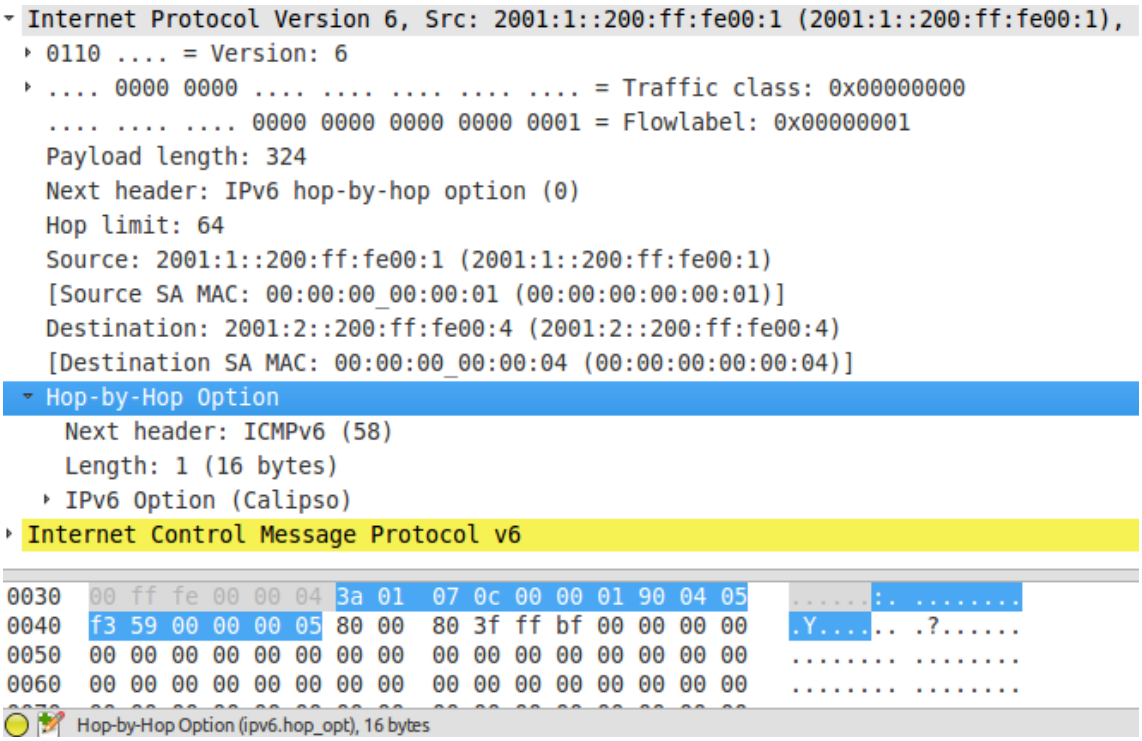

Figura 1.10.26 – Paquete con Opción *Calipso*¸ vista general (nodo 0)

Se aprecia en la figura superior que el tamaño de la cabecera de extensión *Hop by Hop* es de 16 bytes, cumpliendo el requisito de ser éste un número múltiplo de 8.

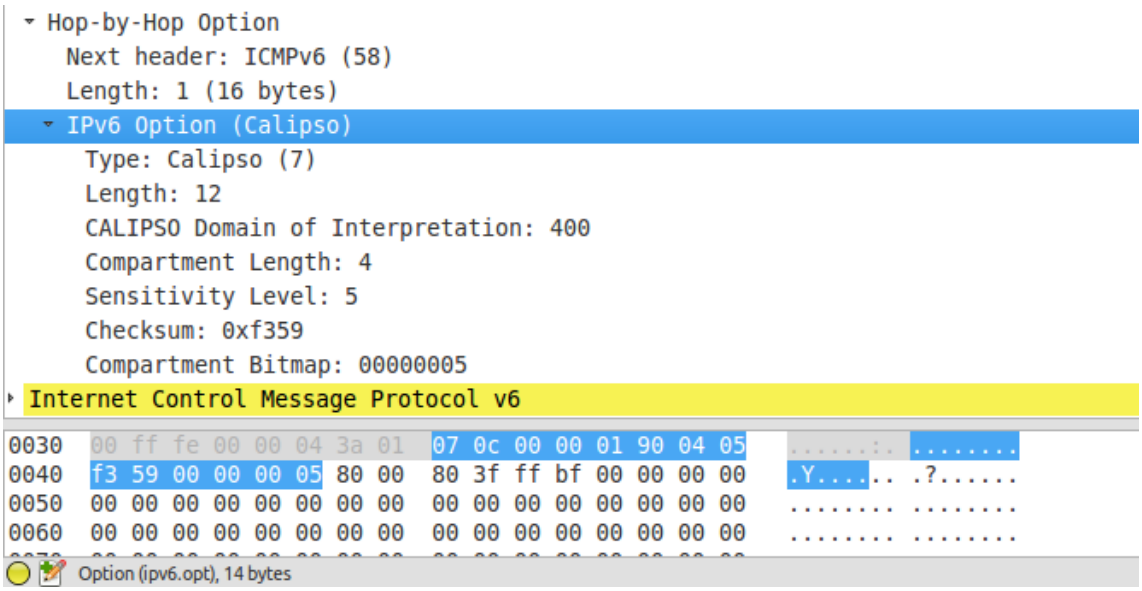

Figura 1.10.27 – Paquete con Opción *Calipso*¸ detalle de la opción *Calipso* (nodo 0)

La cabecera de la opción *Calipso* tiene un tamaño de 14 bytes, por lo que, sumados a los dos bytes de los dos primeros campos de la cabecera *Hop by Hop*, suman los 16 totales:

2 bytes (*Hop by Hop*) + 14 bytes (Calipso) = 16 bytes

A continuación se muestran las salidas de la ejecución del programa, comprobando qué paquetes llegan a su destino y cuáles son descartados, dependiendo de si cumple las reglas del control de acceso de *Calipso*.

1.10.3.1 Escenario Calipso con paquete llegando a su destino

En el primer caso, descrito en la figura 1.9.27 del apartado 1.9.10, el paquete llega correctamente a su destino, ya que pasa el control de todas las interfaces de red que se encuentra a su paso. En la figura 1.10.28 se confirma un procesamiento con éxito de la opción *Calipso* en el nodo origen 0:

```
Ipv6L3Protocol:Send(0x9f4af58, 0x9f1d410, 2001:0001:0000:0000:0200:00ff:fe00:0001, 2001:0002:0000:0000:0
200:00ff:fe00:0004, 58, 0x9f1e0a8)
Ipv6L3Protocol::Send case 1: passed in with a route
Ipv6L3Protocol:BuildHeader(0x9f4af58, 2001:0001:0000:0000:0200:00ff:fe00:0001, 2001:0002:0000:0000:0200:
00ff:fe00:0004, 0, 324, 64, 0)
Ipv6L3Protocol:SendRealOut(0x9f4af58, 0x9f1e0a8, 0x9f1d410, (Version 6 Traffic class 0x0 Flow Label 0x1
Payload Length 324 Next Header 0 Hop Limit 64 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:
0000:0200:00ff:fe00:0004)
Ipv6L3Protocol:GetInterfaceForDevice(0x9f4af58, 0x9f1adb0)
Ipv6L3Protocol:GetInterface(0x9f4af58, 1)
Send via NetDevice ifIndex 1 Ipv6InterfaceIndex 1
Processing sending package containing an Ipv6 Calipso Option on node 0
Calipso Header processed successfully. Sending package containing an Ipv6 Calipso Option on node 0
Send to gateway 2001:0001:0000:0000:0200:00ff:fe00:0002
```

Figura 1.10.28 – Paquete con opción *Calipso* saliendo de nodo 0

La figura 1.10.29 muestra una correcta recepción y procesamiento del paquete en el nodo 1:

```
Ipv6L3Protocol:Receive(0x9f206b8, 0x9f1ba30, 0x9f88808, 34525, 02-06-00:00:00:00:00:03, 02-06-00:00:00:0
0:00:04, 0)
Packet from 02-06-00:00:00:00:00:03 received on node 1
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9f206b8, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x9f1ec68, 0x9f892a0, [▦] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
24 Next Header 0 Hop Limit 63 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:f
e00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Calipso Option received and being processed on node 1
Ipv6Option:GetNode()
Ipv6L3Protocol:GetInterface(0x9f206b8, 1)
Ipv6L3Protocol:GetDoiPermittedList(0x9f206b8)
Ipv6L3Protocol:GetInterface(0x9f206b8, 1)
Ipv6L3Protocol:GetMaxLevelRange(0x9f206b8)
Ipv6L3Protocol:GetInterface(0x9f206b8, 1)
Ipv6L3Protocol:GetMinLevelRange(0x9f206b8)
Ipv6L3Protocol:GetInterface(0x9f206b8, 1)
Ipv6L3Protocol:GetMaxCompRange(0x9f206b8)
Ipv6L3Protocol:GetInterface(0x9f206b8, 1)
Ipv6L3Protocol:GetMinCompRange(0x9f206b8)
Ipv6L3Protocol:GetInterface(0x9f206b8, 1)
Ipv6Option:GetNode()
```

Figura 1.10.29 – Paquete con opción *Calipso* recibido en nodo 1

La siguiente figura 1.10.30 muestra un correcto procesamiento de la opción *Calipso* en la interfaz de salida del nodo 4:

```
Ipv6L3Protocol:IpForward(0x9f38620, 0x9f349d0, 0x9f892a0, (Version 6 Traffic class 0x0 Flow Label 0x0 Pa
yload Length 324 Next Header 0 Hop Limit 64 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:00
00:0200:00ff:fe00:0004)
Forwarding logic for node: 4
Ipv6L3Protocol:SendRealOut(0x9f38620, 0x9f349d0, 0x9f68bf8, (Version 6 Traffic class 0x0 Flow Label 0x0
Payload Length 324 Next Header 0 Hop Limit 63 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:
0000:0200:00ff:fe00:0004)
Ipv6L3Protocol:GetInterfaceForDevice(0x9f38620, 0x9f1b588)
Ipv6L3Protocol:GetInterface(0x9f38620, 2)
Send via NetDevice ifIndex 2 Ipv6InterfaceIndex 2
Processing sending package containing an Ipv6 Calipso Option on node 4
Calipso Header processed successfully. Sending package containing an Ipv6 Calipso Option on node 4
Send to destination 2001:0002:0000:0000:0200:00ff:fe00:0004
```

Figura 1.10.30 – Paquete con opción *Calipso* saliendo de nodo 4

Por último, la figura 1.10.31 muestra una correcta recepción del paquete en el nodo 4, siendo éste el nodo destino del paquete:

```
Ipv6L3Protocol:Receive(0x9f38620, 0x9f1b030, 0x9f88808, 34525, 02-06-00:00:00:00:00:01, 02-06-00:00:00:0
0:00:02, 0)
Packet from 02-06-00:00:00:00:00:01 received on node 4
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x9f38620, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x9f1fff8, 0x9f892a0, [] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
24 Next Header 0 Hop Limit 64 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:f
e00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Calipso Option received and being processed on node 4
Ipv6Option:GetNode()
Ipv6L3Protocol:GetInterface(0x9f38620, 1)
Ipv6L3Protocol:GetDoiPermittedList(0x9f38620)
Ipv6L3Protocol:GetInterface(0x9f38620, 1)
Ipv6L3Protocol:GetMaxLevelRange(0x9f38620)
Ipv6L3Protocol:GetInterface(0x9f38620, 1)
Ipv6L3Protocol:GetMinLevelRange(0x9f38620)
Ipv6L3Protocol:GetInterface(0x9f38620, 1)
Ipv6L3Protocol:GetMaxCompRange(0x9f38620)
Ipv6L3Protocol:GetInterface(0x9f38620, 1)
Ipv6L3Protocol:GetMinCompRange(0x9f38620)
Ipv6L3Protocol:GetInterface(0x9f38620, 1)
Ipv6Option:GetNode()
Calipso Header processed successfully on node4
```

Figura 1.10.31 – Paquete con opción *Calipso* recibido en nodo 1

1.10.3.2 Escenario Calipso con paquete descartado en recepción

En este segundo caso, descrito en la figura 1.9.28 del apartado 1.9.10, las interfaces están configuradas de tal forma que el procesamiento de la opción fallará en su llegada al nodo 2, tal y como se muestra en la figura 1.10.32:

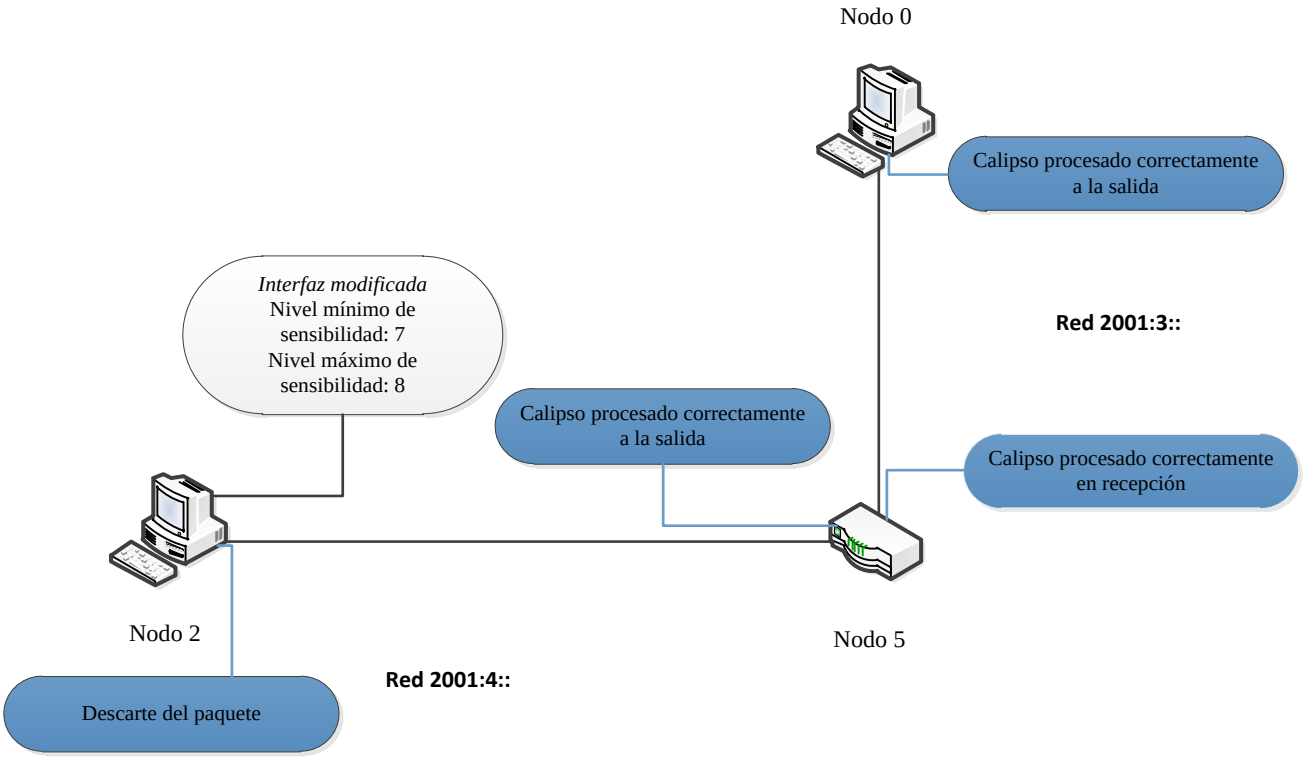

Figura 1.10.32 – Paquete con opción *Calipso* enviado desde el nodo 0 al nodo 2

En la figura 1.10.33 se confirma un procesamiento con éxito de la opción *Calipso* en el nodo origen 0:

```
Ipv6L3Protocol:Send(0x888df58, 0x88cc2a0, 2001:0003:0000:0000:0200:00ff:fe00:0005, 2001:0004:0000:0000:0
200:00ff:fe00:0008, 58, 0x88cbcf8)
Ipv6L3Protocol::Send case 1: passed in with a route
Ipv6L3Protocol:BuildHeader(0x888df58, 2001:0003:0000:0000:0200:00ff:fe00:0005, 2001:0004:0000:0000:0200:
00ff:fe00:0008, 0, 324, 64, 0)
Ipv6L3Protocol:SendRealOut(0x888df58, 0x88cbcf8, 0x88cc2a0, (Version 6 Traffic class 0x0 Flow Label 0x1
Payload Length 324 Next Header 0 Hop Limit 64 )2001:0003:0000:0000:0200:00ff:fe00:0005 > 2001:0004:0000:
0000:0200:00ff:fe00:0008)
Ipv6L3Protocol:GetInterfaceForDevice(0x888df58, 0x885efd0)
Ipv6L3Protocol:GetInterface(0x888df58, 2)
Send via NetDevice ifIndex 2 Ipv6InterfaceIndex 2
Processing sending package containing an Ipv6 Calipso Option on node 0
Calipso Header processed successfully. Sending package containing an Ipv6 Calipso Option on node 0
Send to gateway 2001:0003:0000:0000:0200:00ff:fe00:0006
```

