Tutorial para la simulación de algoritmos de ruteo en redes alámbricas e inalámbricas utilizando ns-3

Versión del Documento: 1.0.0

Cristian Duran-Faundez, Jannyna Etter, Cristhian Aguilera
Universidad del Bío-Bío, Concepción, Chile
crduran@ubiobio.cl, jetter@alumnos.ubiobio.cl, cristhia@ubiobio.cl

29 de febrero de 2016

Resumen

Índice

1. Instalación de ns-3 **2**
1.1. Instalación de los prerequisitos 2
1.2. Instalación de ns-3 3
1.3. Comprobación de la instalación 6

2. Primeros pasos: Ejecución de las primeras simulaciones **6**

3. Simulación de un algoritmo de ruteo **11**

4. El caso inalámbrico **16**

5. La simulación de un algoritmo de ruteo desde cero **25**
5.1. Implementación del algoritmo de ruteo 26
5.1.1. `discovery-routing-protocol.cc` y `.h` 26
5.1.2. `discovery-id-cache.cc` y `.h` 48
5.1.3. `discovery-rtable.cc` y `.h` 50
5.1.4. `discovery-packet.cc` y `.h` 55
5.1.5. `discovery.cc` 60

Proyecto FDD2011-02: : Tutorial para la simulación de algoritmos de ruteo en redes alámbricas e inalámbricas utilizando ns-3

Hoy en día, las tecnologías de información y comunicación (y sobre todo aquellas relacionadas con la Internet) están teniendo un impacto visible en nuestra vida cotidiana, y en el quehacer profesional en los más diversos campos de aplicación. Desde el punto de vista académico, este impacto ha provocado que muchas carreras tecnológicas incluyan en sus mallas curriculares asignaturas (sean obligatorias o electivas) ligadas a tópicos de computación y a las redes de computadoras. En el primero de los casos es regularmente sencillo (o factible) llevar la teoría a la práctica. La disponibilidad de computadoras en los centros de estudio es amplia, por lo que la formación práctica de alumnos y profesionales en tópicos como utilización de software de ofimática, diseño y construcción de software, sistemas operativos, bases de datos, y otros, es común y forma parte integral y obligatoria de las

asignaturas. Para el segundo de los casos mencionados, la historia es un poco distinta.

Las asignaturas de redes de computadores se basan por lo general en la explicación de estándares y protocolos de comunicación, utilizando como referencia el modelo de referencia OSI. Este modelo agrupa en siete capas las distintas normas y protocolos utilizados en las redes de computadoras, partiendo con la identificación de los estándares de nivel físico (capa 1) como normas eléctricas, equipos, cableado, y otros, y terminando con los protocolos utilizados a nivel de aplicaciones (capa 7). En algunos tópicos de las asignaturas, si bien no es siempre posible, es factible generar algunos talleres. Por ejemplo, en la explicación de la primera capa, se puede siempre complementar con un taller de armado de cableado o un taller de interconexión de dispositivos (si se cuenta con el hardware necesario). En otras capas, la utilización de software sniffer (analizadores de paquetes) es también posible.

Uno de los tópicos fundamentales está en lo que se refiere a la interconexión de varias redes, y a los mecanismos de ruteo utilizados. Ésto es, la explicación de no sólo qué tipos de topologías pueden existir y cuáles son las características de los distintos tipos de redes, sino también el conocimiento de las estrategias que permiten hacer llegar un paquete de datos desde un computador a otro que no están necesariamente conectados el uno al lado del otro. Tal como funciona en Internet, en varios casos, los datos deben pasar por varios equipos intermedios, los que ejecutan (en muchos casos) algoritmos distribuidos que permiten esta comunicación (saber por dónde debe pasar un mensaje para ir de una parte a otra, considerando condiciones del canal, congestión, y otros, y con una visión muchas veces reducida de la topología global).

Lamentablemente, llevar esta teoría a la práctica es en muchas ocaciones poco factible, pues implica la adquisición de una gran cantidad y capacidad de equipamiento. Si bien la Universidad cuenta con redes de computadoras, no se puede hacer mucho si no se puede intervenir (además, la mayor parte de las unidades no cuentan con equipamiento mínimo). En este sentido, mucho del aprendizaje de los tópicos prácticos de redes (por ejemplo: los mencionados problemas de ruteo), el análisis de múltiples configuraciones y el desarrollo de nuevas estrategias que pudiesen concebirse producto de la curiosidad de un alumno o profesor, o con motivos de investigación, requieren muchas veces de herramientas informáticas que permitan simular el comportamiento de sistemas reales. Para el caso de las redes de computadoras, una de estas herramientas de simulación es el software ns-3.

ns-3 es un simulador de redes a eventos discretos, que fue concebido principalmente para uso educacional. Es un software libre diseñado bajo la licencia GNU GPLV2, y está publicamente disponible para investigación, desarrollo y uso. (`http://www.nsnam.org/`)

En nuestra Universidad, el desarrollo y la investigación en redes de computadoras es bastante pequeño, y existe un desconocimiento casi generalizado de este tipo de herramientas de simulación para redes. Sin embargo, en Internet existe una amplia comunidad de desarrollo y muchas Universidades prestigiosas a nivel mundial utilizan esta herramienta. Se ha generado una gran cantidad de documentación (por supuesto, la mayor parte en inglés). Este proyecto tiene como principal meta la concepción de un documento (tutorial), integrando teoría y ejemplos de aplicación, que permita dar al lector una comprensión rápida y práctica de la utilización del simulador ns-3, de forma de evitarle una búsqueda demasiado extensa, y que le permita abordar rapidamente los problemas asignados en las asignaturas (y proyectos en general) de una forma distinta y mucho más completa a la teórica convensional, ya que permitirá la visualización y el análisis rápido de factores difíciles de resolver en el cuaderno a gran escala (es decir, cuando se integran grandes redes con grandes tasas de transferencia).

Si bien ns-3 permite la simulación de modelos de estándares de diferentes capas, el documento a generar se centrará (y por motivos de simplificación de la búsqueda y para asegurar su calidad) en la expilcación de diversos protocolos de ruteo para redes alámbricas e inalámbricas (por ser uno de los tópicos con menos posibilidad de ejercitación práctica, y uno de los más complejos de desarrollar y analizar en el cuaderno). Se tomará como referencia los programas aplicados en las asignaturas dictadas para la carrera de Ingeniería Civil en Automatización: Redes de Computadores (ramo de malla) y Redes Inalámbricas Industriales (ramo electivo). Esto es sin pérdida de generalidad, ya que los ramos de redes de computadoras son similares en las diferentes carreras y facultades.

Se espera que este documento sea una guía amigable y útil que pueda ser compartida a la comunidad universitaria relacionada con la materia. Por supuesto, la experiencia generada será útil para la indagación futura en otros aspectos y capas protocolares de los modelos de redes.

1. Instalación de ns-3

Esta sección resume los pasos seguidos desde el tutorial de ns-3 disponible en `http://www.nsnam.org/wiki/index.php/Insta`
En este tutorial, se utiliza GNU Linux y Debian

Nota: Para distintos Sistemas Operativos y distribuciones, los requisitos de instalación pueden variar sutilmente. En este tutorial se utilizará, como sistema operativo de referencia, GNU Linux de una distribución Debian.*** Para información acerca de otras plataformas, diríjase al tutorial de ns-3, en la sección Getting Started.

1.1. Instalación de los prerequisitos

En una primera instancia, se deben instalar los paquetes necesarios para desarrollar aplicaciones con C, C++ y Phyton. La instalación de los diferentes paquetes puede realizarse con el comando `apt-get` escribiendo en una terminal:

```
1 $ sudo apt-get install <nombre-paquete>
```

donde `<nombre-paquete>` es, evidentemente, el nombre del paquete a instalar. Lo mismo se puede realizar con el programa `aptitude`.

La lista de paquetes prerequisitos para la instalación y el trabajo con ns-3 es la siguiente:

```
gcc g++ python python-dev mercurial bzr gdb valgrind gsl-bin libgsl0-dev libgsl0ldbl flex
bison 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*
```

La construcción de módulos necesarios en este último paso (instalación de MPI) es la etapa que más tiempo toma de este proceso (esto puede tardar unos pocos minutos).

1.2. Instalación de ns-3

Los códigos de ns-3 están disponibles en los repositorios de Mercurial. Antes de descargar el simulador, se debe crear una carpeta llamada para guardar el código fuente, ejemplos, la API, y otros recursos de ns-3. Siguiendo el tutorial oficial de ns-3, para este fin se crea una carpeta llamada `repos` bajo el repositorio ***`/home/username/`*** con el comando `mkdir repos`. Luego, entrar a la carpeta haciendo `cd repos` y ejecutar la siguiente línea:

```
1 hg clone http://code.nsnam.org/ns-3-allinone
```

Debería obtenerse la siguiente salida:

```
1 destination directory: ns-3-allinone
2 requesting all changes
3 adding changesets
4 adding manifests
5 adding file changes
6 added 56 changesets with 83 changes to 7 files
7 updating to branch default
8 7 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Si no reportan errores, dentro de repos debería haberse creado una carpeta con el nombre `ns-3-allinone`. Ahora, ingresar a la carpeta `ns-3-allinone` con el comando `cd`. Dentro de la carpeta se encontrarán los siguientes archivos:

```
1 . .. build.py constants.py dist.py download.py .hg .hgignore README util.
  py
```

Para descargar ns-3 ejecutar la siguiente línea:

```
1 ./download.py -n ns-3-dev
```

Debería obtener la siguiente salida:

```
1     #
2     # Get NS-3
3     #
4
5 Cloning ns-3 branch
6  => hg clone http://code.nsnam.org/ns-3-dev ns-3-dev
7 requesting all changes
8 adding changesets
9 adding manifests
10 adding file changes
11 added 9977 changesets with 44751 changes to 6959 files
```

```
12 updating to branch default
13 2729 files updated, 0 files merged, 0 files removed, 0 files unresolved
14
15     #
16     # Get PyBindGen
17     #
18
19 Required pybindgen version:  0.16.0.825
20 Trying to fetch pybindgen; this will fail if no network connection is available.
     Hit Ctrl-C to skip.
21  => bzr checkout -rrevno:825 https://launchpad.net/pybindgen pybindgen
22 Fetch was successful.
23
24     #
25     # Get NetAnim
26     #
27
28 Required NetAnim version:  netanim-3.103
29 Retrieving NetAnim from http://code.nsnam.org/netanim
30  => hg clone http://code.nsnam.org/netanim netanim
31 requesting all changes
32 adding changesets
33 adding manifests
34 adding file changes
35 added 190 changesets with 1007 changes to 117 files
36 updating to branch default
37 107 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Ahora, se debe contruir ns-3. Esto se hace con el programa build.py (descargado anteriormente con el comando hg) haciendo:

```
1  ./build.py --enable-examples --enable-tests
```

Con esto comienza la construcción de la herramienta, con lo que se verá una serie de mensajes mostrando el estado del proceso. Esto puede tomar algunos minutos. El proceso debería terminar mostrando en la salida las siguiente líneas:

⋮

```
1  [2096/2098] cxx: src/tap-bridge/model/tap-creator.cc -> build/src/tap-bridge/model/
     tap-creator.cc.4.o
2  [2097/2098] cxx: src/tap-bridge/model/tap-encode-decode.cc -> build/src/tap-bridge/
     model/tap-encode-decode.cc.4.o
3  [2098/2098] cxxprogram: build/src/tap-bridge/model/tap-creator.cc.4.o build/src/tap
     -bridge/model/tap-encode-decode.cc.4.o -> build/src/tap-bridge/ns3-dev-tap-
     creator-debug
4  Waf: Leaving directory `/home/crduran/repos/ns-3-allinone/ns-3-dev/build'
5  'build' finished successfully (8m50.000s)
6
7  Modules built:
8  antenna                    aodv                       applications
9  bridge                     buildings                  config-store
10 core                       csma                       csma-layout
11 dsdv                       dsr                        emu
```

```
12 energy                fd-net-device          flow-monitor
13 internet              lte                    mesh
14 mobility              mpi                    netanim (no Python)
15 network               nix-vector-routing     olsr
16 point-to-point        point-to-point-layout  propagation
17 spectrum              stats                  tap-bridge
18 test (no Python)      topology-read          uan
19 virtual-net-device    visualizer             wifi
20 wimax
21
22 Modules not built (see ns-3 tutorial for explanation):
23 brite                 click                  openflow
24
25 Leaving directory `./ns-3-dev'
```

Con esto, la herramienta ns-3 está lista para operar.

Otra alternativa para construir ns-3 es utilizando la herramienta Waf. Esta permite, entre otras cosas, construir una versión optimizada del código ya que, según el tutorial, ns-3 viene por defecto configurado para construir versiones de depuración.

Para trabajar con estas versiones optimizadas de los ejemplos incluidos en ns-3, realizar lo siguiente:

Lo primero es ubicarse en la carpeta `ns-3-dev` que se encuentra en el directorio `ns-3-allinone`. Luego, ejecutar las siguientes líneas:

```
1 ./waf clean
2 ./waf -d optimized --enable-examples --enable-tests configure
```

Varias verificaciones se realizarán en ese momento. El proceso termina por mostrar un mensaje similar al siguiente en la salida:

```
1 'configure' finished successfully (8.419s)
```

Finalmente, ejecutar:

```
1 ./waf
```

Este proceso durará algunos minutos. Finalmente, la salida terminará por algo similar a lo que sigue:

⋮

```
1 [2096/2098] cxx: src/tap-bridge/model/tap-creator.cc -> build/src/tap-bridge/model/
    tap-creator.cc.4.o
2 [2097/2098] cxx: src/tap-bridge/model/tap-encode-decode.cc -> build/src/tap-bridge/
    model/tap-encode-decode.cc.4.o
3 [2098/2098] cxxprogram: build/src/tap-bridge/model/tap-creator.cc.4.o build/src/tap
    -bridge/model/tap-encode-decode.cc.4.o -> build/src/tap-bridge/ns3-dev-tap-
    creator-optimized
4 Waf: Leaving directory `/home/crduran/repos/ns-3-allinone/ns-3-dev/build'
5 'build' finished successfully (15m48.718s)
6
7 Modules built:
8 antenna               aodv                   applications
9 bridge                buildings              config-store
10 core                 csma                   csma-layout
11 dsdv                 dsr                    emu
12 energy               fd-net-device          flow-monitor
```

```
13 internet                 lte                     mesh
14 mobility                 mpi                     netanim (no Python)
15 network                  nix-vector-routing      olsr
16 point-to-point           point-to-point-layout   propagation
17 spectrum                 stats                   tap-bridge
18 test (no Python)         topology-read           uan
19 virtual-net-device       visualizer              wifi
20 wimax
21
22 Modules not built (see ns-3 tutorial for explanation):
23 brite                    click                   openflow
```

Para volver a la configuración por defecto (versión de 'debug'; aquella que teníamos antes de utilizar Waf), verificar el tutorial en línea (`http://www.nsnam.org/docs/release/3.14/tutorial/singlehtml/index.html`) en la sección Building with Waf.

Más detalles acerca del proceso de instalación de ns-3, la integración de otras herramientas (ej: Eclipse, QT-creator, Netbeans, etc.), y otra información útil, aparece en `http://www.nsnam.org/wiki/index.php/Installation`.