Figura 1.10.33 – Paquete con opción *Calipso* saliendo de nodo 0

La figura 1.10.34 muestra una correcta recepción y procesamiento del paquete en el nodo 5:

```
Ipv6L3Protocol:Receive(0x8862328, 0x885f490, 0x8860410, 34525, 02-06-00:00:00:00:00:05, 02-06-00:00:00:0
0:00:06, 0)
Packet from 02-06-00:00:00:00:00:05 received on node 5
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x8862328, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x8862d38, 0x88cc3e8, [redacted] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
24 Next Header 0 Hop Limit 64 )2001:0003:0000:0000:0200:00ff:fe00:0005 > 2001:0004:0000:0000:0200:00ff:f
e00:0008, 0)
Ipv6Option:GetNode()
Ipv6 Calipso Option received and being processed on node 5
Ipv6Option:GetNode()
Ipv6L3Protocol:GetInterface(0x8862328, 1)
Ipv6L3Protocol:GetDoiPermittedList(0x8862328)
Ipv6L3Protocol:GetInterface(0x8862328, 1)
Ipv6L3Protocol:GetMaxLevelRange(0x8862328)
Ipv6L3Protocol:GetInterface(0x8862328, 1)
Ipv6L3Protocol:GetMinLevelRange(0x8862328)
Ipv6L3Protocol:GetInterface(0x8862328, 1)
Ipv6L3Protocol:GetMaxCompRange(0x8862328)
Ipv6L3Protocol:GetInterface(0x8862328, 1)
Ipv6L3Protocol:GetMinCompRange(0x8862328)
Ipv6L3Protocol:GetInterface(0x8862328, 1)
Ipv6Option:GetNode()
Calipso Header processed successfully on node5
```

Figura 1.10.34 – Paquete con opción *Calipso* recibido en nodo 5

La siguiente figura 1.10.35 muestra un correcto procesamiento de la opción *Calipso* en la interfaz de salida del nodo 5 antes de ser enviado:

```
Ipv6L3Protocol:IpForward(0x8862328, 0x88b1060, 0x88cc3e8, (Version 6 Traffic class 0x0 Flow Label 0x0 Pa
yload Length 324 Next Header 0 Hop Limit 64 )2001:0003:0000:0000:0200:00ff:fe00:0005 > 2001:0004:0000:00
00:0200:00ff:fe00:0008)
Forwarding logic for node: 5
Ipv6L3Protocol:SendRealOut(0x8862328, 0x88b1060, 0x88ab4f8, (Version 6 Traffic class 0x0 Flow Label 0x0
Payload Length 324 Next Header 0 Hop Limit 63 )2001:0003:0000:0000:0200:00ff:fe00:0005 > 2001:0004:0000:
0000:0200:00ff:fe00:0008)
Ipv6L3Protocol:GetInterfaceForDevice(0x8862328, 0x885fa40)
Ipv6L3Protocol:GetInterface(0x8862328, 2)
Send via NetDevice ifIndex 2 Ipv6InterfaceIndex 2
Processing sending package containing an Ipv6 Calipso Option on node 5
Calipso Header processed successfully. Sending package containing an Ipv6 Calipso Option on node 5
Send to destination 2001:0004:0000:0000:0200:00ff:fe00:0008
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x8862328, 58)
Ipv6L3Protocol:Send(0x8862328, 0x88cba90, 2001:0004:0000:0000:0200:00ff:fe00:0007, ff02:0000:0000:0000:0
000:0001:ff00:0008, 58, 0)
```

Figura 1.10.35 – Paquete con opción *Calipso* saliendo de nodo 5

En la figura 1.10.36, el nodo 2 recibe y procesa el paquete, pero éste no cumple una de las reglas de control de acceso de *Calipso* configuradas en la interfaz. Por este motivo se descarta el paquete:

```
Ipv6Option:Process(0x885b790, 0x88cc3e8, [illegible] (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
24 Next Header 0 Hop Limit 63 )2001:0003:0000:0000:0200:00ff:fe00:0005 > 2001:0004:0000:0000:0200:00ff:f
e00:0008, 0)
Ipv6Option:GetNode()
Ipv6 Calipso Option received and being processed on node 2
Ipv6Option:GetNode()
Ipv6L3Protocol:GetInterface(0x885ad80, 1)
Ipv6L3Protocol:GetDoiPermittedList(0x885ad80)
Ipv6L3Protocol:GetInterface(0x885ad80, 1)
Ipv6L3Protocol:GetMaxLevelRange(0x885ad80)
Ipv6L3Protocol:GetInterface(0x885ad80, 1)
Ipv6L3Protocol:GetMinLevelRange(0x885ad80)
Ipv6L3Protocol:GetInterface(0x885ad80, 1)
Ipv6L3Protocol:GetMaxCompRange(0x885ad80)
Ipv6L3Protocol:GetInterface(0x885ad80, 1)
Ipv6L3Protocol:GetMinCompRange(0x885ad80)
Ipv6L3Protocol:GetInterface(0x885ad80, 1)
Sensitivity Level is out of range. Dropping packet.
```

Figura 1.10.36 – Paquete con opción *Calipso* recibido en nodo 2

1.10.3.3 Escenario Calipso con paquete descartado en interfaz de salida

El tercer y último caso comprueba cómo un paquete con la opción *Calipso* es descartado en la interfaz de salida de un nodo al llevar el paquete en su cabecera un valor de *DOI* no permitido por la interfaz:

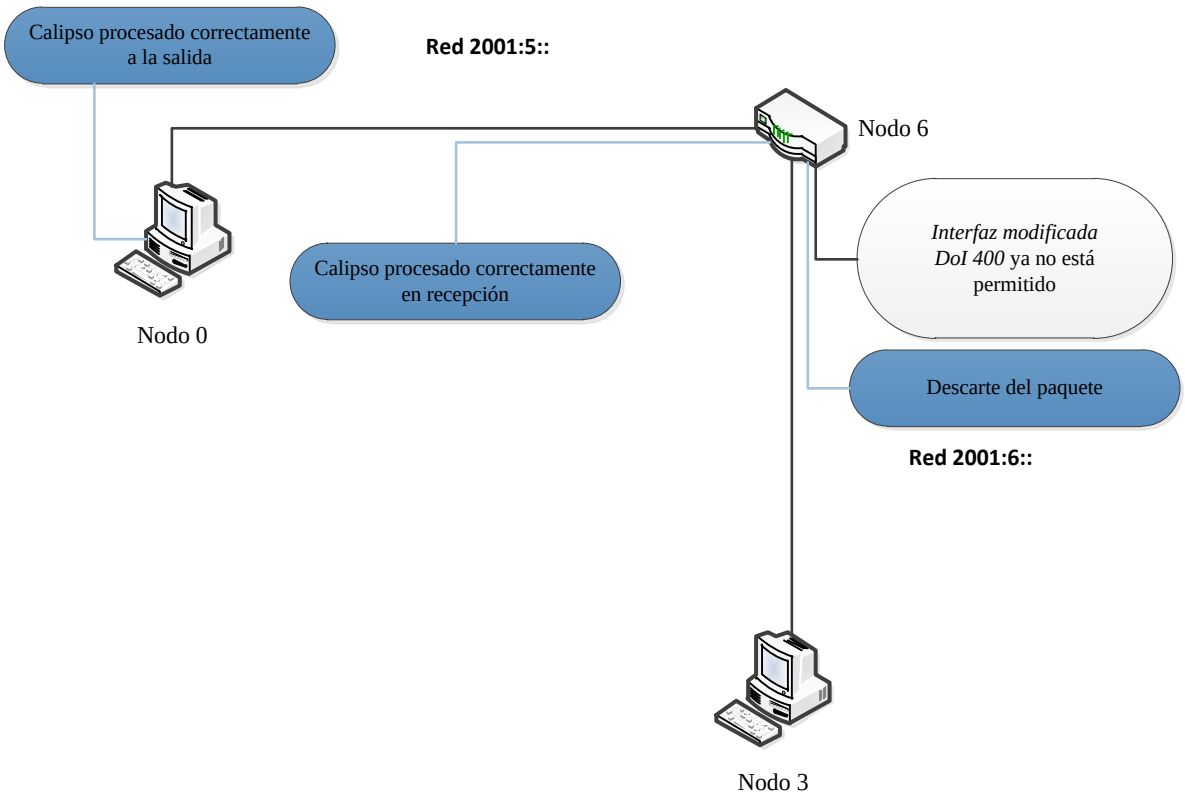

Figura 1.10.37 – Paquete con opción *Calipso* enviado desde el nodo 0 al nodo 3

En la figura 1.10.38 se confirma un procesamiento con éxito de la opción *Calipso* en el nodo origen 0:

```
Ipv6L3Protocol:Send(0x8114f58, 0x8138060, 2001:0005:0000:0000:0200:00ff:fe00:0009, 2001:0006:0000:0000:0
200:00ff:fe00:000c, 58, 0x8152c30)
Ipv6L3Protocol::Send case 1: passed in with a route
Ipv6L3Protocol:BuildHeader(0x8114f58, 2001:0005:0000:0000:0200:00ff:fe00:0009, 2001:0006:0000:0000:0200:
00ff:fe00:000c, 0, 324, 64, 0)
Ipv6L3Protocol:SendRealOut(0x8114f58, 0x8152c30, 0x8138060, (Version 6 Traffic class 0x0 Flow Label 0x1
Payload Length 324 Next Header 0 Hop Limit 64 )2001:0005:0000:0000:0200:00ff:fe00:0009 > 2001:0006:0000:
0000:0200:00ff:fe00:000c)
Ipv6L3Protocol:GetInterfaceForDevice(0x8114f58, 0x80e7458)
Ipv6L3Protocol:GetInterface(0x8114f58, 3)
Send via NetDevice ifIndex 3 Ipv6InterfaceIndex 3
Processing sending package containing an Ipv6 Calipso Option on node 0
Calipso Header processed successfully. Sending package containing an Ipv6 Calipso Option on node 0
Send to gateway 2001:0005:0000:0000:0200:00ff:fe00:000a
```

Figura 1.10.38 – Paquete con opción *Calipso* saliendo de nodo 0

La figura 1.10.39 muestra una correcta recepción y procesamiento del paquete en el nodo 6:

```
Ipv6L3Protocol:Receive(0x80e2e50, 0x80e7968, 0x8153888, 34525, 02-06-00:00:00:00:00:09, 02-06-00:00:00:0
0:00:0a, 0)
Packet from 02-06-00:00:00:00:00:09 received on node 6
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x80e2e50, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x80e3860, 0x81532a0, 80 (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 3
24 Next Header 0 Hop Limit 64 )2001:0005:0000:0000:0200:00ff:fe00:0009 > 2001:0006:0000:0000:0200:00ff:f
e00:000c, 0)
Ipv6Option:GetNode()
Ipv6 Calipso Option received and being processed on node 6
Ipv6Option:GetNode()
Ipv6L3Protocol:GetInterface(0x80e2e50, 1)
Ipv6L3Protocol:GetDoiPermittedList(0x80e2e50)
Ipv6L3Protocol:GetInterface(0x80e2e50, 1)
Ipv6L3Protocol:GetMaxLevelRange(0x80e2e50)
Ipv6L3Protocol:GetInterface(0x80e2e50, 1)
Ipv6L3Protocol:GetMinLevelRange(0x80e2e50)
Ipv6L3Protocol:GetInterface(0x80e2e50, 1)
Ipv6L3Protocol:GetMaxCompRange(0x80e2e50)
Ipv6L3Protocol:GetInterface(0x80e2e50, 1)
Ipv6L3Protocol:GetMinCompRange(0x80e2e50)
Ipv6L3Protocol:GetInterface(0x80e2e50, 1)
Ipv6Option:GetNode()
Calipso Header processed successfully on node6
```

Figura 1.10.39 – Paquete con opción *Calipso* recibido en nodo 6

La última figura 1.10.40 muestra la salida del programa en el momento en el que la interfaz de salida del nodo 6 descarta el paquete antes de enviarlo al no cumplir con los requisitos de acceso de dicha interfaz:

```
Ipv6L3Protocol:IpForward(0x91b8e50, 0x91d49d0, 0x9229290, (Version 6 Traffic class 0x0 Flow Label 0x0 Pa
yload Length 324 Next Header 0 Hop Limit 64 )2001:0005:0000:0000:0200:00ff:fe00:0009 > 2001:0006:0000:00
00:0200:00ff:fe00:000c)
Forwarding logic for node: 6
Ipv6L3Protocol:SendRealOut(0x91b8e50, 0x91d49d0, 0x9208bf8, (Version 6 Traffic class 0x0 Flow Label 0x0
Payload Length 324 Next Header 0 Hop Limit 63 )2001:0005:0000:0000:0200:00ff:fe00:0009 > 2001:0006:0000:
0000:0200:00ff:fe00:000c)
Ipv6L3Protocol:GetInterfaceForDevice(0x91b8e50, 0x91bdf00)
Ipv6L3Protocol:GetInterface(0x91b8e50, 2)
Send via NetDevice ifIndex 2 Ipv6InterfaceIndex 2
Sending package containing an Ipv6 Calipso Option on node 6
Not permitted Domain of Interpretation. Dropping packet.
```

Figura 1.10.40 – Paquete con opción *Calipso* saliendo de nodo 6

1.10.4 Resultados del escenario Quick Start

En este apartado se muestran los resultados de ejecutar el escenario para la opción *Quick Start* descrito en el apartado *1.9.6* de la Memoria.