1.3. Comprobación de la instalación

Para comprobar la instalación realizada se utiliza el programa `test.py`. Para esto, entrar a la carpeta ns-3-dev y ejecutar:

```
1 ./test.py -c core
```

Este proceso es rápido, y debería concluir con una línea similar a la siguiente:

```
1 157 of 160 tests passed (157 passed, 3 skipped, 0 failed, 0 crashed, 0 valgrind
    errors)
```

2. Primeros pasos: Ejecución de las primeras simulaciones

El tutorial de ns-3 menciona una aplicación sencilla de prueba, la que permite verificar que se puede compilar y ejecutar una aplicación muy básica. Esta aplicación es un clásico "*Hola Mundo*", que en este caso entrega un "*Hola Simulador*". Ojo que esta aplicación sólo muestra el mensaje si estamos en la configuración de 'debug' (no en modo optimizado)[1].

Para ejecutar este ejemplo, debemos estar posicionados en la carpeta `ns-3-dev`. Para esto, ejecutar la siguiente línea de comando:

```
1 ./waf --run hello-simulator
```

Una vez hecho esto, se debería ver en la salida el mensaje "Hola mundo". Esto es sólo para probar que funciona ns-3. A continuación, se explicarán ejemplos sencillos de simulación de redes en ns-3.

Para realizar las primeras experiencias, el tutorial de ns-3 propone varios ejemplos de simulación sencillos utilizando medios cableados, inalámbricos y combinaciones de estos. En este tutorial se seguirán dos de estos ejemplos (más detalles en el turorial oficial de ns-3).

El primer ejemplo consiste en comunicar dos nodos directamente conectados n_0 y n_1. donde n_0 envía una solicitud de echo y n_1 responde. La Figura 1 ilustra este ejemplo.

[1] Tipear:
```
./waf clean
./waf -d debug --enable-examples --enable-tests configure
./waf
```

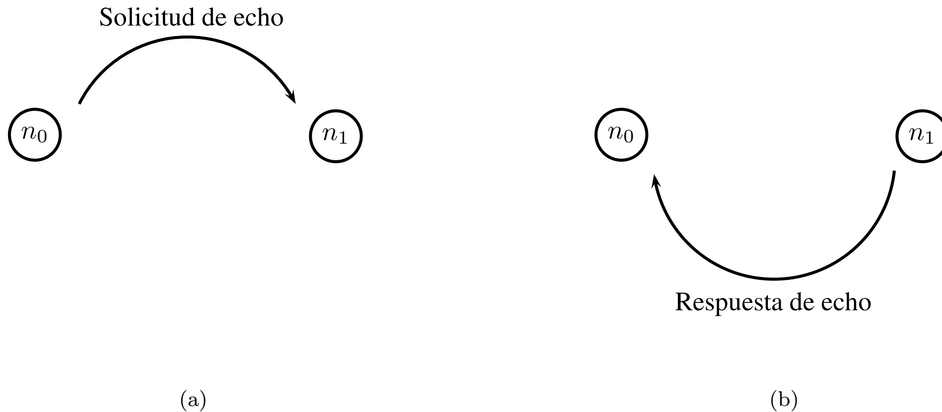

Figura 1: Ilustración de ejemplo en `first.cc`.

El ejemplo a utilizar se llama 'first' y se encuentra en la carpeta '`examples/tutorial/`' que se encuentra dentro de la carpeta `ns-3-dev`. Junto con first, en esta carpeta, se encuentran varios otros ejemplos.

El contenido del archivo `first.cc` es el siguiente:

Listado 1: Contenido del archivo `first.cc`.

```
1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2  /*
3   * This program is free software; you can redistribute it and/or modify
4   * it under the terms of the GNU General Public License version 2 as
5   * published by the Free Software Foundation;
6   *
7   * This program is distributed in the hope that it will be useful,
8   * but WITHOUT ANY WARRANTY; without even the implied warranty of
9   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
10  * GNU General Public License for more details.
11  *
12  * You should have received a copy of the GNU General Public License
13  * along with this program; if not, write to the Free Software
14  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
15  */
16
17 #include "ns3/core-module.h"
18 #include "ns3/network-module.h"
19 #include "ns3/internet-module.h"
20 #include "ns3/point-to-point-module.h"
21 #include "ns3/applications-module.h"
22
23 using namespace ns3;
24
25 NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
26
27 int
```

```
28 main (int argc, char *argv[])
29 {
30   LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
31   LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
32 
33   NodeContainer nodes;
34   nodes.Create (2);
35 
36   PointToPointHelper pointToPoint;
37   pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
38   pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
39 
40   NetDeviceContainer devices;
41   devices = pointToPoint.Install (nodes);
42 
43   InternetStackHelper stack;
44   stack.Install (nodes);
45 
46   Ipv4AddressHelper address;
47   address.SetBase ("10.1.1.0", "255.255.255.0");
48 
49   Ipv4InterfaceContainer interfaces = address.Assign (devices);
50 
51   UdpEchoServerHelper echoServer (9);
52 
53   ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
54   serverApps.Start (Seconds (1.0));
55   serverApps.Stop (Seconds (10.0));
56 
57   UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
58   echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
59   echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
60   echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
61 
62   ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
63   clientApps.Start (Seconds (2.0));
64   clientApps.Stop (Seconds (10.0));
65 
66   Simulator::Run ();
67   Simulator::Destroy ();
68   return 0;
69 }
```

A continuación se dará una explicación resumida del funcionamiento de este ejemplo.

Luego de los comentarios iniciales que contienen mensajes que conciernen a la licencia del simulador, aparece la declaración las librerías utilizadas en este programa (las líneas comenzadas por "`#include `", las que se ubican en el directorio "`build/ns3`" bajo la carpeta `ns-3-dev`. Estas librerías son, en realidad, llamadas a conjuntos de librerías más grandes donde se encuentra la declaración de las clases que componen los diferentes módulos del simulador.

La línea `using namespace ns3;` refiere a la utilización de un espacio de nombres donde encuentran todas las declaraciones relacionadas con ns-3 (más detalles en el tutorial oficial).

La línea `NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");` es una macro que define un nuevo componente "log", . Los mensajes que muestra ns-3 en pantalla (mensajes tipo log) se clasifican en 7 niveles distintos, donde

cada uno incrementa el nivel de detalle que se va a mostrar en la simulación. Los niveles de los mensajes log son los siguientes:

1. `NS_LOG_ERROR` — Log error messages;
2. `NS_LOG_WARN` — Log warning messages;
3. `NS_LOG_DEBUG` — Log relatively rare, ad-hoc debugging messages;
4. `NS_LOG_INFO` — Log informational messages about program progress;
5. `NS_LOG_FUNCTION` — Log a message describing each function called;
6. `NS_LOG_LOGIC` – Log messages describing logical flow within a function;
7. `NS_LOG_ALL` — Log everything.

La lista de macros disposibles se encuentra en `http://www.nsnam.org/doxygen/group__logging.html`. En este caso sea crea un nuevo componente con el nombre "`FirstScriptExample`"que podría ser más tarde deshabilitado o habilitado a través de los métodos `LogComponentDisable`, `LogComponentDisableAll`, `LogComponentEnable` y `LogComponentEnableAll`, según se requiera. Por defecto, los mensajes de log están deshabilitados.

Las líneas:

```
1  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
2  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

Habilitan los mensajes de log, de nivel `LOG_LEVEL_INFO`, de los componentes `UdpEchoClientApplication` y `UdpEchoServerApplic`

La línea `NodeContainer nodes;` declara un nuevo contenedor de nodos (llamado `nodes`) que corresponde a un vector que almacena punteros a objetos de la clase nodo. Luego, la línea `nodes.Create (2);` crea 2 nodos en el contenedor `nodes`.

La línea PointToPointHelper pointToPoint; declara un nuevo objeto "Helper" que crea redes punto a punto. Los objetos Helper son objetos que proveen métodos que facilitan la utilización de ciertas tareas, métodos, mecanismos o protocolos comunes que se llevan a cabo en una red de computadores, y que, de otra forma, serían complejos y engorrosos de aplicar. En este caso, se crea un nuevo Helper para implementar una comunicación Punto a Punto.

Las líneas:

```
1  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
2  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

Establecen distintos atributos de los dispositivos mismos y del canal, respectivamente. En este caso, se define que los dispositivos son capaces de transmitir a una tasa de 5Mbps. Además, se define que el retardo de propagación a través del canal será de 2ms.

La línea `NetDeviceContainer devices;` define un nuevo contenedor de dispositivos de red llamado devices, que contiene punteros a objetos de la clase `NetDevice`.

En la línea `devices = pointToPoint.Install (nodes);`, el método `Install`, para ambos nodos creados, crea un dispoisitivo de red punto a punto, le asigna una dirección MAC (automaticamente) y lo asocia al nodo correspondiente. Luego, se crea un canal punto a punto (una instancia de la clase `PontToPointChannel`) y asigna a los dos dispositivos creados. Estos dispositivos son devueltos en un objeto de la clase `NetDeviceContainer`, asignado en la línea de comando del ejemplo, al objeto `devices`.

La línea `InternetStackHelper stack;` crea un objeto de la clase `InternetStackHelper` el que, por defecto, agrega los protocolos IP, ARP, UDP, TCP e ICMP, es decir, un conjunto de los protocolos más utilizados en aplicaciones de Internet. Luego, el método `Install`, en la línea `stack.Install (nodes);`, recorre el contenedor de nodos y, para cada uno de ellos, crea y agrega el conjunto de protocolos antes listado a cada nodo.

Las siguientes líneas:

```
1  Ipv4AddressHelper address;
2  address.SetBase ("10.1.1.0", "255.255.255.0");
```

Crean un objeto (de la clase `Ipv4AddressHelper`) que permite la generación de direcciones IP. Luego, se definen la dirección y la máscara de la red. En este ejemplo: Dirección de red = 10.1.1.0 con máscara 255.255.255.0.

Luego, en la línea `Ipv4InterfaceContainer interfaces = address.Assign (devices);` se instancia un contenedor de interfaces IPv4. El método Assign recorre el contenedor de dispositivos y asigna direcciones IPv4, en la red configurada antes, a cada interface de cada nodo agregado. Estas direcciones son asignadas de forma secuencial, en este caso, se asignan las direcciones 10.1.1.1 y 10.1.1.2.

La línea `UdpEchoServerHelper echoServer (9);` sirve para indicar que el servidor echo estará escuchando solicitudes de echo a través del puerto 9.

La línea `ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));` crea un contenedor de aplicaciones y el método Install crea en el nodo 1 una aplicación de servidor "Echo UDP" (de la clase *UdpEchoServer*).

Las líneas:

```
1  serverApps.Start (Seconds (1.0));
2  serverApps.Stop (Seconds (10.0));
```

indican que la aplicación servidora se iniciará transcurrido 1 segundo del inicio de la simulación, y que terminará 10 segundos después del inicio de la simulación.

La línea `UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);` sirve para definir que el cliente "Echo UDP" se conectará al Servidor Echo utilizando como destino la dirección y el puerto de la segunda interface definida en el contenedor interfaces (es decir, la dirección del servidor, 10.1.1.2, y puerto remoto 9).

Las líneas:

```
1  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
2  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
3  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

establecen los atributos del cliente echo. En este caso: sólo se enviará un paquete al servidor echo, que habrá una espera de 1 segundo antes de enviar una nueva solicitud de echo, y que la cantidad de datos a transmitir en un paquete UDP (sin considerar encabezados) es de 1024.

En la línea `ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));`, el método `Install` crea una instancia de la clase UdpEchoClient y se agrega al nodo en la posición 0 en el contenedor de nodos.

Luego, las líneas,

```
1  clientApps.Start (Seconds (2.0));
2  clientApps.Stop (Seconds (10.0));
```

indican que la aplicación cliente se comenzará en el segundo 2 de la simulación (tiempo simulado), y que terminará en el segundo 10 después del inicio (es decir, 8 segundos simulados después de la inicialización de la aplicación).

La línea `Simulator::Run ();` busca y ejecuta la lista de eventos programados. En este caso, los eventos de inicio y término de ambas aplicación, servidor y cliente.

Finalmente, el método `Simulator::Destroy ();` libera la memoria asignada por el simulador. La simulación acaba con un "`return 0;`".

Para ejecutar esta simulación debemos ubicarnos en la carpeta `examples` que está en `ns-3-dev`. Una vez dentro, ejecutar:

```
1  ./waf --run tutorial/first
```

La salida de este programa es la siguiente:

```
1  Waf: Entering directory '/home/crduran/repos/ns-3-allinone/ns-3-dev/build'
2  Waf: Leaving directory '/home/crduran/repos/ns-3-allinone/ns-3-dev/build'
3  'build' finished successfully (0.726s)
4  At time 2s client sent 1024 bytes to 10.1.1.2 port 9
5  At time 2.00369s server received 1024 bytes from 10.1.1.1 port 49153
```

```
6 At time 2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
7 At time 2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

Las primeras líneas corresponden a la construcción de la simulación, lo que culmina (si todo está bien) con la frase “'build' finished successfully ”, indicando que el proceso fue terminado satisfactoriamente, seguido del tiempo invertido en este proceso (en este caso 0.726s). Estas líneas aparecerán comunmente cada vez que lancemos una simulación con waf. En el resto de este documento estas serán omitidas.

Las siguientes líneas muestran 4 eventos reportados por la simulación. La primera indica que a los 2 segundos de comenzada la simulación (como indicado en el código fuente en el llamado `clientApps.Start`) el cliente envía 1024 bytes al servidor (con dirección 10.1.1.2, puesto 9). Luego, en el tiempo 2.00369s, el servidor recibe estos 1024 bytes. En el tiempo 2.00369s el servidor termina de enviar la respuesta de echo respectiva, la que es recibida en el tiempo 2.00737s por el cliente.

3. Simulación de un algoritmo de ruteo

A continuación, desarrollaremos la temática principal de este documento, mostrando cómo simular un algoritmo de ruteo. Varios algoritmos de ruteo ya están implementados en ns-3. Estos protocolos de enrutamiento son subclases de la clase `ns3::Ipv4RoutingProtocol`. Para ver detalles de esto, se puede acceder a la documentación en `www.nsnam.org`.