En resumen, este escenario envía un paquete *ICMPv6 Echo Request*, conteniendo en su interior una extensión *Hop by Hop* con una opción *Quick Start*, desde el nodo 0 hacia el nodo 1 de la figura 1.10.41:

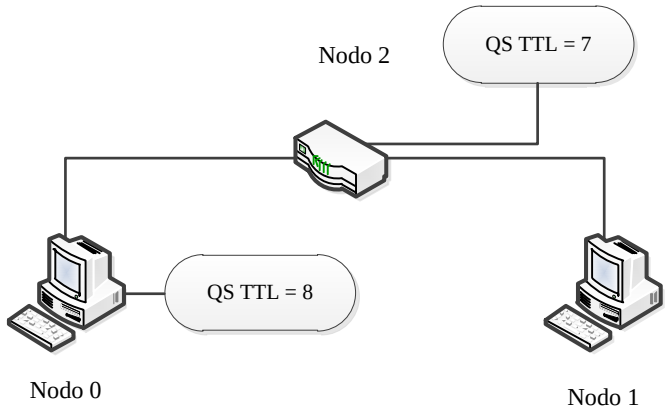

Figura 1.10.41 – Escenario *Quick Start*

El paquete parte en el origen con el campo *Quick Start TTL* de la opción *Quick Start* con un valor de 8. Este valor verá su valor reducido en una unidad a su paso por el nodo 2, por lo que llegará al nodo 1 con un valor de 7.

La figura 1.10.42 muestra un extracto de la salida del programa al ejecutar el escenario *quickstart.cc*, donde se confirma la llegada del paquete al nodo 2 y el reconocimiento y procesamiento de la opción *Quick Start*:

```
Ipv6L3Protocol:Receive(0x86b03a8, 0x86af068, 0x86abce8, 34525, 02-06-00:00:00:00:00:01, 02-06-00:00:00:0
0:00:02, 0)
Packet from 02-06-00:00:00:00:00:01 received on node 2
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x86b03a8, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x86ae940, 0x86abdb0, 00 07 (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 2
24 Next Header 0 Hop Limit 64 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:f
e00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Quick Start Option received and being processed on node 2
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x86ae8b0, 0x86abdb0,
, (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 224 Next Header 0 Hop Limit 64 )2001:0001:0
000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Router PadN received and being processed on node 2
```

Figura 1.10.42 – Paquete con Quick Start recibido en nodo 2

La figura 1.10.43 muestra la recepción del paquete en el nodo 2:

```
Ipv6L3Protocol:Receive(0x86c8318, 0x86afa68, 0x86fb158, 34525, 02-06-00:00:00:00:00:03, 02-06-00:00:00:0
0:00:04, 0)
Packet from 02-06-00:00:00:00:00:03 received on node 1
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x86c8318, 58)
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x86b12e8, 0x86f1a80, 00 07 (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 2
24 Next Header 0 Hop Limit 63 )2001:0001:0000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:f
e00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Quick Start Option received and being processed on node 1
Ipv6Option:GetOptionNumber()
Ipv6Option:GetOptionNumber()
Ipv6Option:Process(0x86b1258, 0x86f1a80,
, (Version 6 Traffic class 0x0 Flow Label 0x0 Payload Length 224 Next Header 0 Hop Limit 63 )2001:0001:0
000:0000:0200:00ff:fe00:0001 > 2001:0002:0000:0000:0200:00ff:fe00:0004, 0)
Ipv6Option:GetNode()
Ipv6 Router PadN received and being processed on node 1
```

Figura 1.10.43 – Paquete con Quick Start recibido en nodo 1 (destino)

El paquete es recibido con éxito en tambos nodos y la opción es correctamente reconocida y procesada en cada uno de ellos.

La figura 1.10.44 muestra la captura de la cabecera básica de *IPv6* del paquete enviado desde el nodo 0:

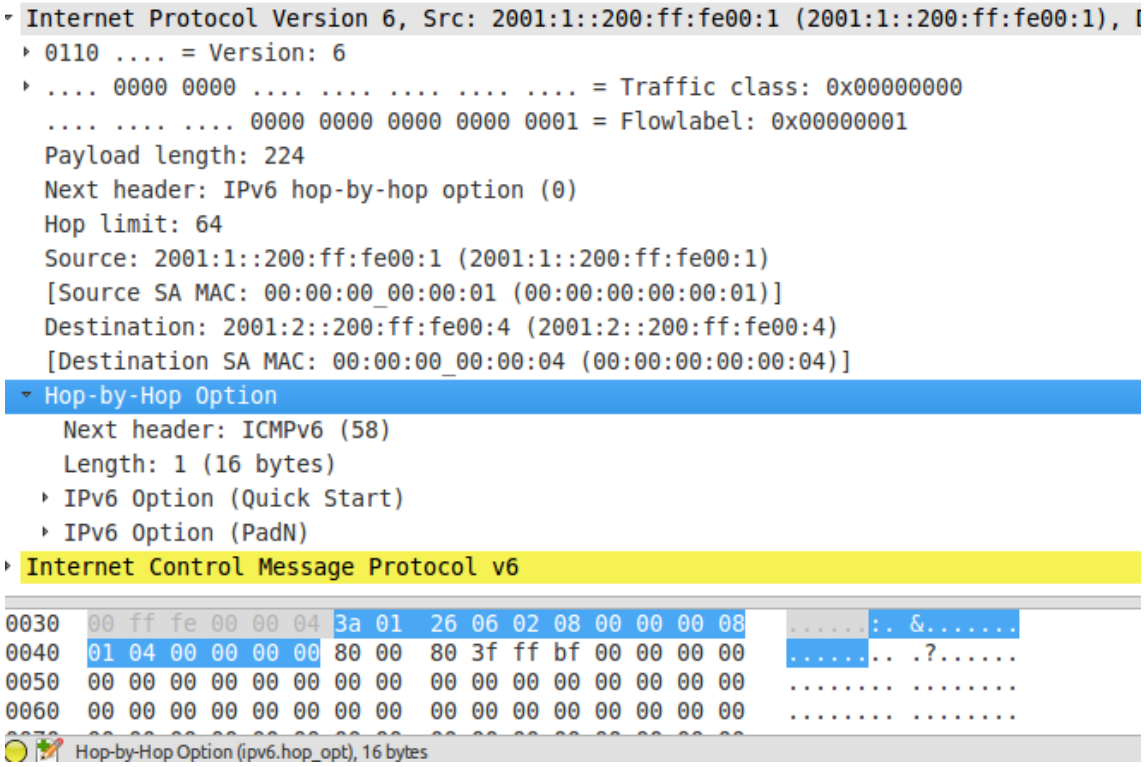

Figura 1.10.44 – Paquete con Quick Start, vista general (nodo 0)

La cabecera de *Hop by Hop* se presenta correctamente con un tamaño de 16 bytes, siendo múltiplo de 8, tal y como se exige en la *RFC 2460*. Contiene, además de los campos básicos *Next header* y *Length*, la opción *Quick Start* y la opción *PadN*. A continuación puede observarse el desglose de campos de la opción *Quick Start*:

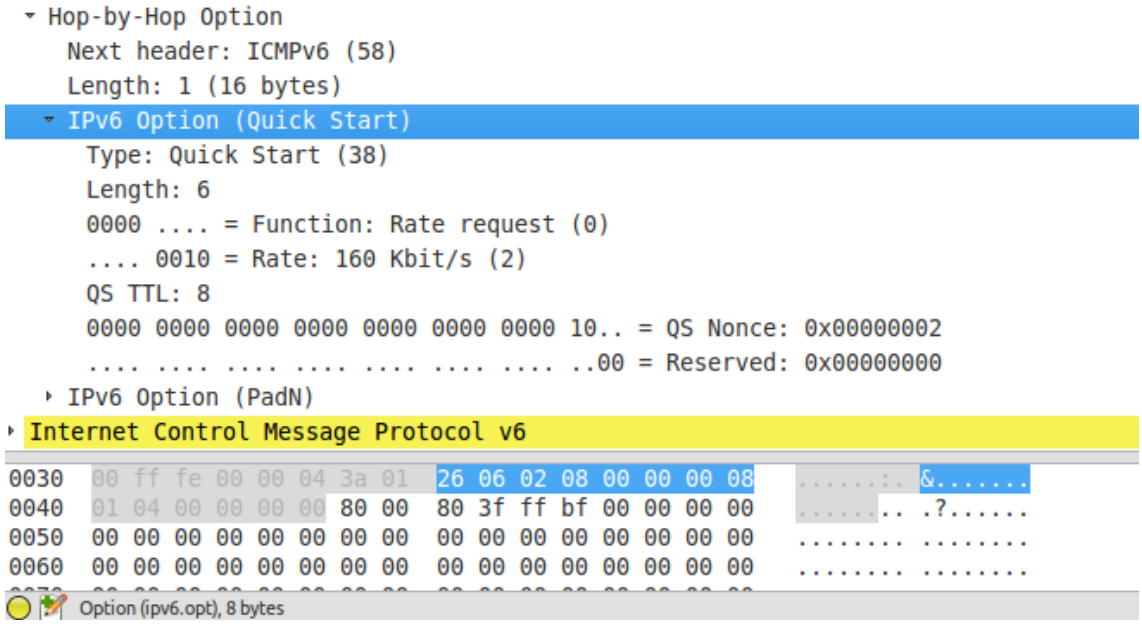

Figura 1.10.45 – Paquete con *Quick Start*, detalle *Quick Start* (nodo 0)

El campo *QS TTL* tiene, en efecto, el valor 8 que le fue asignado en el nodo origen.

La siguiente figura 1.10.46 muestra la opción contigua a *Quick Start*, *PadN*:

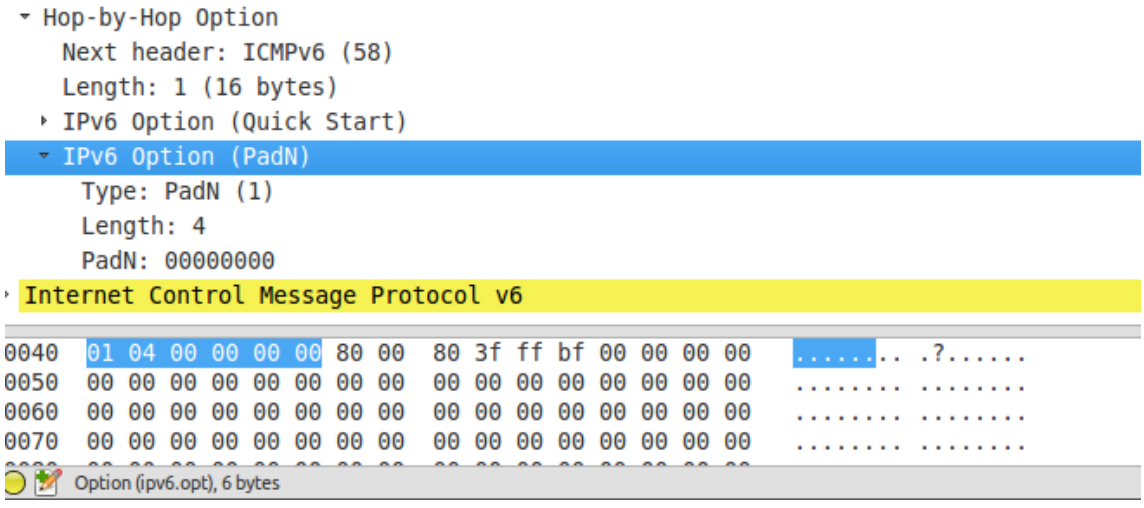

Figura 1.10.46 – Paquete con *Quick Start*, detalle *PadN* (nodo 0)

La opción ocupa un total de 6 bytes, necesarios para que, sumados a los dos bytes iniciales de la extensión *Hop by Hop* y a los 8 bytes de la opción *Quick Start*, la totalidad de la cabecera *Hop by Hop* sea de 16 bytes:

2 bytes (*Hop by Hop*) + 8 bytes (Quick Start) + 4 bytes (*PadN*) = 16 bytes

La última figura 1.10.47 muestra el mismo paquete recibido en el nodo 1; es decir, en el nodo destino:

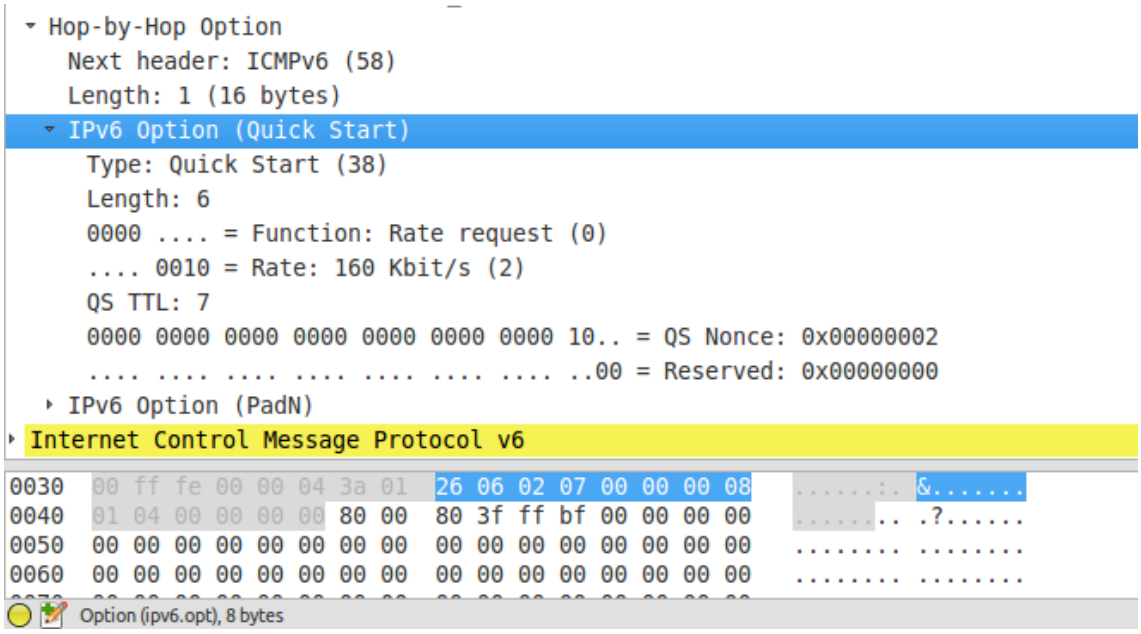

Figura 1.10.47 – Paquete con *Quick Start*, detalle *Quick Start* (nodo 1)

Como se esperaba, el valor del campo *QS TTL* ha sido reducido en una unidad, teniendo ahora el valor 7, confirmando que el nodo 1 ha modificado la cabecera en ruta durante el procesamiento del paquete en su recepción.

1.11 Conclusiones

Una vez finalizado el proyecto, podemos decir que han sido cumplidos los objetivos que éste tenía marcados.

Han sido añadidas nuevas opciones de *IPv6* al simulador *NS3*, concretamente las opciones *Calipso* y *Quick Start*, tanto la generación y procesamiento de sus cabeceras como la implementación de algunas de sus funcionalidades. Las opciones *Router Alert* y *Jumbogram*, ya introducidas anteriormente en *NS3*, han sido actualizadas con nuevas características.

El sistema de procesamiento de la extensión *Hop by Hop* ha sido modificado para hacer posible la modificación en ruta de las cabeceras de las opciones de *IPv6*, funcionalidad completamente ausente anteriormente en *NS3*.

La programación de estas opciones y las modificaciones sobre la extensión *Hop by Hop* han requerido de un examen exhaustivo del simulador *NS3* y su código. Éste contiene una cantidad considerable de módulos y de líneas de código que han sido estudiados debidamente y sobre los que se ha invertido gran cantidad de tiempo en su examen y en la realización previa de pruebas para comprender el funcionamiento del programa; proceso que ha sido necesario realizar antes de realizar todas las modificaciones que han ayudado a cumplir con los objetivos del proyecto.

Además, ha sido desarrollada una serie de aplicaciones que demuestran el funcionamiento de las características programadas y de la implementación de *Hop by Hop* ya presente en *NS3* antes de comenzar el proyecto. Estas aplicaciones y su código pueden servir de guía a futuros programadores y usuarios que quieran hacer uso de las opciones de *Hop by Hop* en *NS3*, al igual que otras aplicaciones ya presentes en *NS3* han sido de gran ayuda a la hora de entender el funcionamiento del simulador durante el desarrollo de este proyecto.

Por último, se ha generado la documentación requerida por este proyecto, la cual ayuda a un futuro usuario a realizar sus propias modificaciones sobre las características programadas en este proyecto.

2. PLIEGO DE CONDICIONES

2.1 Índice

2.2 Condiciones técnicas

En este apartado se describen los requisitos técnicos para la correcta ejecución del proyecto, incluyendo tanto los recursos materiales como los recursos humanos necesarios.

2.2.1 Hardware PC

A continuación se numeran las características del hardware utilizado en el desarrollo del presente proyecto:

- Ordenador personal con procesador *Intel Core i5-2500 CPU* de 3.30 GHz, tanto como para programación del proyecto como para las pruebas y ejecuciones de los escenarios.
- 4 GBs de memoria *RAM*.
- Tarjeta de red *Ethernet* y conexión a Internet para descarga de librerías y consulta de información.

2.2.2 Librerías y software adicional

En este sub-apartado se listan el software y las librerías instalados que han sido utilizados en el proyecto para su desarrollo. La versión del paquete instalado de cada software acompaña a este listado, si bien una versión más moderna de estos paquetes no debería dar problema para un correcto funcionamiento:

- Sistema Operativo *GNU/Linux*, distribución *Ubuntu 13.10*, con el *kernel 3.11.0-12-generic* instalado en el ordenador personal y con el software y librerías de las siguientes líneas.
- Sistema Operativo *Microsoft Windows 7*, instalado en el mismo ordenador que el anterior.
- El compilador *g++* para *GNU/Linux* versión *4:4.8.1-2Ubuntu3*.

- El software *Mercurial*, versión 2.6.3-1, utilizado para revisar y actualizar el repositorio del código del proyecto.

- El software *TortoiseHG,* versión 2.8-1, herramienta gráfica utilizada como complemento a *Mercurial*.

- El software *Wireshark,* versión 1.12.0 Release Candidate 3, con el código modificado, utilizado para visualización del tráfico generado en las redes simuladas.

- Editor de texto *gedit*, versión 3.8.3.

- El software *Doxygen*, versión 1.8.4-1, utilizado en este proyecto para la generación de la documentación del código fuente de *NS3*.

- Paquete ofimático *Microsoft Office 2010*, incluyendo, como mínimo, los programas *Microsoft Word* y *Microsoft Visio* para la realización de la documentación.

2.2.3 Recursos humanos

Las personas encargadas de la ejecución del proyecto han debido cumplir los siguientes requisitos:

- Un Director de Proyecto que marque los objetivos del proyecto y asista al Ingeniero Programador en la toma de decisiones que afecten a su desarrollo. Debe tener amplios conocimientos sobre redes *Ipv6*.

- Un Ingeniero Técnico de Telecomunicación con conocimientos en el lenguaje de programación *C++*, en redes *Ipv6*, en los entornos *GNU/Linux* y *Microsoft Windows* y en la suite ofimática *Microsoft Office*. Es el encargado de hacer todas las modificaciones al código de *NS3*, programar los escenarios y redactar toda la documentación del trabajo realizado.

2.2.4 Normativas utilizadas

En este apartado se describen las normas, tanto legales como recomendaciones, que se han seguido durante el desarrollo del proyecto. Las recomendaciones se han seguido en base al juicio del autor, que consideró adecuado el seguimiento de éstas para un mejor resultado final.

GPL

El código fuente del simulador *Network Simulator 3* está licenciado bajo la *General Public License v2,* queriendo decir esto que este mismo código podrá ser utilizado y modificado en el desarrollo del presente proyecto, además de ser re-distribuido y comercializado con libertad con la condición de proporcionar siempre el código fuente del programa modificado.

Este proyecto trabaja sólo con el código fuente de *NS3* y con el código creado por el autor, sin tomar de otras fuentes.

RFCs

La implementación de las librerías y programas desarrollados en el proyecto siguen e interpretan multitud de documentos *RFC (Request for Comments)* que dictan las normas de implementación de los distintos objetivos del trabajo.

En el apartado 1.4 de la Memoria, *Referencias*, vienen citados los distintos documentos estudiados en el proyecto.

Estilo de NS3

El proyecto *NS3* dicta ciertas normas de estilo de código en sus módulos requeridas en caso de querer contribuir al desarrollo del proyecto. Este proyecto sigue estas normas y adapta el código generado a dicho estilo.

Doxygen

El código redactado debe estar documentado mediante *Doxygen*, por lo que se debe utilizar su sintaxis específica para tal cometido, de manera que se genere automáticamente la documentación *Doxygen*.

3. ESTADO DE LAS MEDICIONES

3.1 Índice

3.2 Mediciones

Este apartado define y determina las unidades de cada partida que configuran la totalidad del proyecto.

Código	Resumen	Unidades(horas)
CAPÍTULO 1 – ESTUDIO Y REVISIÓN DE LA IMPLEMENTACIÓN		
	Estudio del funcionamiento del programa NS3 y de la estructura de su código, así como la revisión de la implementación de IPv6 en dicho software, para así poder aplicar los cambios necesarios y cumplir con los objetivos del proyecto.	
1.1	Estudio previo de *NS3*	**30**
1.2	Revisión de *IPv6* en *NS3*	**20**
CAPÍTULO 2 – DISEÑO DEL PROYECTO		
	Trabajo dedicado a la investigación de soluciones y métodos adecuados a aplicar durante el desarrollo del proyecto.	
2.1	Diseño de opciones *Hop by Hop*	**15**
2.2	Diseño de sistema de procesamiento de la extensión *Hop by Hop*	**4**
CAPÍTULO 3 – IMPLEMENTACIÓN		
	Todo el esfuerzo dedicado a la programación de código siguiendo los requisitos de diseño marcados en el proyecto.	
3.1	Programación de opciones de *IPv6*	**60**
3.2	Programación del *Hop by Hop Tag*	**10**
3.3	Programación de las aplicaciones escenario	**20**
CAPÍTULO 4 – REDACCIÓN DE LA DOCUMENTACIÓN		
	Documentación del código con Doxygen y redacción de los manuales de Usuario y de Mantenimiento	
4.1	Documentación del código	**10**
4.2	Redacción del Manual de Usuario y del Manual de Mantenimiento	**10**

4. PRESUPUESTO

4.1 Índice

4.2 Presupuesto

En este documento se determinará el coste económico de cada partida de obra y del proyecto en su totalidad.

4.2.1 Capítulo 1. Estudio y revisión de la implementación

Unidades (hora)	**Concepto**	**Precio unitario (€)**	**Subtotal (€)**
30	Estudio previo de *NS3*	10.00	300.00
20	Revisión de *IPv6* en *NS3*	10.00	200.00

Subtotal (€): **500.00**

4.2.2 Capítulo 2. Diseño del proyecto

Unidades (hora)	**Concepto**	**Precio unitario (€)**	**Subtotal (€)**
15	Diseño de opciones *Hop by Hop*	40.00	600.00
4	Diseño de sistema de procesamiento de la extensión *Hop by Hop*	40.00	160.00

Subtotal (€): **760.00**

4.2.3 Capítulo 3. Implementación

Unidades (hora)	**Concepto**	**Precio unitario (€)**	**Subtotal (€)**
60	Programación de opciones de *IPv6*	60.00	3,600.00
10	Programación del *Hop by Hop Tag*	60.00	600.00
20	Programación de las aplicaciones escenario	60.00	1,200.00

Subtotal (€): **5,400.00**

4.2.4 Capítulo 4. Redacción de la documentación

Unidades (hora)	Concepto	Precio unitario (€)	Subtotal (€)
10	Documentación del código	40.00	400.00
10	Redacción del Manual de Usuario del Manual de Mantenimiento	40.00	400.00

Subtotal (€): **800.00**

4.2.5 Presupuesto total del proyecto

Total – Capítulo 1 + Capítulo 2 + Capítulo 3 + Capítulo 4 (€): **7,460.00**

IVA (21%) (€): **1,566.60**

Total + IVA (€): **9,026.60**

Asciende el presente Presupuesto de Proyecto de Programación de Opciones *IPv6 Hop by Hop* en *NS3* a la cantidad de **NUEVE MIL VEINTISÉIS CON SESENTA CÉNTIMOS DE EURO**.

En Linares, a 18 de septiembre de 2014.

5. ANEXOS

5.1 Índice

5.2 Manual de usuario

NS3 puede ser utilizado en varios sistemas operativos, aunque este manual va dirigido a su instalación y ejecución *GNU/Linux*, concretamente una distribución basada en *Debian*, ya sea la misma *Debian* o una distribución de *Ubuntu*.

Este manual comprende la instalación de prerrequisitos para poder compilar y ejecutar *NS3*, las instrucciones de compilación y, por último, indicaciones para la ejecución y configuración de los escenarios analizados en la Memoria. Este manual está inspirado en el tutorial de instalación de *NS3* del sitio web del proyecto original [9], con los cambios y añadidos adecuados a este proyecto.

5.2.1 Instalación de prerrequisitos

NS3 requiere la instalación de ciertos paquetes para su uso. Una forma de instalar estos paquetes es haciendo uso de la terminal y del programa *apt* propio de las distribuciones basadas en *Debian*.

Los siguientes paquetes fueron instalados para el desarrollo de este proyecto, incluyendo el comando en la terminal:

sudo apt-get install gcc g++ python python-dev mercurial bzr gdb valgrind gsl-bin libgsl0-dev libgsl0ldbl flex bison libfl-dev g++-3.4 gcc-3.4 TCPdump sqlite sqlite3 libsqlite3-dev libxml2 libxml2-dev libgtk2.0-0 libgtk2.0-dev vtun lxc uncrustify doxygen graphviz imagemagick texlive texlive-extra-utils texlive-latex-extra python-sphinx dia python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev libboost-signals-dev libboost-filesystem-dev openmpi-bin openmpi-common openmpi-doc libopenmpi-dev gcc-multilib

Con estos paquetes instalados no debería haber problema para compilar *NS3* y ejecutar los escenarios programados en este proyecto.

5.2.2 Instalación de NS3

En este apartado se explica cómo integrar los cambios realizados en este proyecto en el código de *NS3* para así obtener un entorno de desarrollo con el que poder trabajar con las nuevas características programadas, incluyendo la ejecución de los escenarios que prueban estas características.

Como primer paso, es necesario obtener una versión de desarrollo de *NS3* que contenga, al menos, hasta la rama número 10226. La siguiente figura muestra una captura del software *TortoiseHG*, el cual muestra de forma gráfica las revisiones del proyecto:

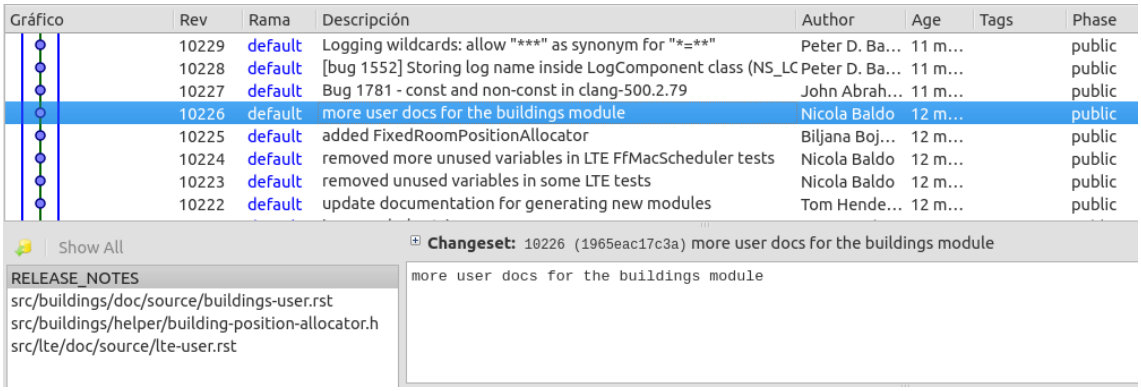

Figura 5.2.1 – Versión de *NS3* con la rama 10226

Esta versión puede descargarse desde la página oficial del proyecto *NS3* (http://www.nsnam.org/ns-3-dev/) o puede utilizarse el paquete *ns-3-dev.tar-gz* que acompaña a este proyecto en su *CD*.

Los cambios introducidos en el código de *NS3* por este proyecto comenzaron a partir de la rama 10226, por lo que es necesario partir de esta rama y no de una posterior.