Para comenzar, tomaremos como ejemplo el código del archivo `nix-simple.cc` ubicado en la carpeta `src/nix-vector-routin` dentro de `ns-3-dev`. El código escrito en el archivo es el siguiente:

```
1 /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2 /*
3  * This program is free software; you can redistribute it and/or modify
4  * it under the terms of the GNU General Public License version 2 as
5  * published by the Free Software Foundation;
6  *
7  * This program is distributed in the hope that it will be useful,
8  * but WITHOUT ANY WARRANTY; without even the implied warranty of
9  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
10 * GNU General Public License for more details.
11 *
12 * You should have received a copy of the GNU General Public License
13 * along with this program; if not, write to the Free Software
14 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
15 */
16
17 #include "ns3/core-module.h"
18 #include "ns3/network-module.h"
19 #include "ns3/internet-module.h"
20 #include "ns3/point-to-point-module.h"
21 #include "ns3/applications-module.h"
22 #include "ns3/ipv4-static-routing-helper.h"
23 #include "ns3/ipv4-list-routing-helper.h"
24 #include "ns3/ipv4-nix-vector-helper.h"
25
26 /*
27  *  Simple point to point links:
28  *
29  *  n0 -- n1 -- n2 -- n3
30  *
```

```
31  *   n0 has UdpEchoClient
32  *   n3 has UdpEchoServer
33  *
34  *   n0 IP: 10.1.1.1
35  *   n1 IP: 10.1.1.2, 10.1.2.1
36  *   n2 IP: 10.1.2.2, 10.1.3.1
37  *   n3 IP: 10.1.3.2
38  *
39  */
40
41 using namespace ns3;
42
43 NS_LOG_COMPONENT_DEFINE ("NixSimpleExample");
44
45 int
46 main (int argc, char *argv[])
47 {
48   LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
49   LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
50
51   NodeContainer nodes12;
52   nodes12.Create (2);
53
54   NodeContainer nodes23;
55   nodes23.Add (nodes12.Get (1));
56   nodes23.Create (1);
57
58   NodeContainer nodes34;
59   nodes34.Add (nodes23.Get (1));
60   nodes34.Create (1);
61
62   PointToPointHelper pointToPoint;
63   pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
64   pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
65
66   NodeContainer allNodes = NodeContainer (nodes12, nodes23.Get (1), nodes34.Get (1)
       );
67
68   // NixHelper to install nix-vector routing
69   // on all nodes
70   Ipv4NixVectorHelper nixRouting;
71   Ipv4StaticRoutingHelper staticRouting;
72
73   Ipv4ListRoutingHelper list;
74   list.Add (staticRouting, 0);
75   list.Add (nixRouting, 10);
76
77   InternetStackHelper stack;
78   stack.SetRoutingHelper (list); // has effect on the next Install ()
79   stack.Install (allNodes);
80
81   NetDeviceContainer devices12;
```

```
82    NetDeviceContainer devices23;
83    NetDeviceContainer devices34;
84    devices12 = pointToPoint.Install (nodes12);
85    devices23 = pointToPoint.Install (nodes23);
86    devices34 = pointToPoint.Install (nodes34);
87
88    Ipv4AddressHelper address1;
89    address1.SetBase ("10.1.1.0", "255.255.255.0");
90    Ipv4AddressHelper address2;
91    address2.SetBase ("10.1.2.0", "255.255.255.0");
92    Ipv4AddressHelper address3;
93    address3.SetBase ("10.1.3.0", "255.255.255.0");
94
95    address1.Assign (devices12);
96    address2.Assign (devices23);
97    Ipv4InterfaceContainer interfaces = address3.Assign (devices34);
98
99    UdpEchoServerHelper echoServer (9);
100
101   ApplicationContainer serverApps = echoServer.Install (nodes34.Get (1));
102   serverApps.Start (Seconds (1.0));
103   serverApps.Stop (Seconds (10.0));
104
105   UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
106   echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
107   echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.)));
108   echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
109
110   ApplicationContainer clientApps = echoClient.Install (nodes12.Get (0));
111   clientApps.Start (Seconds (2.0));
112   clientApps.Stop (Seconds (10.0));
113
114   // Trace routing tables
115   Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("nix-simple
          .routes", std::ios::out);
116   nixRouting.PrintRoutingTableAllAt (Seconds (8), routingStream);
117
118   Simulator::Run ();
119   Simulator::Destroy ();
120   return 0;
121 }
```

Este código, muestra un ejemplo similar al anterior (solicitud y envío de echo entre dos unidades), pero definiendo al emisor y receptor de los mensajes como ubicados en redes distintas, definiendo una topología con varios saltos, como muestra la Figura 2.

El código fuente es similar al anterior, salvo por algunos detalles que a continuación se explican.

En el ejemplo de la sección anterior, se creó un sólo contenedor para una comunicación punto a punto entre dos nodos. En este nuevo caso, debemos crear varios contenedores para definir enlaces punto a punto entre ellos, sabiendo que, en algunos casos, un nodo tendrá dos enlaces. En el ejemplo, la declaración de esta topología, se encuentra representada por las siguientes líneas:

1. `NodeContainer nodes12;` Declara un contenedor para asociar los nodos 1 y 2. Esto, porque, a posterior,

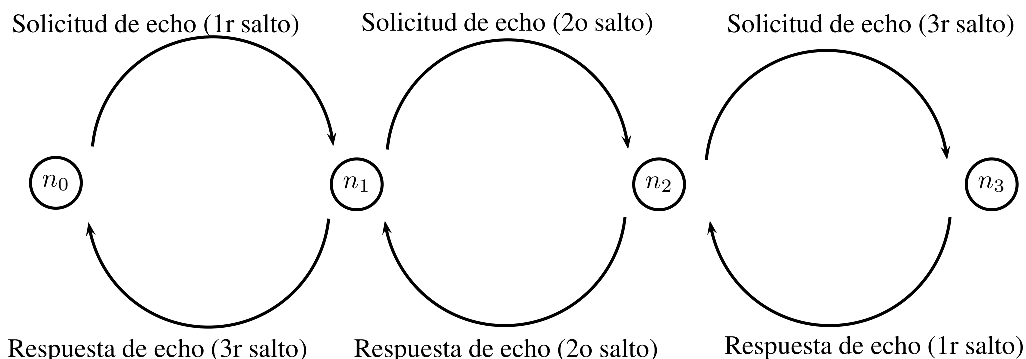

Figura 2: Ilustración de ejemplo en `nix-simple.cc`.

se utilizará una clase para definir una conexión punto a punto entre estos 2 dispositivos (el método install asociado conecta sólo 2 nodos).

2. `nodes12.Create (2);` Crea los 2 nodos en el contenedor `nodes12` (nodos 1 y 2).
3. `NodeContainer nodes23;` Declara un contenedor para asociar los nodos 2 y 3.
4. `nodes23.Add (nodes12.Get (1));` Agrega el segundo nodo (con índice 1) del contenedor `nodes12` al contenedor `nodes23`.
5. `nodes23.Create (1);` Crea un nuevo nodo en el contenedor `nodes23` (el nodo 3).
6. `NodeContainer nodes34;` Declara un contenedor para asociar los nodos 3 y 4.
7. `nodes34.Add (nodes23.Get (1));` Agrega el segundo nodo (con índice 1) del contenedor `nodes23` al contenedor `nodes34`.
8. `nodes34.Create (1);` Crea un nuevo nodo en el contenedor `nodes34` (el nodo 4).

Luego, las líneas

```
1  Ipv4NixVectorHelper nixRouting;
2  Ipv4StaticRoutingHelper staticRouting;
3
4  Ipv4ListRoutingHelper list;
5  list.Add (staticRouting, 0);
6  list.Add (nixRouting, 10);
```

definen y agregan los protocolos de ruteo que cada utilizará. Estos quedan agregado en la lista `list` de la clase `Ipv4ListRoutingHelper`. En la práctica, el componente ejecutará el protocolo de enrutamiento con la mayor prioridad, es decir, el que tenga el segundo parámetro con un valor más alto (en este caso, `nixRouting`).

Las líneas

```
1  InternetStackHelper stack;
2  stack.SetRoutingHelper (list); // has effect on the next Install ()
3  stack.Install (allNodes);
```

ayudan a definir el stack de funcionalidades de protocolos TCP/UDP/IP a utilizar por los nodos. Se configura la utilización del protocolo NIX y se instala en todos los nodos.

Luego (de forma análoga al ejemplo de la sección anterior), por cada par de nodos declarado en los distintos contenedores, `nodes12`, `nodes23`, `nodes34`, se declara un contenedor de dispositivos de red. Luego, el método `Install` del objeto `pointToPoint` de la clase `PointToPointHelper` crea los dispositivos de red punto a punto (instancias de la clase `PointToPointNetDevice`) para cada nodo, les asigna una dirección MAC y adhiere el dispositivo al nodo correspondiente. Luego, estos dispositivos son adheridos a un canal punto a punto. Esto se realiza con las líneas:

```
1   NetDeviceContainer devices12;
2   NetDeviceContainer devices23;
3   NetDeviceContainer devices34;
4   devices12 = pointToPoint.Install (nodes12);
5   devices23 = pointToPoint.Install (nodes23);
6   devices34 = pointToPoint.Install (nodes34);
```

Luego, las siguientes líneas:

```
1   Ipv4AddressHelper address1;
2   address1.SetBase ("10.1.1.0", "255.255.255.0");
3   Ipv4AddressHelper address2;
4   address2.SetBase ("10.1.2.0", "255.255.255.0");
5   Ipv4AddressHelper address3;
6   address3.SetBase ("10.1.3.0", "255.255.255.0");
7
8   address1.Assign (devices12);
9   address2.Assign (devices23);
10  Ipv4InterfaceContainer interfaces = address3.Assign (devices34);
```

Configuran y asignan las direcciones IP y las máscaras de cada subred. La última asignación es retornada sobre un objeto `interfaces` de la clase `Ipv4InterfaceContainer`, con el objeto de facilitar la recuperación de la dirección del nodo cliente, como mostrado en el ejemplo de la sección anterior.

El resto del código es equivalente al del ejemplo anterior, salvo por las líneas:

```
1   Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("nix-simple
        .routes", std::ios::out);
2   nixRouting.PrintRoutingTableAllAt (Seconds (8), routingStream);
```

En resumen, con estas, se crea un archivo llamado `nix-simple.routes`. Cuando se ejecuta la simulación, se crea este archivo en el directorio `ns-3-dev`, y en él se registran las tablas de ruteo para cada nodo. En este caso, luego de ejecutar la simulación (`./waf --run src/nix-vector-routing/examples/nix-simple` desde la carpeta `ns-3-dev`), el archivo `nix-simple.routes` muestra lo siguiente:

```
1   Node: 0 Time: 8s Ipv4ListRouting table
2     Priority: 10 Protocol: ns3::Ipv4NixVectorRouting
3   NixCache:
4   Destination     NixVector
5   10.1.3.2        011
6   Ipv4RouteCache:
7   Destination     Gateway         Source          OutputDevice
8   10.1.3.2        10.1.1.2        10.1.1.1        1
9     Priority: 0 Protocol: ns3::Ipv4StaticRouting
10  Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
11  127.0.0.0       0.0.0.0         255.0.0.0       U     0      -      -   0
12  10.1.1.0        0.0.0.0         255.255.255.0   U     0      -      -   1
```

```
13
14  Node: 1 Time: 8s Ipv4ListRouting table
15    Priority: 10 Protocol: ns3::Ipv4NixVectorRouting
16  NixCache:
17  Ipv4RouteCache:
18  Destination     Gateway          Source             OutputDevice
19  10.1.1.1        10.1.1.1         10.1.1.2           1
20  10.1.3.2        10.1.2.2         10.1.2.1           2
21    Priority: 0 Protocol: ns3::Ipv4StaticRouting
22  Destination     Gateway          Genmask          Flags Metric Ref    Use Iface
23  127.0.0.0       0.0.0.0          255.0.0.0        U     0      -      -   0
24  10.1.1.0        0.0.0.0          255.255.255.0    U     0      -      -   1
25  10.1.2.0        0.0.0.0          255.255.255.0    U     0      -      -   2
26
27  Node: 2 Time: 8s Ipv4ListRouting table
28    Priority: 10 Protocol: ns3::Ipv4NixVectorRouting
29  NixCache:
30  Ipv4RouteCache:
31  Destination     Gateway          Source             OutputDevice
32  10.1.1.1        10.1.2.1         10.1.2.2           1
33  10.1.3.2        10.1.3.2         10.1.3.1           2
34    Priority: 0 Protocol: ns3::Ipv4StaticRouting
35  Destination     Gateway          Genmask          Flags Metric Ref    Use Iface
36  127.0.0.0       0.0.0.0          255.0.0.0        U     0      -      -   0
37  10.1.2.0        0.0.0.0          255.255.255.0    U     0      -      -   1
38  10.1.3.0        0.0.0.0          255.255.255.0    U     0      -      -   2
39
40  Node: 3 Time: 8s Ipv4ListRouting table
41    Priority: 10 Protocol: ns3::Ipv4NixVectorRouting
42  NixCache:
43  Destination     NixVector
44  10.1.1.1        000
45  Ipv4RouteCache:
46  Destination     Gateway          Source             OutputDevice
47  10.1.1.1        10.1.3.1         10.1.3.2           1
48    Priority: 0 Protocol: ns3::Ipv4StaticRouting
49  Destination     Gateway          Genmask          Flags Metric Ref    Use Iface
50  127.0.0.0       0.0.0.0          255.0.0.0        U     0      -      -   0
51  10.1.3.0        0.0.0.0          255.255.255.0    U     0      -      -   1
```

4. El caso inalámbrico

En el caso de las redes inalámbricas ad-hoc, existen algunas diferencias con respecto al caso alámbrico. Particularmente, en la declaración de la topología y aspectos relacionados con las características del canal. Revisemos el siguiente ejemplo, encontrado en el archivo `wifi-simple-adhoc-grid.cc`:

Listado 2: Contenido del archivo `wifi.simple-adhoc-grid.cc`.

```
1  /* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
2  /*
3   * Copyright (c) 2009 University of Washington
4   *
```

```
5   * This program is free software; you can redistribute it and/or modify
6   * it under the terms of the GNU General Public License version 2 as
7   * published by the Free Software Foundation;
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17  *
18  */
19
20 //
21 // This program configures a grid (default 5x5) of nodes on an
22 // 802.11b physical layer, with
23 // 802.11b NICs in adhoc mode, and by default, sends one packet of 1000
24 // (application) bytes to node 1.
25 //
26 // The default layout is like this, on a 2-D grid.
27 //
28 // n20  n21  n22  n23  n24
29 // n15  n16  n17  n18  n19
30 // n10  n11  n12  n13  n14
31 // n5   n6   n7   n8   n9
32 // n0   n1   n2   n3   n4
33 //
34 // the layout is affected by the parameters given to GridPositionAllocator;
35 // by default, GridWidth is 5 and numNodes is 25..
36 //
37 // There are a number of command-line options available to control
38 // the default behavior. The list of available command-line options
39 // can be listed with the following command:
40 // ./waf --run "wifi-simple-adhoc-grid --help"
41 //
42 // Note that all ns-3 attributes (not just the ones exposed in the below
43 // script) can be changed at command line; see the ns-3 documentation.
44 //
45 // For instance, for this configuration, the physical layer will
46 // stop successfully receiving packets when distance increases beyond
47 // the default of 500m.
48 // To see this effect, try running:
49 //
50 // ./waf --run "wifi-simple-adhoc --distance=500"
51 // ./waf --run "wifi-simple-adhoc --distance=1000"
52 // ./waf --run "wifi-simple-adhoc --distance=1500"
53 //
54 // The source node and sink node can be changed like this:
55 //
56 // ./waf --run "wifi-simple-adhoc --sourceNode=20 --sinkNode=10"
```

```
57  //
58  // This script can also be helpful to put the Wifi layer into verbose
59  // logging mode; this command will turn on all wifi logging:
60  //
61  // ./waf --run "wifi-simple-adhoc-grid --verbose=1"
62  //
63  // By default, trace file writing is off-- to enable it, try:
64  // ./waf --run "wifi-simple-adhoc-grid --tracing=1"
65  //
66  // When you are done tracing, you will notice many pcap trace files
67  // in your directory.  If you have tcpdump installed, you can try this:
68  //
69  // tcpdump -r wifi-simple-adhoc-grid-0-0.pcap -nn -tt
70  //
71
72  #include "ns3/core-module.h"
73  #include "ns3/network-module.h"
74  #include "ns3/mobility-module.h"
75  #include "ns3/config-store-module.h"
76  #include "ns3/wifi-module.h"
77  #include "ns3/internet-module.h"
78  #include "ns3/olsr-helper.h"
79  #include "ns3/ipv4-static-routing-helper.h"
80  #include "ns3/ipv4-list-routing-helper.h"
81
82  #include <iostream>
83  #include <fstream>
84  #include <vector>
85  #include <string>
86
87  NS_LOG_COMPONENT_DEFINE ("WifiSimpleAdhocGrid");
88
89  using namespace ns3;
90
91  void ReceivePacket (Ptr<Socket> socket)
92  {
93    NS_LOG_UNCOND ("Received one packet!");
94  }
95
96  static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
97                               uint32_t pktCount, Time pktInterval )
98  {
99    if (pktCount > 0)
100     {
101       socket->Send (Create<Packet> (pktSize));
102       Simulator::Schedule (pktInterval, &GenerateTraffic,
103                            socket, pktSize,pktCount-1, pktInterval);
104     }
105   else
106     {
107       socket->Close ();
108     }
```

```
109 }
110
111
112 int main (int argc, char *argv[])
113 {
114   std::string phyMode ("DsssRate1Mbps");
115   double distance = 500;  // m
116   uint32_t packetSize = 1000; // bytes
117   uint32_t numPackets = 1;
118   uint32_t numNodes = 25;  // by default, 5x5
119   uint32_t sinkNode = 0;
120   uint32_t sourceNode = 24;
121   double interval = 1.0; // seconds
122   bool verbose = false;
123   bool tracing = false;
124
125   CommandLine cmd;
126
127   cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
128   cmd.AddValue ("distance", "distance (m)", distance);
129   cmd.AddValue ("packetSize", "size of application packet sent", packetSize);
130   cmd.AddValue ("numPackets", "number of packets generated", numPackets);
131   cmd.AddValue ("interval", "interval (seconds) between packets", interval);
132   cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbose);
133   cmd.AddValue ("tracing", "turn on ascii and pcap tracing", tracing);
134   cmd.AddValue ("numNodes", "number of nodes", numNodes);
135   cmd.AddValue ("sinkNode", "Receiver node number", sinkNode);
136   cmd.AddValue ("sourceNode", "Sender node number", sourceNode);
137
138   cmd.Parse (argc, argv);
139   // Convert to time object
140   Time interPacketInterval = Seconds (interval);
141
142   // disable fragmentation for frames below 2200 bytes
143   Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
        StringValue ("2200"));
144   // turn off RTS/CTS for frames below 2200 bytes
145   Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue
        ("2200"));
146   // Fix non-unicast data rate to be the same as that of unicast
147   Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
148                       StringValue (phyMode));
149
150   NodeContainer c;
151   c.Create (numNodes);
152
153   // The below set of helpers will help us to put together the wifi NICs we want
154   WifiHelper wifi;
155   if (verbose)
156     {
157       wifi.EnableLogComponents ();  // Turn on all Wifi logging
158     }
```

```
159
160   YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
161   // set it to zero; otherwise, gain will be added
162   wifiPhy.Set ("RxGain", DoubleValue (-10) );
163   // ns-3 supports RadioTap and Prism tracing extensions for 802.11b
164   wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
165
166   YansWifiChannelHelper wifiChannel;
167   wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
168   wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
169   wifiPhy.SetChannel (wifiChannel.Create ());
170
171   // Add a non-QoS upper mac, and disable rate control
172   NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
173   wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
174   wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
175                                 "DataMode",StringValue (phyMode),
176                                 "ControlMode",StringValue (phyMode));
177   // Set it to adhoc mode
178   wifiMac.SetType ("ns3::AdhocWifiMac");
179   NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);
180
181   MobilityHelper mobility;
182   mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
183                                  "MinX", DoubleValue (0.0),
184                                  "MinY", DoubleValue (0.0),
185                                  "DeltaX", DoubleValue (distance),
186                                  "DeltaY", DoubleValue (distance),
187                                  "GridWidth", UintegerValue (5),
188                                  "LayoutType", StringValue ("RowFirst"));
189   mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
190   mobility.Install (c);
191
192   // Enable OLSR
193   OlsrHelper olsr;
194   Ipv4StaticRoutingHelper staticRouting;
195
196   Ipv4ListRoutingHelper list;
197   list.Add (staticRouting, 0);
198   list.Add (olsr, 10);
199
200   InternetStackHelper internet;
201   internet.SetRoutingHelper (list); // has effect on the next Install ()
202   internet.Install (c);
203
204   Ipv4AddressHelper ipv4;
205   NS_LOG_INFO ("Assign IP Addresses.");
206   ipv4.SetBase ("10.1.1.0", "255.255.255.0");
207   Ipv4InterfaceContainer i = ipv4.Assign (devices);
208
209   TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
210   Ptr<Socket> recvSink = Socket::CreateSocket (c.Get (sinkNode), tid);
```

```
211   InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
212   recvSink->Bind (local);
213   recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));
214
215   Ptr<Socket> source = Socket::CreateSocket (c.Get (sourceNode), tid);
216   InetSocketAddress remote = InetSocketAddress (i.GetAddress (sinkNode, 0), 80);
217   source->Connect (remote);
218
219   if (tracing == true)
220     {
221       AsciiTraceHelper ascii;
222       wifiPhy.EnableAsciiAll (ascii.CreateFileStream ("wifi-simple-adhoc-grid.tr"))
          ;
223       wifiPhy.EnablePcap ("wifi-simple-adhoc-grid", devices);
224       // Trace routing tables
225       Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("wifi-
          simple-adhoc-grid.routes", std::ios::out);
226       olsr.PrintRoutingTableAllEvery (Seconds (2), routingStream);
227
228       // To do-- enable an IP-level trace that shows forwarding events only
229     }
230
231   // Give OLSR time to converge-- 30 seconds perhaps
232   Simulator::Schedule (Seconds (30.0), &GenerateTraffic,
233                        source, packetSize, numPackets, interPacketInterval);
234
235   // Output what we are doing
236   NS_LOG_UNCOND ("Testing from node " << sourceNode << " to " << sinkNode << " with
        grid distance " << distance);
237
238   Simulator::Stop (Seconds (32.0));
239   Simulator::Run ();
240   Simulator::Destroy ();
241
242   return 0;
243 }
```

Sin considerar las nuevas librerías, una diferencia con los ejemplos anteriores es la declaración de dos funciones llamadas `ReceivePacket` y `GenerateTraffic`. Para faciliar la comprensión de este texto, explicaremos su implementación más adelante.

El código principal declara un objeto `cmd` de la clase `CommandLine`. Esta clase permite declarar, modificar o acceder variables o argumentos de programa a través de líneas de comandos.

Las líneas

```
1   cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
2   cmd.AddValue ("distance", "distance (m)", distance);
3   cmd.AddValue ("packetSize", "size of application packet sent", packetSize);
4   cmd.AddValue ("numPackets", "number of packets generated", numPackets);
5   cmd.AddValue ("interval", "interval (seconds) between packets", interval);
6   cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbose);
7   cmd.AddValue ("tracing", "turn on ascii and pcap tracing", tracing);
8   cmd.AddValue ("numNodes", "number of nodes", numNodes);
9   cmd.AddValue ("sinkNode", "Receiver node number", sinkNode);
```

```
10    cmd.AddValue ("sourceNode", "Sender node number", sourceNode);
```

Definen 10 argumentos de programa con el método `AddValue`, donde el primer parámetro es el nombre del argumento, el segundo es una descripción (un texto de ayuda utilizable por la opción `--PrintHelp`), y el tercero es la referencia a la variable donde se almacenará el valor del argumento en cuestión.

Con la línea `cmd.Parse (argc, argv);`, el objeto analiza los argumentos de programa.

La línea `Time interPacketInterval = Seconds (interval);` define un objeto de clase `Time` para definir el intervalo de tiempo en que se transmitirán los paquetes de datos. Obviamente, la función `Seconds` retorna un objeto de la clase `Time` con el valor asignado a la variable interval (en este caso 1,0 segundo).

Las líneas

```
1   Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
        StringValue ("2200"));
2
3   Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue
        ("2200"));
4
5   Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
6                       StringValue (phyMode));
```

configuran características particulares de la simulación a realizar, concernientes a la fregmentación de paquetes, a la utilización del modo RTS/CTS de 802.11, y a la tasa de transmisión en modo “no unicast”.

Luego de creados los nodos (en este caso, 25), se definen los objetos Helper para facilitar la implementación de la simulación.

La clase `WifiHelper` permite (en otras funciones) crear y asignar dispositivos de red a los nodos considerados en la simulación, y configurar atributos. El método `EnableLogComponents` habilita los todos los componentes de registros log de los dispositivos de red inalámbricos.

Las siguientes líneas:

```
1   YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
2   // set it to zero; otherwise, gain will be added
3   wifiPhy.Set ("RxGain", DoubleValue (-10) );
4   // ns-3 supports RadioTap and Prism tracing extensions for 802.11b
5   wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
6
7   YansWifiChannelHelper wifiChannel;
8   wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
9   wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
10  wifiPhy.SetChannel (wifiChannel.Create ());
11
12  // Add a non-QoS upper mac, and disable rate control
13  NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
14  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
15  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
16                                "DataMode",StringValue (phyMode),
17                                "ControlMode",StringValue (phyMode));
18  // Set it to adhoc mode
19  wifiMac.SetType ("ns3::AdhocWifiMac");
```

definen y parametrizan objetos Helper referidos a la implementación de los estándares de capa física y MAC a utilizar en la simulación. A nivel físico se configura un modelo conocido como YANS (Yet Another Network Simulator) descrito en [2], la ganancia de recepción y características del canal Wi-Fi como velocidad de propagación y modelo de pérdida por propagación. A nivel de subcapa MAC, se escoge el estándar de comunicación, algoritmo de control de tasa de transmisión, y modo Adhoc.

Finalmente, la línea `NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);` define el contenedor de dispositivos de red `devices` y crea, para cada nodo en `c`, los dispositivos inalámbricos configurándose dirección MAC, y asignándosele el conjunto de características y estándares declarados en las capas Física y MAC.

En las líneas

```
1   MobilityHelper mobility;
2   mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
3                                  "MinX", DoubleValue (0.0),
4                                  "MinY", DoubleValue (0.0),
5                                  "DeltaX", DoubleValue (distance),
6                                  "DeltaY", DoubleValue (distance),
7                                  "GridWidth", UintegerValue (5),
8                                  "LayoutType", StringValue ("RowFirst"));
9   mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
10  mobility.Install (c);
```

se establece la posición de los diferentes nodos, utilizando un objeto `MobilityHelper`, quién a su vez utiliza un modelo `GridPositionAllocator` para posicionar los objetos en una grilla rectangular bidimensional. La línea llamando al método `SetPositionAllocator` explicita la utilización de este modelo `GridPositionAllocator` y sus parámetros indicando, en este ejemplo, un área que comienza en la coordenada cartesiana (0,0), con distancias Δ_x y $Delta_y$ de, este caso 500 metros (definidos en la variable `distance` al comienzo de la función main, y un total de 5 nodos por línea (ancho de la grilla), como muestra la Figura 3.

El parámetro LayoutType con valor "RowFirst"indica que las posiciones son asignadas línea por línea.

El método `SetMobilityModel` con parámetro `"ns3::ConstantPositionMobilityModel")` refiere a un modelo con posiciones constantes (es decir, los nodos estarán estáticos).

Las siguientes líneas

```
193  OlsrHelper olsr;
194  Ipv4StaticRoutingHelper staticRouting;
195
196  Ipv4ListRoutingHelper list;
197  list.Add (staticRouting, 0);
198  list.Add (olsr, 10);
199
200  InternetStackHelper internet;
201  internet.SetRoutingHelper (list); // has effect on the next Install ()
202  internet.Install (c);
203
204  Ipv4AddressHelper ipv4;
205  NS_LOG_INFO ("Assign IP Addresses.");
206  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
207  Ipv4InterfaceContainer i = ipv4.Assign (devices);
```

son equivalentes al ejemplo de la sección 3, salvo que en este caso se utilza el protocolo OLSR (como Optimized Link State Routing) definido en [1]. Este protocolo fue diseñado para trabajar en redes ad hoc móviles.

A diferencia de los ejemplos anteriores, donde el tráfico es generado por aplicaciones ya implementadas[2], en este ejemplo se utiliza una comunicación a través de interfaces Sockets.

La línea `TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");` define el identificador único para la interface a utilizar. Las siguientes líneas:

```
210  Ptr<Socket> recvSink = Socket::CreateSocket (c.Get (sinkNode), tid);
211  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
```

[2] Existen varias aplicaciones programadas en ns3 como `UdpEchoClient`, `UdpEchoServer`, `V4Ping`, etc.

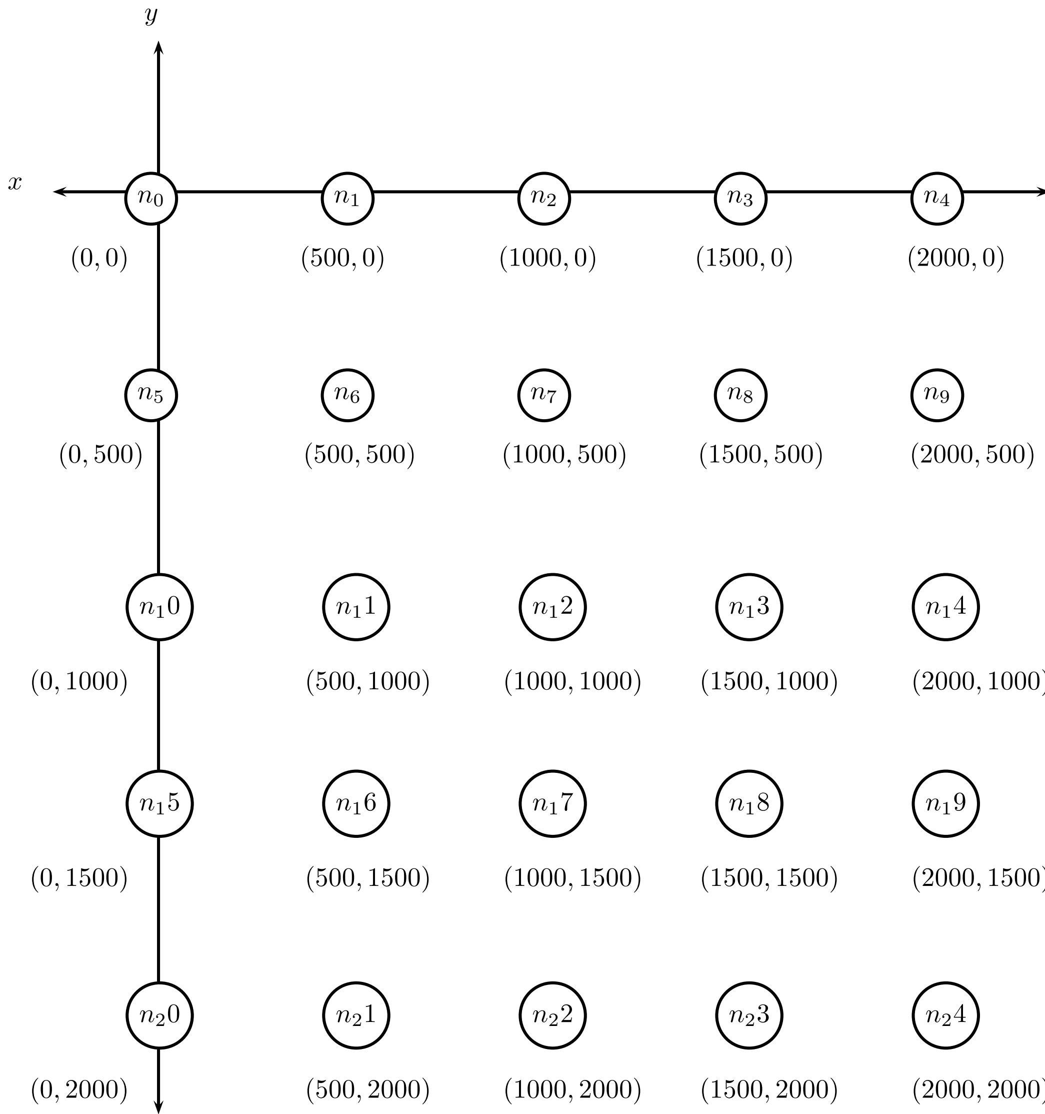

Figura 3: Ilustración (referencial) de la topología definida en `wifi.simple-adhoc-grid.cc`.

```
212    recvSink->Bind (local);
213    recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));
```

permiten crear un nuevo Socket, en este caso el Socket de la aplicación receptora (el Server), y su configuración. La línea `InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);` crea un objeto para identificar la dirección y número de puerto del servidor. El único valor relevante para esta aplicación (pues, para este caso de ejemplo, la aplicación servidora no necesita conocer su propia dirección IP) es el número de puerto a través del cuál escuchará solicitudes de conexión (en este caso, el puerto 80). El método `Bind` asigna al socket los parámetros definidos, y, finalmente, el método `SetRecvCallback` asigna el nombre de la función que será llamada cuando un nodo reciba un paquete (es decir, la función activada ante un evento de recepción). En este caso, se llama a la función `ReceivePacket`, implementada al comienzo del código, que recibe como único parámetro un puntero al objeto `Socket` del nodo receptor. La implementación de esta función consiste en una única línea con una llamada a la macro `NS_LOG_UNCOND` que imprime un mensaje en pantalla independiente de los niveles de registros log activados (no forma parte de ninguno de los niveles definidos, por lo tanto, no necesita activarse o desactivarse).