El siguiente paso sólo es necesario realizarlo en caso de haber obtenido *NS3* descargado de su página oficial. Al descargar el proyecto mediante este método, éste vendrá actualizado a una versión más moderna, por lo que es necesario deshacerse de todas las versiones posteriores a la rama 10226. El siguiente comando crea un clon del proyecto que tendrá como última revisión la rama 10226:

hg clone –r 10226 ruta-proyecto-original ruta-proyecto-clon

```
miguel@miguel-VirtualBox:~$ cd ns3
miguel@miguel-VirtualBox:~/ns3$ hg clone -r 10226 ns-3-dev ns-3-dev-clone
adding changesets
adding manifests
adding file changes
added 10227 changesets with 45786 changes to 7045 files
updating to branch default
2808 files updated, 0 files merged, 0 files removed, 0 files unresolved
miguel@miguel-VirtualBox:~/ns3$
```

Figura 5.2.2 – Comando *hg clone*

Como se aprecia en la siguiente figura, el proyecto quedará solamente actualizado hasta la revisión 10226:

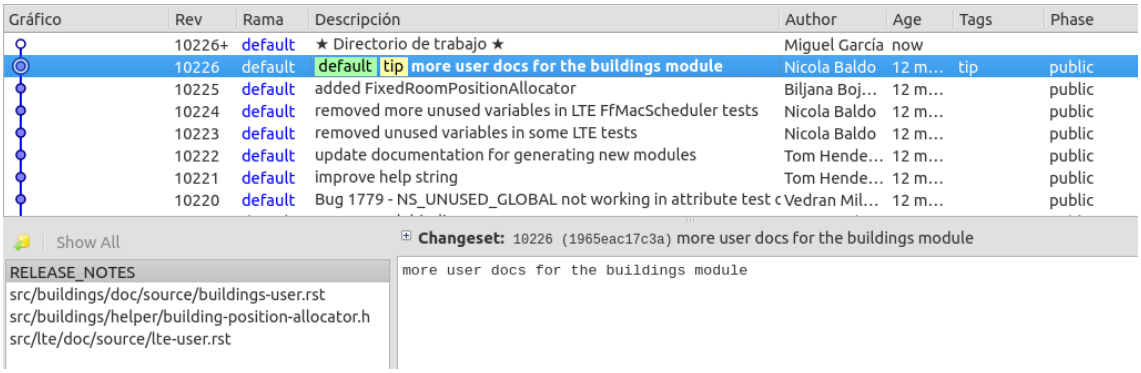

Figura 5.2.3 – Versión de *NS3* con la rama 10226 como última revisión disponible

Si se trabaja desde el fichero *ns-3-dev.tar-gz* no será necesario utilizar el comando *hg clone*, puesto que ya viene aplicado de antemano.

Queda todo listo ahora para aplicar el parche que introduce todos los cambios realizados por este proyecto. Consiste en un fichero de extensión *diff*, utilizado por el software *Mercurial* para aplicar cambios en el código fuente de manera automática.

El fichero en cuestión es *parche.diff*, el cual se encuentra en el *CD* de este proyecto, y debe aplicarse, habiendo copiado el fichero del parche en el directorio raíz de *NS3*, con el siguiente comando:

hg import parche.diff

```
miguel@miguel-VirtualBox:~/ns3/ns-3-dev-clone$ hg import parche.diff
applying parche.diff
miguel@miguel-VirtualBox:~/ns3/ns-3-dev-clone$
```

Figura 5.2.4 – Comando *hg import*

Si se ha realizado correctamente, el proyecto debería contener ahora todas las revisiones introducidas durante el desarrollo de este proyecto, tal y como se muestra en la siguiente captura del software *TortoiseHG*:

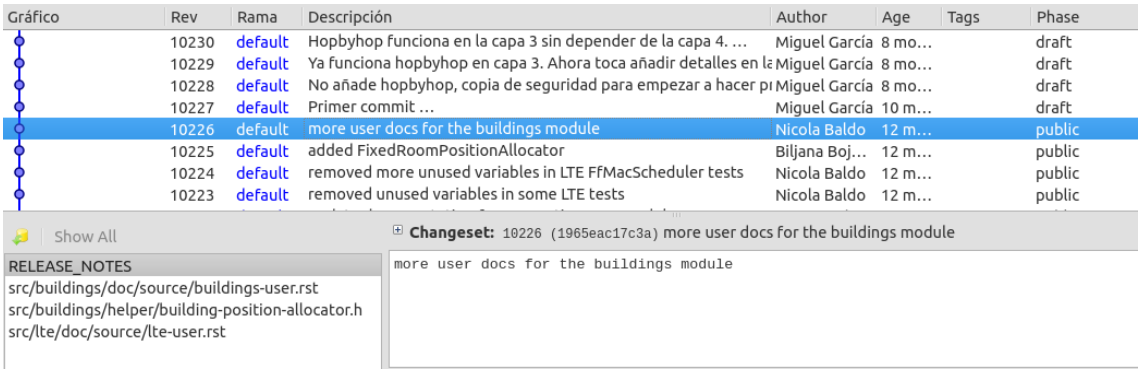

Figura 5.2.5 – Versión de *NS3* actualizada con las revisiones realizadas en este proyecto

5.2.3 Compilación

La siguiente fase comprende los pasos para la compilación del código del simulador, incluyendo los módulos programados y modificados durante el desarrollo del proyecto.

El primer paso consiste en situarse en el directorio raíz del proyecto con un terminal de comandos y ejecutar la siguiente instrucción:

./waf -d debug --enable-examples --enable-tests configure

El sistema empezará a comprobar qué utilidades y librerías hay instaladas en el equipo:

```
root@miguel-VirtualBox:~/ns-3# ./waf -d debug --enable-examples --enable-tests configure
invalid lock file in /home/miguel/ns-3
Setting top to                           : /home/miguel/ns-3
Setting out to                           : /home/miguel/ns-3/build
Checking for 'gcc' (c compiler)          : /usr/bin/gcc
Checking for cc version                  : 4.8.1
Checking for 'g++' (c++ compiler)        : /usr/bin/g++
Checking for compilation flag -Wl,--soname=foo... support : ok
Checking for program python                                : /usr/bin/python
Checking for python version                                : (2, 7, 5, 'final', 0)
Checking for library python2.7 in LIBDIR                   : yes
Checking for program /usr/bin/python-config,python2.7-config,python-config-2.7,python2.7m-config : /usr/bin/python-config
Checking for header Python.h                                                                       : yes
Checking for compilation flag -fvisibility=hidden... support                                       : ok
Checking for compilation flag -Wno-array-bounds... support                                         : ok
Checking for pybindgen location                                                                    : not found
Python module pybindgen                                                                            : not found
pybindgen missing => no python bindings
```

Figura 5.2.6 – Configure a)

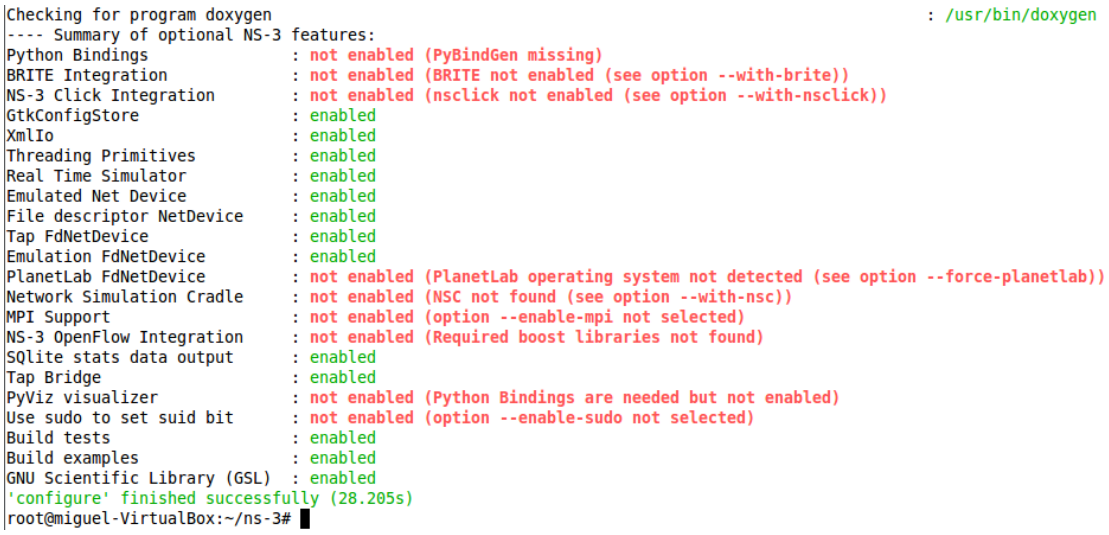

Figura 5.2.7 – Configure b)

Como se muestra en la figura 5.2.7, algunas características pueden no estar instaladas en el equipo. Esto no es problema, al ser sólo opcionales y no ser necesarias para la ejecución de los escenarios y la utilización de las características programadas en este proyecto.

El siguiente y último paso para compilar *NS3* es el de introducir el siguiente comando en el directorio raíz del proyecto:

./waf

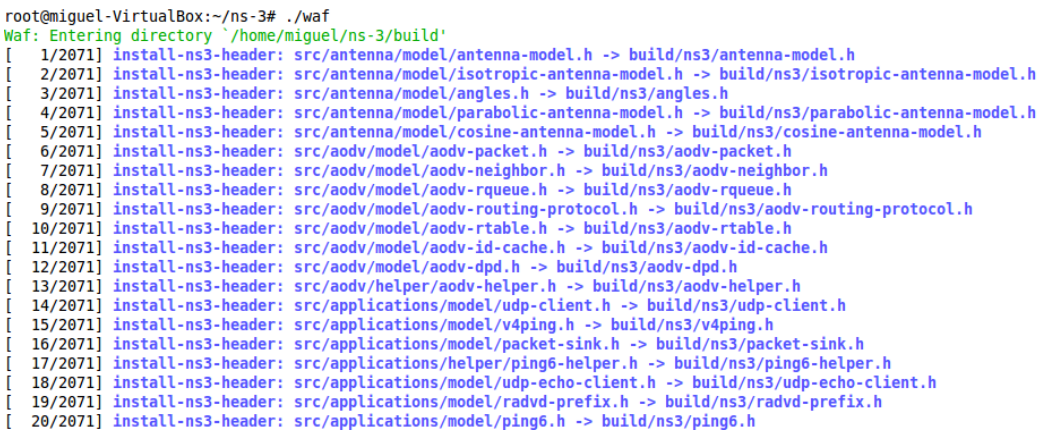

Figura 5.2.8 – Compilación de *NS3*

Una vez termine este proceso, el cual se puede alargar durante varios minutos, el simulador estará listo para su utilización y los escenarios programados podrán ser ejecutados.

5.2.4 Ejecución de los escenarios

Cada opción *Hop by Hop* tiene su propio programa con los escenarios programados en ellos. Este apartado explicará cómo ejecutar cada uno de ellos, obteniendo la salida del programa con toda la información relevante para su análisis y las capturas de tráfico que pueden ser utilizadas mediante el software *Wireshark.*

En primer lugar, es necesario copiar los ficheros de simulación a la carpeta */ns-3-dev/scratch*. Estos ficheros son:

- *routeralert.cc*
- *jumbogram.cc*
- *calipso.cc*
- *quickstart.cc*

Una vez hecho esto y estando situado en dicha carpeta en el terminal de comandos, se deberá introducir la siguiente línea para ejecutar el escenario de *Router Alert* descrito en el apartado 1.9.8 de la Memoria:

./waf -- run routeralert

Para ejecutar el escenario de Quick Start analizado en el apartado 1.9.11 de la Memoria, se debe introducir la siguiente línea en el terminal de comandos:

./waf -- run quickstart

La ejecución del escenario preparado para la opción *Calipso*, analizado en el apartado 1.9.10 de la Memoria, es posible mediante la introducción de este comando en el directorio donde se encuentre el fichero del escenario:

./waf -- run calipso

El programa preparado para la opción *Jumbogram* requiere de un argumento adicional en la ejecución, ya que son cinco los escenarios analizados en la Memoria y que pueden ser ejecutados. Estos comandos son, según el escenario, los que vienen a continuación.

- Escenario del apartado 1.9.9.1 de la Memoria:

  ./waf -- run "Jumbogram -- simulation=0"

- Escenario del apartado 1.9.9.2 de la Memoria:

  ./waf -- run "Jumbogram -- simulation=1"

- Escenario del apartado 1.9.9.3 de la Memoria:

  ./waf -- run "Jumbogram -- simulation=2"

- Escenario del apartado 1.9.9.4 de la Memoria:

  ./waf -- run "Jumbogram -- simulation=3"

- Escenario del apartado 1.9.9.5 de la Memoria:

  ./waf -- run "Jumbogram -- simulation=4"

En cada ejecución, se debería obtener una salida similar a la de la siguiente captura:

```
Waf: Entering directory `/home/miguel/ns-3-dev/build'
Waf: Leaving directory `/home/miguel/ns-3-dev/build'
'build' finished successfully (1.724s)
root@miguel-VirtualBox:~/ns-3-dev/scratch#
```

Figura 5.2.9 – Ejecución de escenario a)

Para obtener datos de salida generados por las clases durante la ejecución es necesario añadir un argumento adicional a los anteriores comandos. He aquí un ejemplo de cómo hacerlo con el escenario de *Jumbogram,* donde hay que añadir el argumento *verbose* para habilitar la salida:

./waf -- run "Jumbogram -- simulation=3 -- verbose"

De esta forma, el programa muestra a la salida los mensajes *NS_LOG* generados por las clases durante la ejecución que hayan sido indicadas en cada escenario:

```
root@miguel-VirtualBox:~/ns-3-dev/scratch# ./waf --run "jumbogram --simulation=3 --verbose"
Waf: Entering directory `/home/miguel/ns-3-dev/build'
Waf: Leaving directory `/home/miguel/ns-3-dev/build'
'build' finished successfully (1.357s)
Create nodes.
Create IPv6 Internet Stack
Ipv6L3Protocol:Ipv6L3Protocol()
Ipv6L3Protocol:SetSendIcmpv6Redirect(0x84141c0, 1)
Ipv6L3Protocol:SetIpForward(0x84141c0, 0)
Ipv6L3Protocol:SetMtuDiscover(0x84141c0, 1)
Ipv6L3Protocol:NotifyNewAggregate()
Ipv6L3Protocol:SetNode(0x84141c0, 0x83f0478)
Ipv6L3Protocol:SetupLoopback()
Ipv6L3Protocol:GetIcmpv6()
Ipv6L3Protocol:GetProtocol(0x84141c0, 58)
Ipv6L3Protocol:AddIpv6Interface(0x84141c0, 0x841b088)
Ipv6L3Protocol:NotifyNewAggregate()
Ipv6L3Protocol:Insert(0x84141c0, 0x841b0f8)
Ipv6L3Protocol:NotifyNewAggregate()
Ipv6L3Protocol:SetRoutingProtocol(0x84141c0, 0x8414198)
Ipv6L3Protocol:GetNInterfaces()
Ipv6L3Protocol:IsUp(0x84141c0, 0)
Ipv6L3Protocol:GetInterface(0x84141c0, 0)
Ipv6L3Protocol:GetNAddresses(0x84141c0, 0)
```

Figura 5.2.10 – Ejecución de escenario b)

5.2.5 Configuración de escenarios

Los escenarios están programados no sólo con el objetivo de mostrar los resultados de la ejecución, sino que pueden ser utilizados como base para futuras modificaciones por otro programador. Es por ello que se describe en este apartado el código de los escenarios y cómo modificar sus líneas para obtener topologías diferentes y envíos de paquetes distintos.