Luego, las líneas

```
215  Ptr<Socket> source = Socket::CreateSocket (c.Get (sourceNode), tid);
216  InetSocketAddress remote = InetSocketAddress (i.GetAddress (sinkNode, 0), 80);
```

describen la creación y configuración del nodo cliente, que se conectará al socket del nodo con dirección IP equivalente a la dirección IP del nodo servidor y puerto 80. La línea `source->Connect (remote);` ordena la conexión del nodo cliente (source) al servidor.

Las líneas

```
221      AsciiTraceHelper ascii;
222      wifiPhy.EnableAsciiAll (ascii.CreateFileStream ("wifi-simple-adhoc-grid.tr"))
             ;
223      wifiPhy.EnablePcap ("wifi-simple-adhoc-grid", devices);
224      // Trace routing tables
225      Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("wifi-
             simple-adhoc-grid.routes", std::ios::out);
226      olsr.PrintRoutingTableAllEvery (Seconds (2), routingStream);
227
228      // To do-- enable an IP-level trace that shows forwarding events only
```

están condicionadas al valor de la variable booleana `tracing` (valor definido por defecto = False). Estas están destinadas a la creación de archivos .tr (de tracing), .pcap (compatibles con Wireshark y Tcpdump) y archivos con las tablas de enrutamiento.

La línea `Simulator::Schedule (Seconds (30.0), &GenerateTraffic, source, packetSize, numPackets, interPacketInterval);` programa un evento que comienza en el segundo 30 que llama a la función `GenerateTraffic` definido luego del método `ReceivePacket` al comienzo del código. Los cuatro últimos parámetros son los parámetros de entrada de la función `GenerateTraffic`.

La interpretación de esta función `GenerateTraffic` es trivial; mientras el contador `pktCount`, ingresado por parámetro, sea mayor que cero, se enviará un paquete de datos de tamaño `pktSize` y se programará el siguiente evento de transmisión `pktInterval` segundos después, con `pktCount = pktCount - 1`. Cuando `pktCount` sea igual a cero, se cerrará la conexión socket.

Las últimas líneas son equivalentes a las de los ejemplos anteriores.

Este y otros ejemplos pueden ser encontrados en el repertorio `/home/crduran/repos/ns-3-allinone/ns-3-dev/examples/wi` Ejemplos más particulares pueden ser, además, encontrados en el repertorio `/home/crduran/repos/ns-3-allinone/ns-3-dev/s` Por ejemplo, el código de una simulación de ejemplo que emplea el protocolo de enrutamiento AODV (blablabla) que corresponde a uno de los protocolos más conocidos orientados a redes ad hoc.

5. La simulación de un algoritmo de ruteo desde cero

Para ilustrar cómo implementar un algoritmo de ruteo desde cero, consideraremos un algoritmo de ruteo sencillo basado en vectores de distancia con sólo un destino y un origen, y con una descubierta de red iniciada por el nodo destino (como podría ser implementado en una red de sensores inalámbricos tipo mesh).

Los algoritmos de ruteo disponibles en ns-3, como AODV y Nix Vector Routing, están almacenados en la carpeta `src`. Cada implementación tiene un directorio independiente en el que se encuentran algunos archivos y subdirectorios particulares. Los archivos y directorios principales que componen cada implementación son los siguientes:

- `doc`: Directorio con archivos de documentación.
- `examples`: Directorio que almacena ejemplo de simulación utilizando el módulo en cuestión.
- `helper`: Directorio que contiene las clases Helper que permiten instanciar objetos que implementan los protocolos (u otros recursos) a utilizar.

- `model`: Directorio que contiene los códigos fuentes (clases, etc.) de la implementación de los algoritmos.
- `test` (CHECKEAR)
- `wscript`: Archivo con declaraciones para apoyar la creación del módulo (CHECKEAR)

Llamaremos al algoritmo antes explicado algoritmo Discovery. Para comenzar se debe crear una carpeta para el proyecto en el directorio `/home/crduran/repos/ns-3-allinone/ns-3-dev/src`. Utilizaremos para este directorio el nombre `discovery`. Dentro de este directorio, se crearán las carpetas y archivos mencionados anteriormente. El conjunto de los archivos mencionados en este tutorial puede ser descargado desde: `http://crduran.ubb.cl/site/documents/tutor`
En todo los casos, la documentación necesaria para crear un nuevo módulo y agregarlo a ns3 se encuentra disponible en `https://www.nsnam.org/docs/release/3.22/manual/html/new-models.html`.

5.1. Implementación del algoritmo de ruteo

Considerando los ejemplos disponibles en ns-3, la implementación del algoritmo de ruteo se realizará en el directorio `model` (dentro de `discovery`). Esta carpeta contendrá 8 llamados:

- `discovery-routing-protocol.cc`
- `discovery-routing-protocol.h`
- `discovery-id-cache.cc`
- `discovery-id-cache.h`
- `discovery-packet.cc`
- `discovery-packet.h`
- `discovery-rtable.cc`
- `discovery-rtable.h`

Las siguientes subsecciones contienen la explicación de cada uno.

5.1.1. `discovery-routing-protocol.cc` y `.h`

El archivo discovery-routing-protocol.cc contiene el código con la implementación general del algoritmo de ruteo. El código de este archivo, que se explicará más adelante, es el siguiente:

Listado 3: Contenido del archivo `discovery-routing-protocol.cc`

```
1  #include "discovery-routing-protocol.h"
2  #include "ns3/log.h"
3  #include "ns3/udp-socket-factory.h"
4  #include "ns3/adhoc-wifi-mac.h"
5
6  #include <map>
7  #include <utility>
8
9  NS_LOG_COMPONENT_DEFINE ("DiscoveryRoutingProtocol");
10
11 namespace ns3 {
12 namespace discovery
13 {
14
```

```
15 NS_OBJECT_ENSURE_REGISTERED (RoutingProtocol);
16
17 // Estructura que almacena la interfaz y el socket de cada nodo
18 struct nodes_sockAddr {
19   Ptr<Socket> sockNode;
20   Ipv4InterfaceAddress ifNode;
21   };
22
23 // Mapa que contiene la interfaz y el socket de cada nodo
24 uint16_t key = 0;
25 std::map<uint16_t, nodes_sockAddr> m_sockIfNodes;
26
27 uint32_t packetId = 0;
28 double timeInitBroad = 1.5; // Tiempo de inicio de redifusión del primer paquete
     discovery.
29
30 RoutingProtocol::RoutingProtocol ():
31   d_packsDiscvIdCache (MilliSeconds (5600)),
32   d_routingTable (Time (15)),
33   d_timeStartDiscv (Seconds (1)), // Indica el tiempo en que el destino difunde el
       primer mensaje discovery.
34   d_broadInterval (0.005) // Intervalo de tiempo para la redifusión.
35 {
36
37 }
38
39 RoutingProtocol::~RoutingProtocol()
40 {
41 }
42
43 TypeId
44 RoutingProtocol::GetTypeId (void)
45 {
46   static TypeId tid = TypeId ("ns3::discovery::RoutingProtocol")
47     .SetParent<Ipv4RoutingProtocol> ()
48     .AddConstructor<RoutingProtocol> ()
49     .AddAttribute ("d_timeStartDiscv", "Tiempo de inicio para que el destino envíe
         el primer mensaje discovery.",
50                    TimeValue (Seconds (1)),
51                    MakeTimeAccessor (&RoutingProtocol::d_timeStartDiscv),
52                    MakeTimeChecker ())
53   ;
54   return tid;
55 }
56
57 // Método invocado cada vez que un nodo busca una ruta para un paquete recibido.
     Retorna el siguiente salto para este paquete.
58 Ptr<Ipv4Route>
59 RoutingProtocol::RouteOutput (Ptr<Packet> p, const Ipv4Header &header,
60                               Ptr<NetDevice> oif, Socket::SocketErrno &sockerr)
61 {
62   sockerr = Socket::ERROR_NOTERROR;
```

```
63     Ptr<Ipv4Route> route;
64     Ipv4Address dst = header.GetDestination ();
65
66     RoutingTableEntry rt;
67
68     // Buscar una ruta en la tabla de enrutamiento.
69     if (d_routingTable.LookupRoute (dst, rt))
70       {
71         route = rt.GetRoute ();
72         NS_ASSERT (route != 0);
73         NS_LOG_DEBUG ("Exist route to " << route->GetDestination () << " from
               interface " << route->GetSource ());
74
75         if (oif != 0 && route->GetOutputDevice () != oif)
76           {
77             NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
78             sockerr = Socket::ERROR_NOROUTETOHOST;
79             return Ptr<Ipv4Route> ();
80           }
81
82         return route;
83       }
84     else
85       {
86         return Ptr<Ipv4Route> ();
87       }
88   }
89
90   // Método llamado cada vez que un nodo recibe un paquete. Determina si un paquete
        entrante es liberado localmente o si es enviado a otros
91   // nodos mediante el método forwarding().
92   bool
93   RoutingProtocol::RouteInput (Ptr<const Packet> p, const Ipv4Header &header,
94                                Ptr<const NetDevice> idev, UnicastForwardCallback ucb,
95                                MulticastForwardCallback mcb, LocalDeliverCallback lcb
                                   , ErrorCallback ecb)
96   {
97
98     NS_LOG_FUNCTION (this << p->GetUid () << header.GetDestination () << idev->
           GetAddress ());
99     NS_ASSERT (d_ipv4 != 0);
100    NS_ASSERT (p != 0);
101    NS_ASSERT (d_ipv4->GetInterfaceForDevice (idev) >= 0);
102
103    int32_t iif = d_ipv4->GetInterfaceForDevice (idev);
104    Ipv4Address dst = header.GetDestination ();
105    Ipv4Address origin = header.GetSource ();
106
107    // Liberar localmente el paquete recibido
108    for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
               m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
110      {
```

```
111         Ipv4InterfaceAddress iface = j->second;
112
113         if (d_ipv4->GetInterfaceForAddress (iface.GetLocal ()) == iif)
114           {
115             if (dst == iface.GetBroadcast () || dst.IsBroadcast ())
116               {
117                 if (lcb.IsNull () == false)
118                   {
119                     NS_LOG_LOGIC ("Broadcast local delivery to " << iface.GetLocal ());
120                     lcb (p, header, iif);
121                   }
122                 else
123                   {
124                     NS_LOG_ERROR ("Unable to deliver packet locally due to null 
                            callback " << p->GetUid () << " from " << origin);
125                     ecb (p, header, Socket::ERROR_NOROUTETOHOST);
126                   }
127                 return true;
128               }
129           }
130       }
131
132   // Si la dirección del destino no es la dirección de broadcast
133   if (d_ipv4->IsDestinationAddress (dst, iif))
134     {
135       if (lcb.IsNull () == false)
136         {
137           NS_LOG_LOGIC ("Unicast local delivery to " << dst);
138           lcb (p, header, iif);
139
140         }
141       else
142         {
143           NS_LOG_ERROR ("Unable to deliver packet locally due to null callback " <<
                  p->GetUid () << " from " << origin);
144           ecb (p, header, Socket::ERROR_NOROUTETOHOST);
145         }
146       return true;
147     }
148
149   return Forwarding (p, header, ucb, ecb); // Forwarding sólo es usado para enviar
        el paquete de datos
150 }
151
152 // Envía un paquete UDP proveniente de la aplicación en el nodo origen.
153 bool
154 RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
155                              UnicastForwardCallback ucb, ErrorCallback ecb)
156 {
157   NS_LOG_FUNCTION (this);
158
159   Ipv4Address dst = header.GetDestination ();
```

```
160     Ipv4Address origin = header.GetSource ();
161     RoutingTableEntry toDst;
162 
163     if (d_routingTable.LookupRoute (dst, toDst))
164       {
165         Ptr<Ipv4Route> route = toDst.GetRoute ();
166         std::cout << route->GetSource () << " forwarding to " << dst << " from " <<
               origin << " packet " << p->GetUid () << std::endl;
167         NS_LOG_LOGIC (route->GetSource () << " forwarding to " << dst << " from " <<
                 origin << " packet " << p->GetUid ());
168 
169         ucb (route, p, header);
170         return true;
171       }
172     NS_LOG_LOGIC ("route not found to "<< dst << ".");
173     return false;
174 }
175 
176 // Asigna el índice del destino indicando su posición en el mapa donde está guardado
       .
177 void
178 RoutingProtocol::SetIndexDest (uint32_t indDst)
179 {
180   indexDstApp = indDst;
181 }
182 
183 // Notifica al protocolo de ruteo que una determinada interface está lista para ser
       utilizada.
184 void
185 RoutingProtocol::NotifyInterfaceUp (uint32_t i)
186 {
187   NS_LOG_FUNCTION (this << d_ipv4->GetAddress (i, 0).GetLocal ());
188   Ptr<Ipv4L3Protocol> l3 = d_ipv4->GetObject<Ipv4L3Protocol> ();
189   nodes_sockAddr dataNodes;
190 
191   Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
192   dataNodes.ifNode = iface;
193   Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
194                                              UdpSocketFactory::GetTypeId ());
195 
196   NS_ASSERT (socket != 0);
197   socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvPackDiscovery, this)
       );
198   socket->Bind (InetSocketAddress (Ipv4Address::GetAny (), 80));
199   socket->BindToNetDevice (l3->GetNetDevice (i));
200   socket->SetAllowBroadcast (true);
201   m_socketAddresses.insert (std::make_pair (socket, iface));
202 
203   key++;
204   dataNodes.sockNode = socket;
205   m_sockIfNodes.insert (std::make_pair (key, dataNodes));
206 
```

```
207    // Agregar a la tabla de enrutamiento una ruta de broadcast para los paquetes
         discovery.
208    Ptr<NetDevice> dev = d_ipv4->GetNetDevice (d_ipv4->GetInterfaceForAddress (iface.
         GetLocal ()));
209
210    RoutingTableEntry rt (dev, iface.GetBroadcast (), 0, iface, 0, iface.GetBroadcast
         ()); /*Crear una entrada para la tabla de ruteo.*/
211    d_routingTable.AddRoute (rt);
212  }
213
214  // Notifica que una interfaz no está disponible para ser usada por la capa IP.
215  void
216  RoutingProtocol::NotifyInterfaceDown (uint32_t i)
217  {
218
219  }
220
221  // Notifica al protocolo de que una dirección ip está siendo agregada a una
       interfaz.
222  void
223  RoutingProtocol::NotifyAddAddress (uint32_t i, Ipv4InterfaceAddress address)
224  {
225
226  }
227
228  // Notifica cuando una dirección IPv4 es removida de una interfaz.
229  void
230  RoutingProtocol::NotifyRemoveAddress (uint32_t i, Ipv4InterfaceAddress address)
231  {
232
233  }
234
235  // Crea varios eventos de difusión de paquetes discovery desde el destino.
236  void
237  RoutingProtocol::CreateEvBroadDst (double tStopApp, double intervBroadDest)
238  {
239    while (d_timeStartDiscv < Seconds (tStopApp))
240      {
241        Simulator::Schedule (d_timeStartDiscv, &RoutingProtocol::SendDiscovery, this);
242        d_timeStartDiscv = d_timeStartDiscv + Seconds (intervBroadDest);
243      }
244  }
245
246  // Crea un objeto Ipv4 el cual permite el acceso a las tablas de ruteo, interfaces,
       y configuración.
247  void
248  RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
249  {
250    NS_ASSERT (ipv4 != 0);
251    NS_ASSERT (d_ipv4 == 0);
252
253    d_ipv4 = ipv4;
```

```
254 }
255
256 // Difunde un paquete discovery desde el destino.
257 void
258 RoutingProtocol::SendDiscovery ()
259 {
260   DiscvHeader discvHeader;
261   nodes_sockAddr destSockIface;
262
263   destSockIface = m_sockIfNodes[indexDstApp];
264   Ipv4InterfaceAddress addrDest = destSockIface.ifNode;
265   Ptr<Socket> socketDest = destSockIface.sockNode;
266
267   packetId++;
268   discvHeader.SetId (packetId);
269   discvHeader.SetHopCount (0);
270   discvHeader.SetDst (addrDest.GetLocal ());
271
272   Ptr<Packet> packet = Create<Packet> ();
273   packet->AddHeader (discvHeader);
274   TypeHeader tHeader (DISCOVERYTYPE_DISCV);
275   packet->AddHeader (tHeader);
276
277   Ipv4Address destination;
278   destination = addrDest.GetBroadcast ();
279   socketDest->SendTo (packet, 0, InetSocketAddress (destination, 80));
280 }
281
282 // Método invocado por un evento para enviar un paquete discovery cada cierto
      tiempo.
283 void
284 RoutingProtocol::SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address
      destination)
285 {
286   socket->SendTo (packet, 0, InetSocketAddress (destination, 80));
287 }
288
289 // Este método recibe un paquete discovery, en base al cual busca o actualiza una
      ruta para su destino en la tabla de ruteo.
290 // Programa un nuevo evento de envío por el nodo que está recibiendo el paquete.
291 void
292 RoutingProtocol::RecvPackDiscovery (Ptr<Socket> socket)
293 {
294   Address sourceAddress;
295   Ptr<Packet> p = socket->RecvFrom (sourceAddress);
296   InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress)
        ;
297   Ipv4Address sender = inetSourceAddr.GetIpv4 ();
298   Ipv4Address receiver = m_socketAddresses[socket].GetLocal ();
299
300   TypeHeader tHeader (DISCOVERYTYPE_DISCV);
301   p->RemoveHeader (tHeader);
```

```
302
303   DiscvHeader discvHeader;
304   p->RemoveHeader (discvHeader);
305   uint16_t id = discvHeader.GetId ();
306   uint32_t hop = discvHeader.GetHopCount () + 1;
307   discvHeader.SetHopCount (hop);
308   discvHeader.SetSender (sender);
309
310   RoutingTableEntry toDest;
311
312   // Si no hay una entrada en la tabla para el destino, registrar una entrada para
        él.
313   if (!d_routingTable.LookupRoute (discvHeader.GetDst (), toDest))
314     {
315       Ptr<NetDevice> dev = d_ipv4->GetNetDevice (d_ipv4->GetInterfaceForAddress (
              receiver));
316       RoutingTableEntry newEntry (dev, discvHeader.GetDst (), discvHeader.GetId (),
317                                   d_ipv4->GetAddress (d_ipv4->
                                          GetInterfaceForAddress (receiver), 0),
318                                   discvHeader.GetHopCount (), discvHeader.GetSender
                                          ());
319
320       d_routingTable.AddRoute (newEntry);
321     }
322
323    // Actualizar tabla de ruteo.
324    else if (discvHeader.GetHopCount () <= toDest.GetHop ())
325     {
326       toDest.SetOutputDevice (d_ipv4->GetNetDevice (d_ipv4->GetInterfaceForAddress
              (receiver)));
327       toDest.SetDestination (discvHeader.GetDst ());
328       toDest.SetUltId (discvHeader.GetId ());
329       toDest.SetInterface (d_ipv4->GetAddress (d_ipv4->GetInterfaceForAddress (
              receiver), 0));
330       toDest.SetHop (discvHeader.GetHopCount ());
331       toDest.SetNextHop (discvHeader.GetSender ());
332
333       d_routingTable.Update (toDest);
334     }
335
336   // Verificar si un paquete discovery es duplicado.
337   if (d_packsDiscvIdCache.IsDuplicate (sender, id))
338     {
339       NS_LOG_DEBUG ("Ignoring discovery due to duplicate");
340       return;
341     }
342
343   // Difundir paquetes discovery desde cada interfaz.
344   for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
345          m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
346     {
347       Ptr<Socket> socket = j->first;
```

```
348         Ipv4InterfaceAddress iface = j->second;
349
350         Ptr<Packet> packet = Create<Packet> ();
351
352         packet->AddHeader (discvHeader);
353         TypeHeader tHeader (DISCOVERYTYPE_DISCV);
354         packet->AddHeader (tHeader);
355
356         Ipv4Address destination;
357         destination = iface.GetBroadcast ();
358
359         Simulator::Schedule (Seconds (timeInitBroad), &RoutingProtocol::SendTo, this,
                 socket,
360                                 packet, destination);
361         timeInitBroad = timeInitBroad + d_broadInterval;
362       }
363 }
364
365 // Imprime las tablas de ruteo.
366 void
367 RoutingProtocol::PrintRoutingTable (Ptr<OutputStreamWrapper> stream) const
368 {
369 }
370
371 }
372 }
```

El archivo comienza con la inclusión de diferentes librerías utilizadas por el módulo en desarrollo. Las declaraciones al comienzo del módulo son similares a los módulos explicados más arriba. Nos limitaremos a explicar las particularidades del nuevo módulo y, en general, las líneas necesarias para comprender su funcionamiento.

Se debe notar que en la línea 12 del código aparece la definición de un espacio de nombres particular al módulo en descripción. Esto es común a la mayoría de los módulos que se encuentran como ejemplo en ns-3.

En la línea 15, la macro `NS_OBJECT_ENSURE_REGISTERED` hace registro actual de la clase `RoutingProtocol` en el sistema.

Las siguientes líneas

```
18 struct nodes_sockAddr {
19   Ptr<Socket> sockNode;
20   Ipv4InterfaceAddress ifNode;
21   };
22
23 // Mapa que contiene la interfaz y el socket de cada nodo
24 uint16_t key = 0;
25 std::map<uint16_t, nodes_sockAddr> m_sockIfNodes;
```

definen estructuras necesarias para almacenar los componentes Socket y las interfaces Ipv4 de todos los nodos. Esto será utilizado más adelante para poder acceder a estos componentes del nodo destino, y para que este pueda difundir packetes de datos. La clase map sirve para implementar contenedores asociativos direccionados por un identificador único. En este caso, se define el contenedor `m_sockIfNodes`, que almacenará el par de objetos contenidos en la estructura `nodes_sockAddr`, los que serán direccionados por un entero de 16 bits. La variable global `key` será utilizada para registrar los índices generados para direccionar cada elemento en el objeto `nodes_sockAddr` .

Las 2 líneas siguientes definen las variables globales `packetId` y `timeInitBroad`, destinadas a almacenar un identificador secuencial para los packetes Discovery y el tiempo en el que un nodo vecino al nodo Destino realizará la

primera redifusión del primer mensaje Discovery recibido, respectivamente.

Luego, se comienza con la implementación de los diferentes métodos de la clase `RoutingProtocol`, comenzando por los métodos constructor (`RoutingProtocol::RoutingProtocol ()`) y destructor (`RoutingProtocol::~RoutingProtocol ()`). El método constructor llama a los constructores de varios objetos inicializando algunos de sus atributos. Los objetos y atributos inicializados por el método constructor se resumen en la Tabla 1.

Cuadro 1: Atributos inicializados en método constructor de la clase `RoutingProtocol()`.

Objeto	Atributo inicializado	Valor	Descripción
`IdCache d_packsDiscvIdCache`	`Time m_lifetime`	`MilliSeconds (5600)`	Declarado en `discovery_id_cache.h`. Indica el tiempo que permanecerá registrado un identificador compuesto de una dirección IP, de un identificador del paquete, y un tiempo de vida, en un vector utilizado por el objeto de la clase `IdCache`.
`RoutingTable d_routingTable`	`Time m_badLinkLifetime`	`Time (15)`	Declarado en `discovery-rtable.h`. Tiempo no utilizado por la imlpementación descrita, pero es necesario inicializar.
`Time d_timeStartDiscv`	(atributo interno de la clase Time de ns3)	`d_timeStartDiscv (Seconds (1))`	Declarado en `discovery-routing-protocol.h`. Indica el tiempo en que se inicia cada evento de difusión de un paquete discovery desde el nodo Destino. Este tiempo se incrementa en el evento `CreateEvBroadDst`.

Además, el atributo público `d_broadInterval` (`double`, incializado en 0,005) será utilizado para indicar el intérvalo de tiempo que transcurrirá entre dos eventos de redifusión de mensajes Discovery por parte de los nodos intermedios.

El método GetTypeId (línea 44) es utilizado comunmente por los módulos implementados en ns3 para retornar el identificador (clase `TypeId`) de la clase en cuestión.

Las siguientes líneas implementan el método `RouteOutput` de la clase `RoutingProtocol`:

```
58 Ptr<Ipv4Route>
59 RoutingProtocol::RouteOutput (Ptr<Packet> p, const Ipv4Header &header,
60                               Ptr<NetDevice> oif, Socket::SocketErrno &sockerr)
61 {
62   sockerr = Socket::ERROR_NOTERROR;
63   Ptr<Ipv4Route> route;
64   Ipv4Address dst = header.GetDestination ();
65
66   RoutingTableEntry rt;
67
68   // Buscar una ruta en la tabla de enrutamiento.
69   if (d_routingTable.LookupRoute (dst, rt))
70     {
71       route = rt.GetRoute ();
72       NS_ASSERT (route != 0);
```

```
73        NS_LOG_DEBUG ("Exist route to " << route->GetDestination () << " from 
              interface " << route->GetSource ());
74
75        if (oif != 0 && route->GetOutputDevice () != oif)
76          {
77            NS_LOG_DEBUG ("Output device doesn't match. Dropped.");
78            sockerr = Socket::ERROR_NOROUTETOHOST;
79            return Ptr<Ipv4Route> ();
80          }
81
82        return route;
83      }
84    else
85     {
86       return Ptr<Ipv4Route> ();
87     }
88 }
```

Este corresponde a un método virtual que se debe implementar cuando se codifiquen clases que hereden de la superclase `Ipv4RoutingProtocol`. Este método es ejecutado cada vez que un nodo desea transmitir un paquete de datos a un destino determinado o a un grupo de nodos. En el caso de esta aplicación, esto ocurre cuando el nodo Origen desea transmitir un paquete de datos (de Aplicación) al nodo Destino, o cuando un nodo intermedio redifunde un paquete Discovery a sus vecinos (a la dirección de Broadcast). Este método recibe como parámetros:

- El paquete (`Ptr<Packet>p`).
- El encabezado del paquete (`const Ipv4Header &header`).
- El dispositivo de red que requiere buscar una ruta para transmitir un paquete (`Ptr<NetDevice>oif`).
- Un tipo numerado de la clase `Socket` (parámetro `sockerr`) que se utilizará para indicar un posible error en la utilización de esta clase. Por ejemplo, la línea 78 hace referencia al error `Socket::ERROR_NOROUTETOHOST` utilizado para indicar que no se ha encontrado una ruta para el destino.

El método comienza con la inicialización del tipo numerado sockerr en `Socket::ERROR_NOTERROR`, indicando que, hasta el momento, no hay un error asociado a la utilización del socket. Luego, se definen los objetos `Ptr<Ipv4Route>route`, `Ipv4Address dst` y `RoutingTableEntry rt`. El primero es utilizado para almacenar la ruta para el paquete a rutear. El segundo, corresponde a la dirección IP de destino para el paquete a rutear, la que se obtiene desde el encabezado del paquete (que entra por parámetro) a través del método `header.GetDestination ()`. El tercero, se utilizará para almacenar un registro (entrada) de la tabla de ruteo.

La condición en la línea 69 hace llamado al método `LookupRoute` del objeto `d_routingTable` que corresponde a la clase `RoutingTable`. `LookupRoute` busca una entrada (`rt`) para el destino `dst` (en la tabla de ruteo). En caso de encontrarse la entrada (LookupRoute retorna “Verdadero”), se asigna al objeto `route` la ruta encontrada por `LookupRoute`. La macro `NS_ASSERT` valida una condición lógica, en este caso que la ruta no sea igual a cero.

La condición en la línea **??** valida que el dispositivo que solicita la ruta sea válido y esté correctamente registrado en el objeto `route` como el dispositivo que va a buscar una ruta para el paquete de salida. Si no corresponde, se registra y se muestra el error. Sino, se retorna la ruta encontrada previamente (`return route`).

Las siguientes líneas implementan el método `RouteInput` de la clase `RoutingProtocol`:

```
92 bool
93 RoutingProtocol::RouteInput (Ptr<const Packet> p, const Ipv4Header &header,
94                              Ptr<const NetDevice> idev, UnicastForwardCallback ucb,
95                              MulticastForwardCallback mcb, LocalDeliverCallback lcb
                                  , ErrorCallback ecb)
```

```
96   {
97
98     NS_LOG_FUNCTION (this << p->GetUid () << header.GetDestination () << idev->
         GetAddress ());
99     NS_ASSERT (d_ipv4 != 0);
100    NS_ASSERT (p != 0);
101    NS_ASSERT (d_ipv4->GetInterfaceForDevice (idev) >= 0);
102
103    int32_t iif = d_ipv4->GetInterfaceForDevice (idev);
104    Ipv4Address dst = header.GetDestination ();
105    Ipv4Address origin = header.GetSource ();
106
107    // Liberar localmente el paquete recibido
108    for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
109           m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
110      {
111        Ipv4InterfaceAddress iface = j->second;
112
113        if (d_ipv4->GetInterfaceForAddress (iface.GetLocal ()) == iif)
114          {
115            if (dst == iface.GetBroadcast () || dst.IsBroadcast ())
116              {
117                if (lcb.IsNull () == false)
118                  {
119                    NS_LOG_LOGIC ("Broadcast local delivery to " << iface.GetLocal ());
120                    lcb (p, header, iif);
121                  }
122                else
123                  {
124                    NS_LOG_ERROR ("Unable to deliver packet locally due to null
                         callback " << p->GetUid () << " from " << origin);
125                    ecb (p, header, Socket::ERROR_NOROUTETOHOST);
126                  }
127                return true;
128              }
129          }
130      }
131
132    // Si la dirección del destino no es la dirección de broadcast
133    if (d_ipv4->IsDestinationAddress (dst, iif))
134      {
135        if (lcb.IsNull () == false)
136          {
137            NS_LOG_LOGIC ("Unicast local delivery to " << dst);
138            lcb (p, header, iif);
139
140          }
141        else
142          {
143            NS_LOG_ERROR ("Unable to deliver packet locally due to null callback " <<
                 p->GetUid () << " from " << origin);
144            ecb (p, header, Socket::ERROR_NOROUTETOHOST);
```

```
145            }
146          return true;
147        }
148
149      return Forwarding (p, header, ucb, ecb); // Forwarding sólo es usado para enviar
             el paquete de datos
150  }
```

Al igual que `RouteOutput`, este corresponde a un método virtual que se debe implementar cuando se codifican clases que heredan de `Ipv4RoutingProtocol`. Este método es ejecutado cada vez que un paquete IP está siendo recibido desde la interfaz de red del nodo en su rol de receptor en el nivel de red de la pila de Internet. RouteInput determinará si dicho parquete será liberado localmente (es decir, si será liberado a las capas de nivel superior de este nodo) o si será reenviado a otros nodos mediante el método `Forwarding`. En ambos casos, este método retorna verdadero, y falso en otros casos. En el caso de esta aplicación, esto ocurre cuando un nodo recibe un paquete de datos de Aplicación un paquete Discovery. Este método recibe como parámetros:

- El paquete recibido (`Ptr<const Packet>p`).
- El encabezado del paquete del paquete `p` (`const Ipv4Header &header`).
- El dispositivo de red que está recibiendo el paquete (`Ptr<const NetDevice>idev`).
- La función que será invocada cuando el paquete se envíe a un único receptor (`UnicastForwardCallback ucb`).
- La función que será invocada cuando el paquete se envíe a una dirección multicast (`MulticastForwardCallback mcb`).
- La función que será invocada cuando el paquete se libere localmente, es decir, cuando se entregue al protocolo de capa superior (`LocalDeliverCallback lcb`).
- La función que se utilizará cuando una ruta no se encuentre o cuando haya un error en el envío (`ErrorCallback ecb`).