Se partirá del escenario de la opción *Jumbogram*, siendo éste el más completo de todos. No será necesario describir el resto, ya que todos los escenarios están programados utilizando los mismos métodos y estructura en el código.

Este apartado analizará ciertas secciones del código y mostrará cómo modificar los parámetros para cambiar el escenario planteado. La primera sección de código hace referencia a la información que mostrará el programa a la salida de la ejecución:

```
int main (int argc, char **argv)
{
  bool verbose = false;
  int simulation = 0;
  GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
  CommandLine cmd;
  cmd.AddValue ("verbose", "turn on log components", verbose);
  cmd.AddValue ("simulation", "choose the simulation to run", simulation);
  cmd.Parse (argc, argv);

  if (verbose)
    {
      LogComponentEnable ("Ipv6Option", LOG_LEVEL_ALL);
      LogComponentEnable ("Ipv6Extension", LOG_LEVEL_INFO);
      LogComponentEnable ("Ipv6L3Protocol", LOG_LEVEL_LOGIC);
      LogComponentEnable ("JumbogramExample", LOG_LEVEL_LOGIC);
    }
```

Figura 5.2.11 – Configuración NS_LOG

Las líneas *LogComponentEnable* habilitan el *log* de un módulo en concreto. Si el usuario desea añadir el *log* del módulo, por ejemplo, *ipv6-interface*, la siguiente línea de código debería ser añadida dentro de la estructura condicional de la figura superior:

LogComponentEnable ("Ipv6Interface", LOG_LEVEL_X);

Donde X puede tener un valor distinto según el nivel de información que se quiera mostrar (ALL, INFO, LOGIC, etc.) y de acuerdo a la definición del *log* del módulo ipv6-*interface* con la siguiente línea:

NS_LOG_COMPONENT_DEFINE ("Ipv6Interface");

La siguiente sección de código referencia a los nodos creados durante la simulación:

```
NS_LOG_INFO ("Create nodes.");
Ptr<Node> h0 = CreateObject<Node> ();
Ptr<Node> h1 = CreateObject<Node> ();
Ptr<Node> r0 = CreateObject<Node> ();
Ptr<Node> r1 = CreateObject<Node> ();
Ptr<Node> r2 = CreateObject<Node> ();
Ptr<Node> r3 = CreateObject<Node> ();

NodeContainer net1 (h0, r0);
NodeContainer net2 (r0, r1);
NodeContainer net3 (r1, r2);
NodeContainer net4 (r2, r3);
NodeContainer net5 (r3, h1);
```

Figura 5.2.12 – Configuración de nodos

Cada línea *Ptr<Node> nombre_nodo = CreateObject<Node> ( )* crea un nodo que podrá agregarse con posterioridad a una red *Ipv6* y al que se le podrán asignar interfaces con direcciones *IP*. Estos nodos podrán ser un *host* o un *router*, dependiendo de su posterior configuración.

Cada línea *NodeContainer nombre_enlace (nodo1, nodo2)* crea un enlace entre dos nodos *nodo1* y *nodo2*. Estos nodos podrán seguir siendo enlazados con otros nodos formando una red con varios nodos. Estarán o no en la misma red dependiendo de las direcciones *IP* que les sean asignadas a sus interfaces. La anterior línea de ejemplo daría el resultado de la siguiente figura:

Figura 5.2.13 – Enlace entre dos nodos

La siguiente sección de código hace referencia a la asignación de direcciones IP a cada enlace y nodo y a la configuración de cada nodo como un router o como un host; es decir, habilitar el *forwarding* en una interfaz.

```
NS_LOG_INFO ("Create networks and assign IPv6 Addresses.");
Ipv6AddressHelper ipv6;

ipv6.SetBase (Ipv6Address ("2001:1::"), Ipv6Prefix (64));
Ipv6InterfaceContainer i1 = ipv6.Assign (d1);
i1.SetForwarding (1, true);
i1.SetDefaultRouteInAllNodes (1);

ipv6.SetBase (Ipv6Address ("2001:2::"), Ipv6Prefix (64));
Ipv6InterfaceContainer i2 = ipv6.Assign (d2);
i2.SetForwarding (0, true);
i2.SetDefaultRouteInAllNodes (0);
i2.SetForwarding (1, true);
i2.SetDefaultRouteInAllNodes (1);

ipv6.SetBase (Ipv6Address ("2001:3::"), Ipv6Prefix (64));
Ipv6InterfaceContainer i3 = ipv6.Assign (d3);
i3.SetForwarding (0, true);
i3.SetDefaultRouteInAllNodes (0);
i3.SetForwarding (1, true);
i3.SetDefaultRouteInAllNodes (1);
```

Figura 5.2.14 – Configuración de redes

Cada línea *ipv6.SetBase (Ipv6Address ("dirección_IP"), Ipv6Prefix (64))* crea una nueva red *IPv6* dentro del *stack* de *IPv6* creado *en Ipv6AddressHelper ipv6*.

Partiendo ahora de un *Interface Container*, se le asigna la anterior *IP* a un enlace. La línea *ipv6.Assign(nombre_enlace)* asigna una dirección única a todas las interfaces contenidas en el enlace dentro de esa red automáticamente.

La línea *nombre_interfaz.SetForwarding(índice_interfaz, true)* configura la interfaz indicada por el índice como un router.

El efecto del segundo párrafo de la figura 5.2.14 sería, pues, el mostrado en la figura 5.2.15, donde se asignan las direcciones *IP* de la red *2001:1::* :

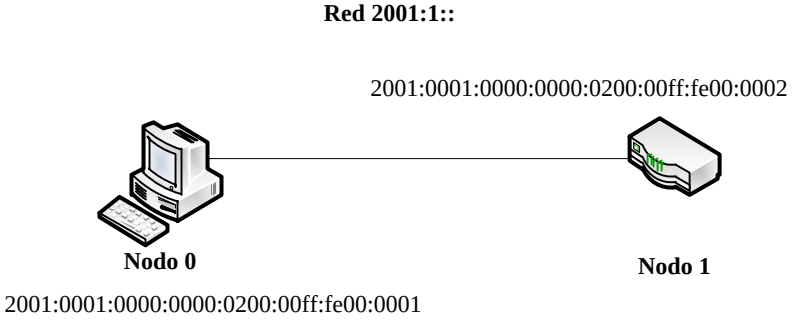

Figura 5.2.15 – Asignación de direcciones en los nodos

En la aplicación *MyApp* de los escenarios es donde se forma el paquete que será enviado. En estos escenarios se envía el paquete de dos formas distintas, según la situación:

- Primera forma: construir un paquete con la cabecera de los niveles de transporte y de *IP* completas. Este paquete se enviaría directamente al nivel de enlace por la interfaz seleccionada y no pasaría por los métodos del módulo *Ipv6L3Protocol*. Por eso es necesario que el paquete esté completamente formado a nivel de *IP*. Este método es utilizado cuando se quiere enviar un paquete que no se podría enviar utilizando el *stack* de *Ipv6* propio de *NS3*, ya que, por ejemplo, es éste el que asigna el valor del *Payload Length* de la cabecera de *IP* y en varios de los escenarios este valor debe ser *0*.

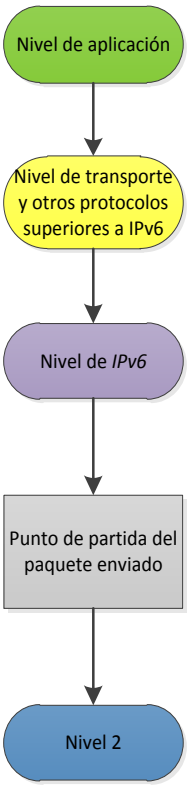

Figura 5.2.16 – Primer método de envío de paquete

- Segunda forma: Construir un paquete sólo hasta el nivel de transporte y dejar que las cabeceras del nivel de *IP* sean añadidas por el módulo *Ipv6L3Protocol*. Este método requeriría el uso de *Packet tags* para añadir la extensión *Hop by Hop* y sus opciones.

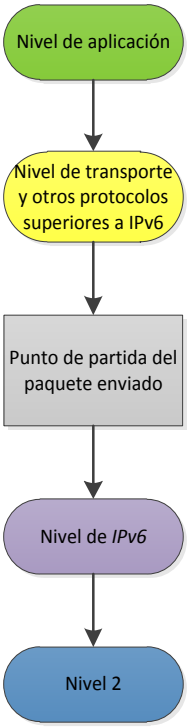

Figura 5.2.17 – Segundo método de envío de paquete

A continuación se describe, paso a paso, el código para las dos formas utilizadas de enviar el paquete. Ambas comienzan con el mismo código:

```
Ptr<Packet> p;
Ptr<Ipv6L3Protocol> ipv6;
Ptr< Ipv6Route > route;
Ipv6OptionJumboTag jumboTag;

// Ipv6 Header //
Ipv6Header hdr;
hdr.SetSourceAddress (m_i1.GetAddress (0, 1));
hdr.SetDestinationAddress (m_i2.GetAddress (1, 1));
hdr.SetHopLimit (255);
hdr.SetTrafficClass (0);
```

Figura 5.2.18 – Envío de paquete a)

Las cuatro primeras líneas definen variables que serán utilizadas más adelante. Las cuatro siguientes líneas de código configuran algunos parámetros de la cabecera de *Ipv6*.

```
// Hop by hop Header //
Ipv6ExtensionHopByHopHeader hopbyhopHeader;
hopbyhopHeader.SetNextHeader (Ipv6Header::IPV6_ICMPV6);
Ipv6OptionJumbogramHeader jumbo;


// ICMPv6 Echo Header //
Icmpv6Echo req (1);
req.SetId (0xFFBF);
```

Figura 5.2.19 – Envío de paquete b)

Sobre la figura superior, las tres primeras líneas definen la cabecera de la extensión *Hop by hop* y la opción *Jumbogram*. Esta cabecera será utilizada por el primer método, no por el segundo.

Las dos últimas líneas crean y definen una cabecera de *ICMPv6 Echo Request*.

Es importante aclarar que aún no ha sido añadida ninguna cabecera al paquete.

La siguiente sección de código se corresponde con el primer método de envío del paquete, donde se confecciona toda la cabecera relativa a niveles de transporte y de *IP* y se envía directamente al nivel 2:

```
p = Create<Packet> (70000);
p->AddHeader(req);
hdr.SetNextHeader (Ipv6Header::IPV6_EXT_HOP_BY_HOP);
jumbo.SetDataLength(p->GetSize() + 8);
hopbyhopHeader.AddOption(jumbo);
p->AddHeader(hopbyhopHeader);
hdr.SetPayloadLength (0);
p->AddHeader(hdr);
m_device.Get(0)->Send(p, m_device.Get(1)->GetAddress() , Ipv6L3Protocol::PROT_NUMBER);
```

Figura 5.2.20 – Envío de paquete c)

En primer lugar crea un paquete con un tamaño definido por el usuario; tras ello le es añadida la cabecera del mensaje *Echo Request*; a lo que le sigue el añadido de la cabecera de extensión *Hop by Hop* y la cabecera de *Ipv6*, después de haber fijado algunos de sus parámetros.

Tras ello, la última línea de código es la que envía el paquete *p* por la interfaz dada por *m_device.Get(0)*, con destino a la interfaz dada por *m_device.Get(1)* y que está conectada directamente a la anterior.

El segundo método de envío de paquete, utilizando el *stack* de *Ipv6*, es el que viene a continuación:

```
p = Create<Packet> (70000);
p->AddHeader(req);

jumboTag.SetDataLength(p->GetSize());
p->AddPacketTag (jumboTag);

ipv6 = m_node->GetObject<Ipv6L3Protocol> ();
route = Create<Ipv6Route> ();
route->SetDestination(hdr.GetDestinationAddress());
route->SetGateway(Ipv6Address ("2001:0001:0000:0000:0200:00ff:fe00:0002"));
route->SetOutputDevice(m_device.Get(0));
route->SetSource(hdr.GetSourceAddress());
ipv6->Send(p, hdr.GetSourceAddress(), hdr.GetDestinationAddress(), 58, route);
```

Figura 5.2.21 – Envío de paquete d)

Las dos primeras líneas crean un paquete de tamaño fijado por el usuario en bytes y añaden al paquete la cabecera del mensaje *ICMPv6 Echo Request*.

Las líneas tercera y cuarta fijan el parámetro del *Hop by Hop Tag* de la opción *Jumbogram* y añaden el *tag* al paquete.

El último párrafo de código invoca manualmente al método *Ipv6L3Protocol::Send*, el cual envía el paquete a la capa de *IPv6* y allí le son añadidas las cabeceras de *Hop by Hop* y de *Ipv6*, además de enviar el paquete con la ruta adecuada, configurada en este código.

5.2.6 Añadido de Hop by hop Tags

La cabecera de extensión *Hop by Hop* y sus opciones pueden ser añadidas manualmente al paquete utilizando los métodos de */src/internet/model/ipv6-option.cc*, como se ha podido observar en el anterior apartado. Sin embargo, las opciones pueden ser añadidas con más comodidad utilizando los llamados *Hop by Hop Tags* creados en este proyecto.

Para tal cometido, se pasa por el siguiente proceso:

- Definición de la variable del *Tag*.
- Configuración de parámetros.
- Añadido del *Tag* al paquete que se desea enviar por la capa de *Ipv6* de *NS3*.

Para añadir, por ejemplo, un *Hop by hop Tag* de la opción *Router Alert* a un paquete, en primer lugar se debe definir la variable del *Tag* con el método adecuado del módulo */src/internet/model/ipv6-hopbyhop-tag.cc*. En código, quedaría traducido de la siguiente forma:

Ipv6OptionRouterAlertTag RAtag; donde *RAtag* es el nombre de la variable.

A continuación se deben introducir los parámetros en el *Packet Tag*. La opción *Router Alert* tiene un parámetro configurable, correspondiente a los últimos cuatro bytes de la cabecera, el cual se configuraría con la siguiente línea de código, partiendo de la variable antes definida:

RAtag.SetRouterAlert(valor_del_campo);

Tras ello, se sigue como con cualquier otro *Packet Tag* de *NS3*, añadiendo el *Tag* al paquete:

packet -> AddPacketTag (RAtag);

5.2.7 Modificación y compilación de Wireshark modificado

La versión del software *Wireshark* utilizada en este proyecto es una modificación de la versión *Wireshark 1.12.0 Release Candidate 3*, la cual tiene unos cambios en una sección de su código fuente debido a los dos siguientes motivos:

- La interpretación errónea de los campos de la cabecera de la opción *Quick Start*.
- La incapacidad del programa de leer correctamente un paquete *IPv6* que contiene el campo *Payload Length* con el valor 0.

Una versión ya modificad se encuentra en el CD de este proyecto en el archivo comprimodo *wireshark.tar.gz*, aunque se muestran a continuación las modificaciones realizadas en el código.

En primer lugar se muestra el código modificado para leer correctamente una opción *Quick Start* en *IPv6*. La versión *1.12.0 Release Candidate 3* interpreta erróneamente esta opción, leyendo 17 bytes de cabecera donde en realidad son 16. Para evitar esto, debe

accederse al fichero de fuente */epan/dissectors/packet-ipv6.c* y eliminar o comentar las líneas que aparecen a continuación en la siguiente figura en el método *static int dissect_opts(argumentos)*:

```
proto_tree_add_item(opt_tree, hf_ipv6_opt_qs_func, tvb, offset, 1, ENC_NA);

if (function == QS_RATE_REQUEST) {
    proto_tree_add_item(opt_tree, hf_ipv6_opt_qs_rate, tvb, offset, 1, ENC_NA);
    offset += 1;
    proto_tree_add_item(opt_tree, hf_ipv6_opt_qs_ttl, tvb, offset, 1, ENC_NA);
    //ttl_diff = (iph->ip_ttl - tvb_get_guint8(tvb, offset) % 256);
    //offset += 1;
    //ti = proto_tree_add_uint_format_value(opt_tree, hf_ipv6_opt_qs_ttl_diff,
    //                        tvb, offset, 1, ttl_diff,
    //                        "%u", ttl_diff);
    //PROTO_ITEM_SET_GENERATED(ti);
    //proto_item_append_text(ti_opt, ", %s, QS TTL %u, QS TTL diff %u",
    //                       val_to_str_ext(rate, &qs_rate_vals_ext, "Unknown (%u)"),
    //                       tvb_get_guint8(tvb, offset), ttl_diff);
    offset += 1;
    proto_tree_add_item(opt_tree, hf_ipv6_opt_qs_nonce, tvb, offset, 4, ENC_NA);
    proto_tree_add_item(opt_tree, hf_ipv6_opt_qs_reserved, tvb, offset, 4, ENC_NA);
    offset += 4;
} else if (function == QS_RATE_REPORT) {
    proto_tree_add_item(opt_tree, hf_ipv6_opt_qs_rate, tvb, offset, 1, ENC_NA);
    offset += 1;
```

Figura 5.2.22 – Código de Wireshark a)

En segundo lugar, se muestra a continuación la modificación que debe hacerse en el código para que *Wireshark* sea capaz de leer correctamente paquetes *IPv6* que contengan el valor 0 en el campo *Payload Length*. Esta modificación se realiza en el mismo fichero fuente que en el caso anterior, */epan/dissectors/packet-ipv6.c*, como se muestra en la figura 5.2.23:

```
/* Adjust the length of this tvbuff to include only the IPv6 datagram. */

if (ipv6.ip6_plen==0)
  {
    set_actual_length(tvb, 70000);
  }
else
{
  set_actual_length(tvb, plen + (guint)sizeof (struct ip6_hdr));
}

TVB_SET_ADDRESS(&pinfo->net_src, AT_IPv6, tvb, offset + IP6H_SRC, 16);
TVB_SET_ADDRESS(&pinfo->src, AT_IPv6,     tvb, offset + IP6H_SRC, 16);
TVB_SET_ADDRESS(&pinfo->net_dst, AT_IPv6, tvb, offset + IP6H_DST, 16);
TVB_SET_ADDRESS(&pinfo->dst, AT_IPv6,     tvb, offset + IP6H_DST, 16);

ipv6_item = proto_tree_add_item(tree, proto_ipv6, tvb, offset, -1, ENC_NA);
ipv6_tree = proto_item_add_subtree(ipv6_item, ett_ipv6);
```

Figura 5.2.23– Código de Wireshark b)

Wireshark utiliza el valor del campo *Payload Length* para reconocer cuánta información debe leer en la cabecera. Al encontrarse un valor de *Payload Length* igual a 0, el programa cree no tener más información que examinar y provoca un error de lectura del paquete. Con esta modificación, al encontrarse con un valor 0 en el campo *Payload Length*, *Wireshark* cree tener un *Payload Length* de 70,000 bytes y lee el paquete correctamente. Este método es un pequeño *hack* que no soluciona el problema de que *Wireshark* no lee datos del paquete más allá de los primeros 65,535 bytes, pero sí es suficiente para examinar las cabeceras de los primeros 65,535 bytes de un paquete *Jumbogram*.

Una vez modificado el código, o simplemente partiendo del código de *Wireshark* que acompaña a este proyecto, se procede a su compilación con los siguientes comandos en el directorio raíz de *Wireshark*:

./autogen.sh

./configure --with-gtk2 (en lugar de *gtk2* se puede utilizar *gtk3* o *qt*, si se dispone de las librerías adecuadas)

make

make install

Una vez preparado el programa, se ejecuta *Wireshark* con el comando:

./wireshark

5.3 Manual de mantenimiento

5.3.1 Doxygen

El código de *NS3* está documentado con sintaxis *doxygen*, la cual ha sido ampliada con los módulos y métodos añadidos en este proyecto. Esta documentación no ha sido incluida en estos anexos debido a su gran extensión y la falta de funcionalidad que tendría en un documento como el actual. Sin embargo, puede ser consultada en el directorio

Documentación Doxygen del CD del proyecto, o puede generarse manualmente a partir del código fuente del proyecto de la siguiente forma:

- Acceder al directorio /doc/ del proyecto y configurar el fichero de configuración *doxygen.conf*, aunque se puede dejar la configuración por defecto.

- Acceder al directorio raíz del proyecto y ejecutar el comando *./waf --doxygen*.

Este último comando generará toda la documentación *doxygen* en base al código fuente del proyecto y puede ser accedida fácilmente utilizando un navegador web. Se recomienda acceder a ella a través del directorio *index.html*:

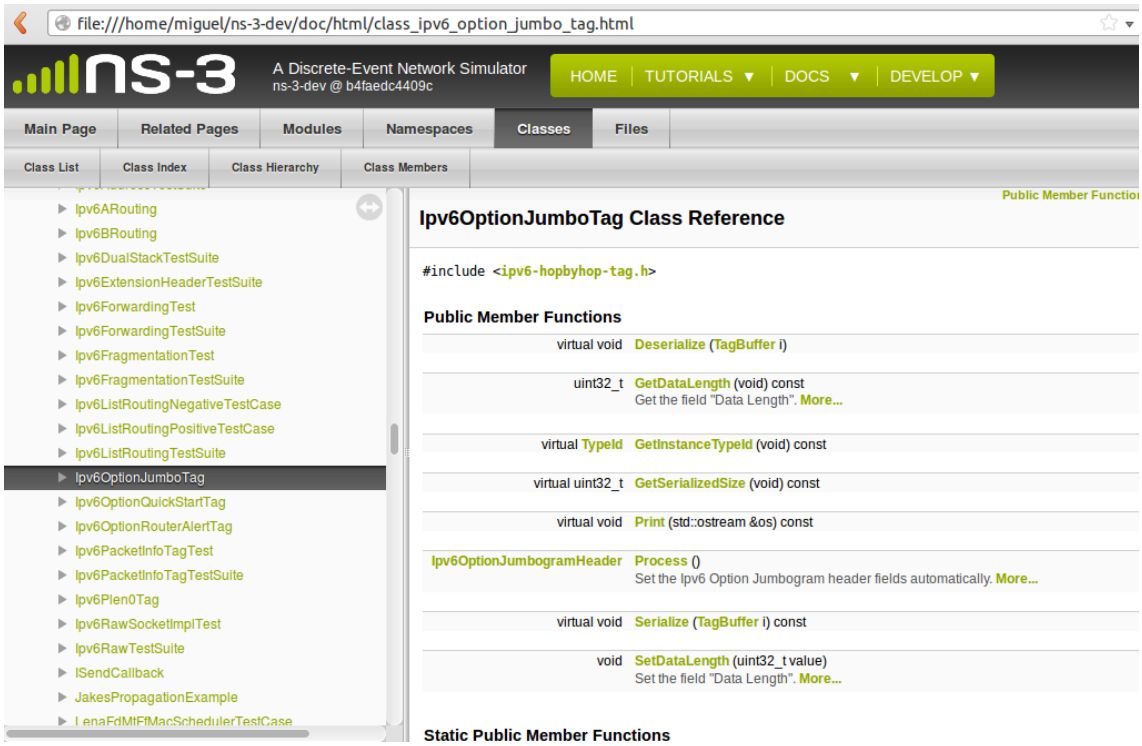

Figura 5.3.1 – Documentación Doxygen

Toda esta documentación proporciona información sobre las clases y métodos implementados en *NS3*, incluyendo el nuevo código programado en este proyecto, y sirve de referencia a futuros programadores del simulador que quieran revisar el código o modificarlo, al igual que ha sido utilizada esta documentación para la realización de este proyecto.

La documentación *Doxygen* almacenada en el CD del proyecto no contiene imágenes ni gráficos, debido a que su gran extensión excede el tamaño máximo de almacenamiento

del dispositivo. Para acceder a la documentación *Doxygen* completa es necesario generarla manualmente mediante los comandos mostrados en este apartado.

5.3.2 Añadido de opciones nuevas

Cualquier programador puede acceder al código de *NS3* y añadir nuevas opciones *Hop by Hop* al simulador, ya sean opciones estandarizadas u opciones creadas libremente por el usuario. Se muestran en este apartado los pasos a seguir para implementar una nueva opción en *NS3*. Esta opción tendrá como nombre para este manual el de *Example*, siendo el equivalente a lo que podría ser *Router Alert*, *Quick Start* o *Jumbogram*.

Para añadir esta nueva opción *Hop by Hop Example*, es necesario hacer cambios en los siguientes módulos:

- */src/internet/model/ipv6-option.cc*
- */src/internet/model/ipv6-option.h*
- */src/internet/model/ipv6-option-header.cc*
- */src/internet/model/ipv6-option-header.h*
- */src/internet/model/ipv6-l3-protocol.cc*

Empezando por el primero de ellos, */src/internet/model/ipv6-option.cc*, habría que implementar los métodos mostrados a continuación, simplemente añadiendo el código posteriormente a las opciones ya implementadas:

```
1 NS_OBJECT_ENSURE_REGISTERED (Ipv6OptionExample);
2
3 TypeId Ipv6OptionJumbogram::GetTypeId ()
4 {
5   static TypeId tid = TypeId ("ns3::Ipv6OptionExample")
6     .SetParent<Ipv6Option> ()
7     .AddConstructor<Ipv6OptionExample> ()
8   ;
9   return tid;
10 }
11
12 Ipv6OptionExample::Ipv6OptionExample ()
13 {
14   NS_LOG_FUNCTION_NOARGS ();
15 }
16
17 Ipv6OptionExample::~Ipv6OptionExample ()
18 {
19   NS_LOG_FUNCTION_NOARGS ();
20 }
```

Figura 5.3.2 – Código Ipv6OptionExample a)

Las líneas de la figura 5.3.2 hacen referencia al constructor de la opción. En principio, estos métodos deben ser implementados tal y como aparecen en esta plantilla, sustituyendo la palabra *Example* por el nombre de la opción programada.

```
22 uint8_t Ipv6OptionExample::GetOptionNumber () const
23 {
24   NS_LOG_FUNCTION_NOARGS ();
25
26   return OPT_NUMBER;
27 }
28
```

Figura 5.3.3 – Código Ipv6OptionExample b)

En la figura 5.3.3 se muestra el código del método que devuelve el campo *Option number* de la opción, el cual está definido en la cabecera */src/internet/model/ipv6-option.h*.

```
29 uint8_t Ipv6OptionExample::Process (Ptr<Packet>& packet, uint8_t offset, Ipv6Header const& ipv6Header, bool& isDropped)
30 {
31   NS_LOG_FUNCTION (this << packet << offset << ipv6Header << isDropped);
32
33   Ptr<Packet> p = packet->Copy ();
34
35   p->RemoveAtStart (offset);
36   Ipv6OptionExampleHeader exampleHeader;
37   p->RemoveHeader (exampleHeader);
38
39   NS_LOG_LOGIC ("Ipv6 Option Example received and being processed on node " << GetNode ()->GetId ());
40
41   Ipv6OptionExampleTag exampleTag;
42   exampleTag.SetParameter(parameter);
43   packet->AddPacketTag (exampleTag);
44
45
46   return exampleHeader.GetSerializedSize ();
47 }
```

Figura 5.3.4 – Código Ipv6OptionExample c)

El método de la figura 5.3.4, *Ipv6OptionExample*, es el más interesante, ya que es el que puede ser personalizado por el programador. En la figura sólo se muestran las líneas que deben ser implementadas de forma obligatoria para ser compatibles con este proyecto, pero es importante destacar que este método será invocado a cada paso del paquete por un nodo. Por lo tanto, todas las características específicas de procesamiento de una opción deberían ser implementadas aquí, ya sea el reconocimiento de posibles errores de formato o la necesidad de realizar alguna acción en particular cada vez que un paquete con la opción programada llega a un nodo.

Siguiendo con la figura 5.3.4, desde la línea 33 a la línea 37 se crea una copia del paquete original y se obtiene la cabecera de la opción.

Las líneas 41-43 crean un *Hop by Hop Tag* de la opción *Example* y es añadido al paquete original (no a la copia) para que en el siguiente envío del paquete la opción sea añadida de nuevo, ya que toda la cabecera de extensión *Hop by Hop* será eliminada una vez terminado el procesamiento. Es importante que este *Packet Tag* sea añadido al paquete antes de salir del método *Process*, ya sea añadiéndolo en el código de este método o desde otra función llamada desde el mismo método *Process*.

La línea 46 es de implementación obligatoria, ya que devuelve el tamaño de la cabecera procesada y es esencial para un correcto procesamiento de la extensión *Hop by Hop* en su conjunto.