El método comienza con la verificación de varios objetos utilizando la macro `NS_ASSERT`. Pasando estas verificaciones, se declara la variable `int32_t iif` a la que se asigna el índica de la interfaz del nodo que está recibiendo el paquete de datos, con ayuda del método `GetInterfaceForDevice()`. Luego, se declaran los objetos `dst` y `origin` de la clase `Ipv4Address` a los que se asignan las direcciones del destino y fuente del paquete recibido, obtenidas desde el encabezado `header`.

El ciclo que comienza en la línea 108 recorre el mapa `m_socketAddresses` que contiene el socket del nodo que recibe el paquete (el mapa `m_socketAddresses` se llena llena en el método `NotifyInterfaceUp` explicado más adelante). En cada registro de `m_socketAddresses` están almacenados (primero) el socket y (segundo) un objeto de la clase `Ipv4InterfaceAddress` que contiene diferentes datos acerca de la configuración IP del nodo local (el que recibe). La línea `Ipv4InterfaceAddress iface = j->second;` permite recuperar el segundo elemento mencionado del `j`ésimo registro y asignarlo al objeto `iface`.

`d_ipv4` (línea 113) es un atributo del tipo `Ptr<Ipv4>` que permite el acceso a las tablas de ruteo, interfaces de red y la configuración de estas; es el objeto que permite manipular aspectos del protocolo IPv4 relacionados con (según `https://www.nsnam.org/doxygen/classns3_1_1_ipv4.html#details`):

- la obtención o configuración de un protocolo de ruteo IPv4 Ipv4RoutingProtocol,
- el registro de un dispositivo de red (NetDevice) para utilizarlo por la capa IPv4,
- la manipulación del estado del `NetDevice` desde la perspectiva de Ipv4 (como por ejemplo, marcarlo como Up o Down),
- la agregación, eliminación y obtención de las direcciones asociadas a una interfaz Ipv4, y

- exportar los parámetros de configuración Ipv4.

Si el índice de la interfaz del nodo registrado en el socket es idéntico al del nodo local (el que recibe el mensaje), se verifica que la dirección de destino (`dst`) sea una dirección de difusión (broadcast). El método `GetBroadcast ()` retorna la dirección de difusión asociada a la configuración de una interfaz IPv4. El método `IsBroadcast ()` retorna verdadero si el objeto Ipv4Address tiene asignada la dirección `255.255.255.255`. Si se cumple esta condición se verifica que la función call back tenga asociada una implementación. Si es así, se le entrega el paquete, junto con su encabezado, y el índice de la interfaz de red. Si no se cumple esta última condición se mostrará y registrará el error a través de la call back `ecb`. En cualquiera de los dos últimos casos (se haya entregado el paquete a la capa superior, o se haya registrado el error), el método `RouteInput` retornará un Verdadero.

En la condición de la línea 133, el método IsDestinationAddress retornará Verdadero si la dirección de destino (`dst`) corresponde a la dirección asociada a la interface indicada por `iif`. Si es así, al igual que en la etapa anterior, el paquete será enviado a la aplicación (la función call back) en la capa superior (si ésta está correctamente definida), y retornará Verdadero. Finalmente, si ninguna de las condiciones anteriores que permiten retornar Verdadero se ejecutan (si la dirección local no es equivalente a la dirección de destino o a una de broadcast) el nodo retransmitirá el paquete con ayuda del método `Forwarding`.

Las siguientes líneas implementan el método `Forwarding` de la clase `RoutingProtocol`:

```
153 bool
154 RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
155                              UnicastForwardCallback ucb, ErrorCallback ecb)
156 {
157   NS_LOG_FUNCTION (this);
158
159   Ipv4Address dst = header.GetDestination ();
160   Ipv4Address origin = header.GetSource ();
161   RoutingTableEntry toDst;
162
163   if (d_routingTable.LookupRoute (dst, toDst))
164     {
165       Ptr<Ipv4Route> route = toDst.GetRoute ();
166       std::cout << route->GetSource () << " forwarding to " << dst << " from " <<
              origin << " packet " << p->GetUid () << std::endl;
167       NS_LOG_LOGIC (route->GetSource () << " forwarding to " << dst << " from " <<
              origin << " packet " << p->GetUid ());
168
169       ucb (route, p, header);
170       return true;
171     }
172   NS_LOG_LOGIC ("route not found to "<< dst << ".");
173   return false;
174 }
```

Este método recibe como parámetros:

- El paquete recibido (`Ptr<const Packet>p`).
- El encabezado del paquete del paquete `p` (`const Ipv4Header &header`).
- La función que será invocada cuando el paquete se envíe a un único receptor (`UnicastForwardCallback ucb`).
- La función que se utilizará cuando una ruta no se encuentre o cuando haya un error en el envío (`ErrorCallback ecb`).

Si el nivel LOG_FUNCTION está habilitado, la macro NS_LOG_FUNCTION muestra todos los parámetros que estén en el paréntesis separados por coma. Al igual que el método anterior, se comienza por definir los objetos dst y origin para almacenar las direcciones de destino y origen. toDst será un registro de la tabla de ruteo. El resto es equivalente a lo explicado en el método RouteOutput. Si existe una ruta (asignada a toDst) con el método LookupRoute para el destino dst, entonces se envía el paquete al siguiente salto (indicado por route). Esto es realizado por la función call back[3].

El método SetIndexDest (línea 177), simplemente sirve para asignar el índica asociado al nodo destino (registrado en el mapa m_sockIfDest) que se utiliza en el método NotifyInterfaceUp que se explica a continuación, y en el método SendDiscovery explicado más adelante.

Las siguientes líneas implementan el método NotifyInterfaceUp de la clase RoutingProtocol:

```
184 void
185 RoutingProtocol::NotifyInterfaceUp (uint32_t i)
186 {
187   NS_LOG_FUNCTION (this << d_ipv4->GetAddress (i, 0).GetLocal ());
188   Ptr<Ipv4L3Protocol> l3 = d_ipv4->GetObject<Ipv4L3Protocol> ();
189   nodes_sockAddr dataNodes;
190
191   Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);
192   dataNodes.ifNode = iface;
193   Ptr<Socket> socket = Socket::CreateSocket (GetObject<Node> (),
194                                              UdpSocketFactory::GetTypeId ());
195
196   NS_ASSERT (socket != 0);
197   socket->SetRecvCallback (MakeCallback (&RoutingProtocol::RecvPackDiscovery, this)
        );
198   socket->Bind (InetSocketAddress (Ipv4Address::GetAny (), 80));
199   socket->BindToNetDevice (l3->GetNetDevice (i));
200   socket->SetAllowBroadcast (true);
201   m_socketAddresses.insert (std::make_pair (socket, iface));
202
203   key++;
204   dataNodes.sockNode = socket;
205   m_sockIfNodes.insert (std::make_pair (key, dataNodes));
206
207   // Agregar a la tabla de enrutamiento una ruta de broadcast para los paquetes
        discovery.
208   Ptr<NetDevice> dev = d_ipv4->GetNetDevice (d_ipv4->GetInterfaceForAddress (iface.
        GetLocal ()));
209
210   RoutingTableEntry rt (dev, iface.GetBroadcast (), 0, iface, 0, iface.GetBroadcast
        ()); /*Crear una entrada para la tabla de ruteo.*/
211   d_routingTable.AddRoute (rt);
212 }
```

Este método, entre otras cosas, sirve para informar al protocolo de ruteo que la interfaz está lista para ser utilizada. NotifyInterfaceUp recibe un único parámetro de tipo uint32_t correspondiente al índice de la interfaz acerca de la que se está notificando.

La línea Ptr<Ipv4L3Protocol>l3 = d_ipv4->GetObject<Ipv4L3Protocol>(); define un puntero a un objeto de la clase IPv4L3Protocol. Esta clase implementa el protocolo Internet (v4) (ver https://www.nsnam.org/doxygen/classns3_

[3]La función de call back está definida en la clase Ipv4RoutingProtocol, la clase base para todos los protocolos de enrrutamiento implementados en ns3.

Luego, se define el objeto dataNodes (clase `nodes_sockAddr`) que servirá para obtener el socket y la interface IPv4 de cada nodo. Con la línea `Ipv4InterfaceAddress iface = l3->GetAddress (i, 0);` se define el objeto `iface`, al que se asigna un Ipv4InterfaceAddress obtenido, gracias al método `GetAddress` del objeto `l3`, tomando como referencia el índice de la interface ingresado por parámetro. El segundo parámetro se refiere al índice de la dirección a utilizar por la interface; como en este caso sólo se utiliza una dirección por interfaz, este valor es cero (la primera y única dirección).

Con la línea `dataNodes.ifNode = iface;` se asigna el objeto `iface` al campo `ifNode` del objeto `dataNodes` que hace referencia a la interfaz de un nodo.

Luego, análogo a ejemplos anteriores, se crea un un puntero a un socket (en este caso UDP) en el nodo actual (línea `Ptr<Socket>socket = Socket::CreateSocket (GetObject<Node>(), UdpSocketFactory::GetTypeId ());`. Luego de validar el socket, se define la Callback que se ejecutará cuando haya un nuevo paquete para ser leído. En este caso, esta función corresponde al método `RecvPackDiscovery` que se expicará más adelante.

Luego, al socket se enlaza un objeto de la clase `InetSocketAddress` que asocia la dirección IP y el número de puerto a utilizar. Seguido, con la línea `socket->BindToNetDevice (l3->GetNetDevice (i));` se enlaza al socket al dispositivo de red (indicado por la interface `i`).

La línea `socket->SetAllowBroadcast (true);` permite al socket transmitir datos a la dirección de broadcast.

El objeto `m_socketAddresses` es un atributo privado de la clase `RoutingProtocol`. Corresponde a un mapa (contenedor asociativo tipo `std::map`) que contiene un puntero a un socket y una interface IPv4 (ver declaración de la clase en Listado 8). En la línea `m_socketAddresses.insert (std::make_pair (socket, iface));` se inserta en el mapa la tupla (`socket, iface`). Recordando lo dicho en la página 34, se declaró el mapa global `m_sockIfNodes;` para facilitar la obtención de los datos de todos los nodos, y que sería utilizado en específico para obtener los datos del destino. Además, se declaró la variable key (inicializada en cero), como identificador para los diferentes elementos de `m_sockIfNodes`.

Luego de incremenentar el índice `key` y de asignar el socket al objeto `dataNodes` se inserta en el mapa `m_sockIfNodes` el par (`key, dataNodes`). Luego, se procede a agregar a la tabla de enrutamiento una ruta de broadcast para los paquetes discovery. Primero, se define un puntero a un objeto `NetDevice`, llamado `dev`, al que se asigna el puntero `NetDevice` obtenido por el método `GetNetdevice` del objeto `d_ipv4` (atributo privado de la clase `RoutingProtocol`) utilizando como entrada la interface que se obtiene en base a la dirección local).

En la línea `RoutingTableEntry rt (dev, iface.GetBroadcast (), 0, iface, 0, iface.GetBroadcast ());` se define una entrada para la tabla de ruteo, utilizando como parámetros el dispositivo de red `dev` (recién definido), la dirección de destino (en este caso, la dirección de broadcast), el identificador del paquete recibido (inicializado aquí en cero), el objeto `iface`, el número de saltos del paquete (inicializado en cero), y el siguiente salto (en este caso, la dirección de broadcast, también).

Finalmente, se agrega la entrada `rt` a la tabla de ruteo contenida en el objeto `d_routingTable` (inicializado en el método constructor de `RoutingProtocol`.

El método `NotifyInterfaceDown` (línea 215) es un método virtual de `Ipv4RoutingProtocol`. Análogo al método anterior, éste está destinado a informar al protocolo de ruteo que la interfaz no está lista para ser utilizada o se encuentra inaccesible, inhabilitando cualquier tráfico IP a través de la interfaz. Este método no se utiliza en este ejemplo.

El método `NotifyAddAddress` (línea 222) es un método virtual de `Ipv4RoutingProtocol`, que se utiliza para notificar que una dirección se está agregando a una interfaz. . Este método no se utiliza en este ejemplo.

El método `NotifyRemoveAddress` (línea 229) es un método virtual de `Ipv4RoutingProtocol`, que se utiliza para notificar que una dirección está siendo removida de una interfaz. Este método no se utiliza en este ejemplo.

Las siguientes líneas implementan el método `CreateEvBroadDst` de la clase `RoutingProtocol`:

```
236 void
237 RoutingProtocol::CreateEvBroadDst (double tStopApp, double intervBroadDest)
238 {
239   while (d_timeStartDiscv < Seconds (tStopApp))
240     {
241       Simulator::Schedule (d_timeStartDiscv, &RoutingProtocol::SendDiscovery, this);
242       d_timeStartDiscv = d_timeStartDiscv + Seconds (intervBroadDest);
```

```
243     }
244 }
```

Este método programa los eventos de difusión un un nuevo mensaje Discovery por parte del nodo Destino, según el protocolo de ejemplo decrito más arriba (método `Schedule` de la clase `Simulator`). Los eventos de difusión de mensajes serán programados cada `intervBroadDest`. El ciclo while se realizará mientras `d_timeStartDiscv` (atributo privado de `RoutingProtocol` inicializado en el constructor de esta clase en 1 segundo) sea menor al tiempo (en segundos) en que la aplicación terminará. Este tiempo está contenido en el parámetro `tStopApp`. Por cada ciclo, se programa un evento, o la ejecución de un procedimiento, con el método Schedule de la clase Simulator, al que ingresan por parámetros: el tiempo en el que se ejecutará el evento (`d_timeStartDiscv`), la función a realizarse (en este caso, el método `RoutingProtocol::SendDiscovery`), y el argumento a pasar al método (en este caso, el puntero `this`). Luego de esto el atributo `d_timeStartDiscv` es incrementado en `intervBroadDest` segundos.