Siguiendo con la cabecera del anterior módulo, */src/internet/model/ipv6-option.h*, el código para la opción de ejemplo quedaría como en la siguiente figura, suponiendo *200* como el número de la opción:

```
1 class Ipv6OptionExample : public Ipv6Option
2 {
3 public:
4
5   static const uint8_t OPT_NUMBER = 200;
6
7
8   static TypeId GetTypeId ();
9
10
11  Ipv6OptionExample ();
12
13  ~Ipv6OptionExample ();
14
15  virtual uint8_t GetOptionNumber () const;
16
17  virtual uint8_t Process (Ptr<Packet>& packet, uint8_t offset, Ipv6Header const& ipv6Header, bool& isDropped)
18 };
19
20
```

Figura 5.3.5 – Código Ipv6OptionExample d)

El siguiente paso es añadir el código que creará las cabeceras de la opción *Example*. Empezando con el módulo */src/internet/model/ipv6-option-header.cc*, habría que añadir el código de las siguientes figuras:

```
1 NS_OBJECT_ENSURE_REGISTERED (Ipv6OptionExampleHeader);
2
3 TypeId Ipv6OptionExampleHeader::GetTypeId ()
4 {
5   static TypeId tid = TypeId ("ns3::Ipv6OptionExampleHeader")
6     .AddConstructor<Ipv6OptionExampleHeader> ()
7     .SetParent<Ipv6OptionHeader> ()
8   ;
9   return tid;
10 }
11
12 TypeId Ipv6OptionExampleHeader::GetInstanceTypeId () const
13 {
14   return GetTypeId ();
15 }
16
17 Ipv6OptionExampleHeader::Ipv6OptionExampleHeader ()
18 {
19   SetType (200);
20   SetLength (4);
21 }
22
23 Ipv6OptionExampleHeader::~Ipv6OptionExampleHeader ()
24 {
25 }
```

Figura 5.3.6 – Código Ipv6OptionHeaderExample a)

En la figura 5.3.6 se muestran los métodos principales de la cabecera. Es importante definir correctamente los parámetros del constructor, en el que *SetType (200)* fija el número de opción a *200* en la cabecera y *SetLength (4)* define el valor del campo *Length* de la opción con *4*.

```
27 void Ipv6OptionExampleHeader::SetParameter (uint32_t parameter)
28 {
29   m_parameter = parameter;
30 }
31
32 uint32_t Ipv6OptionExampleHeader::GetParameter () const
33 {
34   return m_parameter;
35 }
```

Figura 5.3.7 – Código Ipv6OptionHeaderExample b)

La figura 5.3.7 comprende los métodos que deben ser implementados por cada campo de la cabecera de la opción adicionales a los campos *Option Type* y *Option Length*. Debe haber un método para asignar un valor al campo y otro para obtenerlo.

```
37 void Ipv6OptionExampleHeader::Print (std::ostream &os) const
38 {
39   os << "( type = " << (uint32_t)GetType () << " length = " << (uint32_t)GetLength ();
40 }
41
42 uint32_t Ipv6OptionExampleHeader::GetSerializedSize () const
43 {
44   return GetLength () + 2;
45 }
46
47 void Ipv6OptionExampleHeader::Serialize (Buffer::Iterator start) const
48 {
49   Buffer::Iterator i = start;
50
51   i.WriteU8 (GetType ());
52   i.WriteU8 (GetLength ());
53   i.WriteHtonU32 (m_parameter);
54 }
```

Figura 5.3.8 – Código Ipv6OptionHeaderExample c)

La figura 5.3.8 comienza sus cuatro primeras líneas de código con un método auxiliar pero de obligatoria implementación que muestra información relativa a la cabecera, la cual es libremente personalizable por el programador.

El siguiente método, *GetSerializedSize( )*, devuelve el tamaño en bytes de la cabecera, el cual incluye tanto los campos fijos *Option Type* y *Option Length* como los campos adicionales programados para la opción.

El método de las últimas líneas es *Serialize*, el cual debe ser programado cuidadosamente. Se parte de un *buffer* en la variable *i*, el cual debe ser rellenado con los valores asignados a cada campo en el orden en el que aparecen en el paquete, empezando siempre por *Option Type*. Es importante rellenar el *buffer* hasta el tamaño máximo fijado en el método *GetSerializedSize( )*, ya que intentar escribir más allá de este límite provocará un error en ejecución en el simulador.

```
56 uint32_t Ipv6OptionExampleHeader::Deserialize (Buffer::Iterator start)
57 {
58   Buffer::Iterator i = start;
59
60   SetType (i.ReadU8 ());
61   SetLength (i.ReadU8 ());
62   m_parameter = i.ReadNtohU32 ();
63
64   return GetSerializedSize ();
65 }
66
67 Ipv6OptionHeader::Alignment Ipv6OptionExampleHeader::GetAlignment () const
68 {
69   return (Alignment){ 4,2};
70 }
```

Figura 5.3.9 – Código Ipv6OptionHeaderExample d)

La figura 5.3.9 comienza con el método *Deserialize*, el cual tiene la función contraria del anterior método *Serialize*. Su cometido es el de leer el contenido del paquete, asignar

los valores leídos a las variables privadas de la clase y devolver finalmente el tamaño en bytes de la opción.

El último método devuelve las condiciones de alineamiento de la opción dentro de la cabecera de extensión *Hop by Hop*, las cuales siguen las directrices marcadas en la *RFC 2460*.

Fijándonos ahora en el módulo */src/internet/model/ipv6-option-header.h*, sería necesario añadir un código como el de la siguiente plantilla:

```
1 class Ipv6OptionExampleHeader : public Ipv6OptionHeader
2 {
3 public:
4   static TypeId GetTypeId ();
5   virtual TypeId GetInstanceTypeId () const;
6   Ipv6OptionExampleHeader ();
7   virtual ~Ipv6OptionExampleHeader ();
8   void SetParameter (uint32_t parameter);
9   uint32_t GetParameter () const;
10  virtual void Print (std::ostream &os) const;
11  virtual uint32_t GetSerializedSize () const;
12  virtual void Serialize (Buffer::Iterator start) const;
13  virtual uint32_t Deserialize (Buffer::Iterator start);
14  virtual Alignment GetAlignment () const;
15
16 private:
17
18  uint32_t m_parameter;
19 };
```

Figura 5.3.10 – Código Ipv6OptionHeaderExample e)

La última modificación debe hacerse en el módulo */src/internet/model/ipv6-l3-protocol.cc*, en su método *Ipv6L3Protocol::RegisterOptions*, donde se ha de añadir las líneas correspondientes a esta nueva opción llamada *Example*, tal y como se indica en la siguiente captura:

```
void Ipv6L3Protocol::RegisterOptions ()
{
  Ptr<Ipv6OptionDemux> ipv6OptionDemux = CreateObject<Ipv6OptionDemux> ();
  ipv6OptionDemux->SetNode (m_node);

  Ptr<Ipv6OptionPad1> pad1Option = CreateObject<Ipv6OptionPad1> ();
  pad1Option->SetNode (m_node);
  Ptr<Ipv6OptionPadn> padnOption = CreateObject<Ipv6OptionPadn> ();
  padnOption->SetNode (m_node);
  Ptr<Ipv6OptionJumbogram> jumbogramOption = CreateObject<Ipv6OptionJumbogram> ();
  jumbogramOption->SetNode (m_node);
  Ptr<Ipv6OptionRouterAlert> routerAlertOption = CreateObject<Ipv6OptionRouterAlert> ();
  routerAlertOption->SetNode (m_node);
  Ptr<Ipv6OptionQuickStart> quickStartOption = CreateObject<Ipv6OptionQuickStart> ();
  quickStartOption->SetNode (m_node);
  Ptr<Ipv6OptionExample> exampleOption = CreateObject<Ipv6OptionExample> ();
  exampleOption->SetNode (m_node);

  ipv6OptionDemux->Insert (pad1Option);
  ipv6OptionDemux->Insert (padnOption);
  ipv6OptionDemux->Insert (jumbogramOption);
  ipv6OptionDemux->Insert (routerAlertOption);
  ipv6OptionDemux->Insert (quickStartOption);
  ipv6OptionDemux->Insert (exampleOption);

  m_node->AggregateObject (ipv6OptionDemux);
}
```

Figura 5.3.11 – Código Ipv6L3Protocol::RegisterOptions

Este paso es necesario para que el procesamiento de opciones del método *Ipv6Extension::ProcessOptions* tenga en cuenta la nueva opción programada.

5.3.3 Añadido de tags

Este apartado pretende explicar cómo añadir un *Hop by Hop Tag* para una nueva opción de ejemplo a la que se le pondrá el nombre de *Example*. Este proceso requiere la modificación de los siguientes módulos de *NS3*:

- *src/internet/model/ipv6-hopbyhop-tag.cc*
- *src/internet/model/ipv6-hopbyhop-tag.h*

En relación al primer fichero, *src/internet/model/ipv6-hopbyhop-tag.cc*, las siguientes capturas muestran una plantilla de lo que podría ser *Hop by Hop Tag* para la nueva opción *Example*:

```
1 NS_OBJECT_ENSURE_REGISTERED (Ipv6OptionExampleTag);
2
3 TypeId Ipv6OptionExampleTag::GetTypeId ()
4 {
5   static TypeId tid = TypeId ("ns3::Ipv6OptionExampleTag")
6     .SetParent<Tag> ()
7     .AddConstructor<Ipv6OptionExampleTag> ()
8   ;
9   return tid;
10 }
11
12 TypeId
13 Ipv6OptionExampleTag::GetInstanceTypeId (void) const
14 {
15   return GetTypeId ();
16 }
```

Figura 5.3.12 – Código Ipv6-HopbyHop-Tag a)

La figura 5.3.12 muestra los dos métodos que definen el tipo de objeto para el nuevo *Packet Tag* específico para la opción *Example*, el cual es un objeto que hereda directamente los métodos de la clase *Packet Tag* ya implementada en *NS3*.

```
17 uint32_t
18 Ipv6OptionExampleTag::GetSerializedSize (void) const
19 {
20   return 4;
21 }
22 void
23 Ipv6OptionExampleTag::Serialize (TagBuffer i) const
24 {
25   i.WriteU32 (m_parameter);
26 }
27 void
28 Ipv6OptionExampleTag::Deserialize (TagBuffer i)
29 {
30   m_parameter = i.ReadU32();
31 }
```

Figura 5.3.13 – Código Ipv6-HopbyHop-Tag b)

En la figura 5.3.13, el primer método *GetSerializedSize* devuelve el tamaño de la información almacenada por el *Packet Tag*, la cual irá acompañando al paquete donde dicho *Tag* esté asignado. Este tamaño viene dado en bytes y debe coincidir con el tamaño con el que es rellenado el *buffer* en el método *Serialize*, justo debajo del anterior, en el cual datos como los valores de los campos que serán añadidos a la cabecera irán almacenados.

El método *Serialize* requiere de otro método, *Deserialize*, que se encarga del método contrario, es decir, de leer la información almacenada en el buffer y almacenarla en variables para que puedan ser utilizadas por otros métodos.

```
32 void
33 Ipv6OptionExampleTag::Print (std::ostream &os) const
34 {
35
36 }
37
38 void
39 Ipv6OptionExampleTag::SetParameter (uint32_t parameter)
40 {
41   m_dataLength = value;
42 }
43 uint32_t
44 Ipv6OptionExampleTag::GetParameter (void) const
45 {
46   return m_parameter;
47 }
48
```

Figura 5.3.14 – Código Ipv6-HopbyHop-Tag c)

La función *Print* de la figura 5.3.14 es de obligada implementación en la clase *Hop by Hop Tag*. Está destinada a mostrar información sobre el *Packet Tag* implementado y es personalizable por el programador.

Los métodos *SetParameter* y *GetParameter* son los encargados de definir el valor de un campo a la hora de utilizar un *Hop by Hop Tag*. Deberán ser implementados tantos métodos de este tipo como argumentos quieran almacenarse en el *Packet Tag* para su posterior utilización en el añadido de opciones a la cabecera de *Ipv6*.

```
50 Ipv6OptionExamplegramHeader
51 Ipv6OptionExampleTag::Process()
52 {
53  NS_LOG_INFO ("Procesador Examplegram\n");
54  Ipv6OptionExamplegramHeader Example;
55  Example.SetParameter(this->m_parameter);
56  return Example;
57 }
```

Figura 5.3.15 – Código Ipv6-HopbyHop-Tag d)

El último método, *Process*, mostrado en la figura 5.3.15, es utilizado por *Ipv6L3Protocol::CheckHopbyhopTags* para construir una cabecera de la opción relativa al *Packet Tag* en base a los parámetros previamente almacenados con los métodos de tipo *SetParameter*.

En relación ahora con el módulo */src/internet/model/ipv6-hopbyhop-tag.h*, siendo ésta la cabecera del anterior fichero descrito, quedaría su plantilla como la de la figura 5.3.16:

```
1 class Ipv6OptionExampleTag : public Tag
2 {
3 public:
4
5   virtual TypeId GetInstanceTypeId (void) const;
6   virtual uint32_t GetSerializedSize (void) const;
7   virtual void Serialize (TagBuffer i) const;
8   virtual void Deserialize (TagBuffer i);
9   virtual void Print (std::ostream &os) const;
10  static TypeId GetTypeId ();
11  void SetParameter (uint32_t parameter);
12  uint32_t GetParameter (void) const;
13  Ipv6OptionExamplegramHeader Process();
14
15  private:
16
17  uint32_t m_parameter;
18
19 };
```

Figura 5.3.16 – Código Ipv6-HopbyHop-Tag e)

5.3.4 Mercurial / TortoiseHG

El proyecto *NS3* utiliza el software *Mercurial* para llevar el registro de los cambios que se van realizando en el código del programa. Es por este motivo que en el desarrollo de este proyecto se ha mantenido el uso de *Mercurial* y los cambios que han sufrido el código original de *NS3* y los nuevos ficheros añadidos por este proyecto han sido registrados y pueden ser consultados utilizando tanto *Mercurial* como alguna aplicación con interfaz gráfica que sea compatible con este software.

Utilizando el programa *TortoiseHG* es fácil ver los cambios que ha sufrido el código original, como en la siguiente figura, en la que se compara el módulo */src/internet/model/ipv6-l3-protocol.cc* original con la última versión modificada por este proyecto, en el que se añade el código que elimina la extensión *Hop by Hop* del paquete en el método *Ipv6L3Protocol::Receive*:

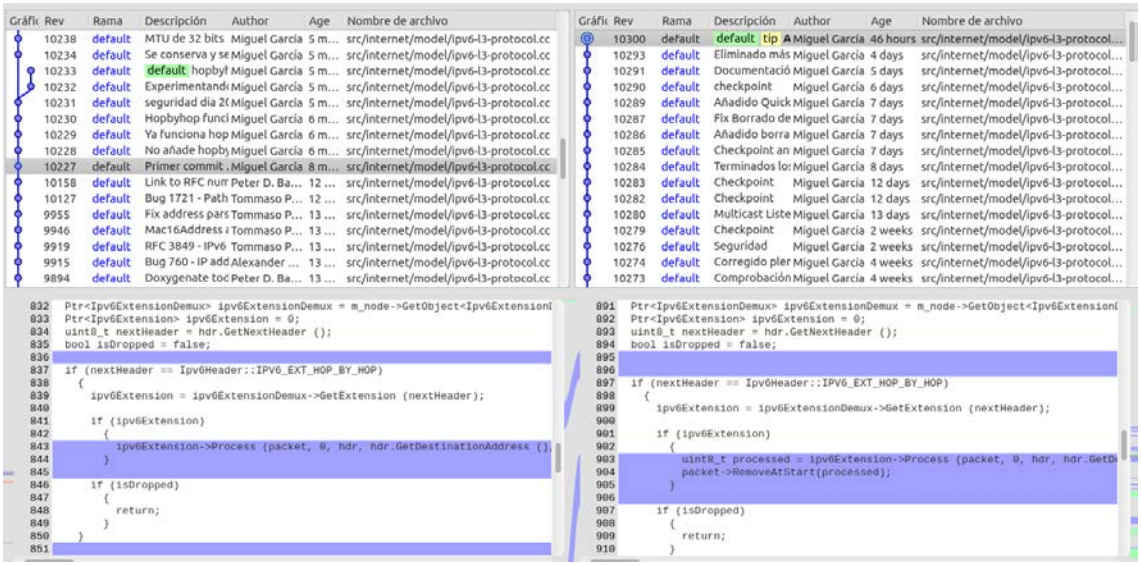

Figura 5.3.17 – Captura Software TortoiseHG