Las siguientes líneas implementan el método `SetIpv4` de la clase `RoutingProtocol`:

```
247 void
248 RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
249 {
250   NS_ASSERT (ipv4 != 0);
251   NS_ASSERT (d_ipv4 == 0);
252
253   d_ipv4 = ipv4;
254 }
```

Este un método virtual de la clase `Ipv4RoutingProtocol`. Es invocado por el método `SetRoutingProtocol` de la clase `Ipv4` (método virtual implementado, en realidad, por la clase `Ipv4L3Protocol`). `SetRoutingProtocol` se invoca en el método `Install` de la clase `InternetStackHelper`, llamado en la función principal de la simulación (`main`). El método se ejecuta al momento de establecer el protocolo de ruteo a utilizar, por cada nodo. La implementación en `RoutingProtocol::SetIpv4`, luego de verificar los valores por defecto de los punteros `ipv4` y `d_ipv4`, asigna a `d_ipv4` el objeto apuntado por `ipv4`.

Las siguientes líneas implementan el método `SendDiscovery` de la clase `RoutingProtocol`:

```
257 void
258 RoutingProtocol::SendDiscovery ()
259 {
260   DiscvHeader discvHeader;
261   nodes_sockAddr destSockIface;
262
263   destSockIface = m_sockIfNodes[indexDstApp];
264   Ipv4InterfaceAddress addrDest = destSockIface.ifNode;
265   Ptr<Socket> socketDest = destSockIface.sockNode;
266
267   packetId++;
268   discvHeader.SetId (packetId);
269   discvHeader.SetHopCount (0);
270   discvHeader.SetDst (addrDest.GetLocal ());
271
272   Ptr<Packet> packet = Create<Packet> ();
273   packet->AddHeader (discvHeader);
274   TypeHeader tHeader (DISCOVERYTYPE_DISCV);
275   packet->AddHeader (tHeader);
276
277   Ipv4Address destination;
278   destination = addrDest.GetBroadcast ();
```

```
279   socketDest->SendTo (packet, 0, InetSocketAddress (destination, 80));
280 }
```

El método `SendDiscovery` se ejecuta según la programación realizada en el método `CreateEvBroadDst` (explicado más arriba). El método comienza con la declaración de los objetos `discvHeader` y `destSockIface` de las clases `DiscvHeader` y `nodes_sockAddr`, respectivamente. El primero será utilizado para almacenar el encabezado del paquete discovery, mientras que el segundo será utilizado para guardar el socket y la interfaz IPv4. En la línea 263, `destSockIface` toma los valores del mapa `m_sockIfNodes` en el registro con índice `indexDstApp`. `m_sockIfNodes` está definido como un objeto global dentro de las primeras líneas al comienzo del espacio de nombres discovery (ver página 34).

En la línea `Ipv4InterfaceAddress addrDest = destSockIface.ifNode;` se declara el objeto `addrDest` (clase `Ipv4InterfaceAddress`), al que se asigna la interface IPv4 del nodo destino (almacenada en el campo `ifnode` de la estructura `nodes_sockAddr`. La línea `Ptr<Socket>socketDest = destSockIface.sockNode;` declara un puntero a un objeto Socket, en este caso, al socket del nodo destino (que se encuentra en el campo sockNode de la estructura `nodes_sockAddr`).

La línea `packetId++;` incrementa el índice del paquete Discovery a transmitir. Luego, se definen los atributos del encabezado del paquete Discovery, particularmente, su identificador, el contador de saltos y la dirección del nodo destino. En `Ptr<Packet>packet = Create<Packet>();` se declara un puntero a un objeto Packet. Luego, a dicho paquete se agregan (gracias al método AddHeader) los componentes del encabezado del paquete; en este caso, el objeto con la estructura de campos del encabezado (el objeto DiscvHeader) y el tipo de encabezado (en este caso `DISCOVERYTYPE_DISCV`, asignado mediante el objeto texttttHeader de la clase `TypeHeader`). Luego, de forma análoga a lo explicado en métodos anteriores, se define y configura un objeto de la clase `Ipv4Address`, para luego transmitir el paquete con ayuda del método `SendTo` del objeto `Socket` declarado anteriormente.

Las siguientes líneas implementan el método `SendTo` de la clase `RoutingProtocol`:

```
283 void
284 RoutingProtocol::SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address
        destination)
285 {
286   socket->SendTo (packet, 0, InetSocketAddress (destination, 80));
287 }
```

Este método es invocado por el método `RecvPackDiscovery` (que se explica a continuación), y está destinado a retransmitir un paquete Discovery por los nodos intermedios.

Las siguientes líneas implementan el método `RecvPackDiscovery` de la clase `RoutingProtocol`:

```
291 void
292 RoutingProtocol::RecvPackDiscovery (Ptr<Socket> socket)
293 {
294   Address sourceAddress;
295   Ptr<Packet> p = socket->RecvFrom (sourceAddress);
296   InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress)
        ;
297   Ipv4Address sender = inetSourceAddr.GetIpv4 ();
298   Ipv4Address receiver = m_socketAddresses[socket].GetLocal ();
299
300   TypeHeader tHeader (DISCOVERYTYPE_DISCV);
301   p->RemoveHeader (tHeader);
302
303   DiscvHeader discvHeader;
304   p->RemoveHeader (discvHeader);
305   uint16_t id = discvHeader.GetId ();
306   uint32_t hop = discvHeader.GetHopCount () + 1;
```

```
307   discvHeader.SetHopCount (hop);
308   discvHeader.SetSender (sender);
309
310   RoutingTableEntry toDest;
311
312   if (receiver == discvHeader.GetDst ())
313     return;
314
315   // Si no hay una entrada en la tabla para el destino, registrar una entrada para
        él.
316   if (!d_routingTable.LookupRoute (discvHeader.GetDst (), toDest))
317     {
318       Ptr<NetDevice> dev = d_ipv4->GetNetDevice (d_ipv4->GetInterfaceForAddress (
            receiver));
319       RoutingTableEntry newEntry (dev, discvHeader.GetDst (), discvHeader.GetId (),
320                                   d_ipv4->GetAddress (d_ipv4->
                                        GetInterfaceForAddress (receiver), 0),
321                                   discvHeader.GetHopCount (), discvHeader.GetSender
                                        ());
322
323       d_routingTable.AddRoute (newEntry);
324     }
325
326    // Actualizar tabla de ruteo.
327    else if (discvHeader.GetHopCount () <= toDest.GetHop ())
328     {
329       toDest.SetOutputDevice (d_ipv4->GetNetDevice (d_ipv4->GetInterfaceForAddress
            (receiver)));
330       toDest.SetDestination (discvHeader.GetDst ());
331       toDest.SetUltId (discvHeader.GetId ());
332       toDest.SetInterface (d_ipv4->GetAddress (d_ipv4->GetInterfaceForAddress (
            receiver), 0));
333       toDest.SetHop (discvHeader.GetHopCount ());
334       toDest.SetNextHop (discvHeader.GetSender ());
335
336       d_routingTable.Update (toDest);
337     }
338
339   // Verificar si un paquete discovery es duplicado.
340   if (d_packsDiscvIdCache.IsDuplicate (discvHeader.GetDst (), id))
341     {
342       NS_LOG_DEBUG ("Ignoring discovery due to duplicate");
343       return;
344     }
345
346   // Difundir paquetes discovery desde cada interfaz.
347   for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
348          m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j)
349     {
350       Ptr<Socket> socket = j->first;
351       Ipv4InterfaceAddress iface = j->second;
352
```

```
353        Ptr<Packet> packet = Create<Packet> ();
354
355        packet->AddHeader (discvHeader);
356        TypeHeader tHeader (DISCOVERYTYPE_DISCV);
357        packet->AddHeader (tHeader);
358
359        Ipv4Address destination;
360        destination = iface.GetBroadcast ();
361
362        Simulator::Schedule (Seconds (timeInitBroad), &RoutingProtocol::SendTo, this,
             socket,
363                            packet, destination);
364        timeInitBroad = timeInitBroad + d_broadInterval;
365      }
366  }
```

Este método se ejecuta cada vez que un nodo recibe un paquete de datos. Este método recibe un único parámetro, que corresponde a un puntero al socket correspondiente a la interface que recibe. Este método recibe un paquete de tipo Discovery, luego crea, utilizando los datos del paquete recepcionado, un nuevo paquete Discovery y programa su difusión.

La línea `Address sourceAddress;` declara un objeto de la clase `Address`. Esta clase genérica sirve para manipular diferentes tipos de dirección como se explica en `https://www.nsnam.org/doxygen/classns3_1_1_address.html#details`. Con la línea `Ptr<Packet>p = socket->RecvFrom (sourceAddress);` se declara un puntero `p` a un objeto `Packet` al que se asigna el paquete recibido. Además, `sourceAddress` obtendrá la dirección del nodo que generó el paquete (en este caso, el nodo que genera la difusión del paquete Discovery). Luego, la línea `InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress);` define el objeto `inetSourceAddr` de la clase `InetSocketAddress`, a la que se asigna (gracias al método `ConvertFrom`) una dirección IPv4 y un número de puerto (en este caso, el número 80, que se utilizó a lo largo del código). Con la línea `Ipv4Address sender = inetSourceAddr.GetIpv4 ();` se declara el objeto sender (clase `Ipv4Address`) a la que se asigna la dirección IPv4 obtenida desde el objeto `inetSourceAddr` gracias al método `GetIpv4`.

Con `Ipv4Address receiver = m_socketAddresses[socket].GetLocal ();` se declara el objeto `Ipv4Address reveiver` al que asigna la dirección del nodo local, obtenida gracias al método `GetLocal` de la clase `Ipv4InterfaceAddress`. `m_socketAddresses`, corresponde a un mapa (contenedor asociativo) que contiene un puntero a un socket y una interface IPv4 como se explicó en la página 41.

Luego se reliza el proceso de obtención de los campos del encabezado de los paquetes Discovery, análogo a lo realizado en el método `SendDiscovery`. Las líneas:

```
300   TypeHeader tHeader (DISCOVERYTYPE_DISCV);
301   p->RemoveHeader (tHeader);
302
303   DiscvHeader discvHeader;
304   p->RemoveHeader (discvHeader);
305   uint16_t id = discvHeader.GetId ();
306   uint32_t hop = discvHeader.GetHopCount () + 1;
307   discvHeader.SetHopCount (hop);
308   discvHeader.SetSender (sender);
```

definen los objetos necesarios para extraer del paquete (`p`) el tipo de encabezado (`TypeHeader`) y la estructura de campos propiamente tal (`DiscvHeader`). Los campos Identificador del Paquete y Contador de Saltos de texttttdiscv-Header son recuperados en las variables id y hop. Esta última variable es incrementada en uno y reasignada a la estructura de encabezado en `discvHeader`. Esto, con el propósito de utilizar la misma estructura, más adelante, para la difusión del nuevo paquete Discovery. En `discvHeader.SetSender (sender);` se asigna a `discvHeader` la

dirección IP correspondiente al paquete que emite mensaje. EXPLICACIÓN DE discvHeader.SetSender (sender); //A REVISAR.

Más adelante, con la línea RoutingTableEntry toDest; se declara una nueva entrada para la tabla de ruteo. La condición que sigue a esta línea sirve para evitar registros y actualizaciones en la tabla de ruteo y evitar que el destino o un nodo intermedio redifunda un mismo paquete Discovery más de una vez.

La condición en la línea 316 ejecuta una consulta en la tabla de ruteo almacenada en el objeto d_routingTable. d_routingTable es de la clase RoutingTable, clase inspirada en la clase homónima implementada en el ejemplo disponible del protocolo AODV. Esta clase será explicada más adelante en XXXXXX. El método LookupRoute retorna verdadero si encuentra una ruta para el destino indicado en el primer parámetro y, en caso de encontrar un registro, retornará la entrada correspondiente a la respuesta de la consulta, por referencia, a través del segundo parámetro (parámetro de la clase RoutingTableEntry (esta clase también fue creada inspirada en el ejemplo de AODV).

La condición de la línea 316 retorna positivo si no se encuentra un registro hacia el destino indicado por discvHeader.GetDst (). Si esto es así (no se encuentra una entrada en la tabla de ruteo), entonces crea una nueva entrada. La línea Ptr<NetDevice>dev = d_ipv4->GetNetDevice (d_ipv4->GetInterfaceForAddress (receiver)); declara un puntero dev al dispositivo de red correspondiente al receptor (la explicación de los métodos involucrados ya se explicó más arriba en este documento). Luego, con la línea RoutingTableEntry newEntry (dev, discvHeader.GetDst (), discvHeader.GetId (), d_ipv4->GetAddress (d_ipv4->GetInterfaceForAddress (receiver), 0), discvHeader.GetHopCount (), discvHeader.GetSender ()); se declara un objeto de la clase RoutingTableEntry al que se asocian los distintos datos necesarios para el registro de la nueva entrada en la tabla de ruteo, para luego ser agregado efectivamente con el método AddRoute del objeto d_routingTable (línea d_routingTable.AddRoute (newEntry);).

En caso de resultar negativo el resultado de la condición (es decir, se encuentra una ruta hacia el destino), se consulta si el contador de saltos indicado en el paquete entrante es menor o igual al registrado en la tabla de ruteo (este último retornado por LookupRoute hacia el objeto toDest). Esto se hace con el objetivo de realizar una actualización de la tabla de ruteo, sólo en los casos en que un nuevo paquete represente una mejor alternativa a uno ya registrado, como es común en los protocolos que buscan minimizar el costo (ejemplo, aquellos basados en vectores de distancia como AODV).

En caso de recibir una mejor alternativa por parte del nodo vecino (aquel que envía el paquete Discovery), se actualizarán los valores de los campos del objeto toDest con los valores indicados por el paquete entrante, para luego actualizar el registro correspondiente utilizando el método Update de la clase RoutingTable (línea d_routingTable.Update (toDest);.

En la línea if (d_packsDiscvIdCache.IsDuplicate (discvHeader.GetDst (), id)) se verifica que el paquete no corresponda a un duplicado, utilizando como referencia su indicador. Si se trata de un duplicado el método simplemente retornará (llamada return).

El ciclo for siguiente tienen objetivo considerar la existencia de más de una conexión (socket + interfaz IPv4), sin embargo, en este ejemplo, el código asociado sólo se realiza una vez. Las líneas dentro del for están destinadas a la creación de un paquete de datos Discover, análogo a lo explicado en caso del método SendDiscovery, salvo que su transmisión no es inmediata, sino que es programada con ayuda del método Simulator::Schedule, timeInitBroad segundos después, análogo a lo explicado en el caso del método CreateEvBroadDst. Luego timeInitBroad es incrementado para el siguiente evento de retransmisión.

Finalmente, el método PrintRoutingTable (línea 366) es un método virtual de la clase Ipv4RoutingProtocol, que en este caso no tiene implementación.

Finalmente, el código del archivo con las declaraciones discovery-routing-protocol.h es el siguiente

Listado 4: Contenido del archivo discovery-routing-protocol.h

```
1 #ifndef DISCOVERYROUTINGPROTOCOLH
2 #define DISCOVERYROUTINGPROTOCOLH
3
4 #include "discovery-id-cache.h"
5 #include "discovery-rtable.h"
6 #include "discovery-packet.h"
```

```
7  #include "ns3/node.h"
8  #include "ns3/ipv4-routing-protocol.h"
9  #include "ns3/ipv4-l3-protocol.h"
10 #include <map>
11
12 namespace ns3 {
13 namespace discovery
14 {
15
16 class RoutingProtocol : public Ipv4RoutingProtocol
17 {
18 public:
19   static TypeId GetTypeId (void);
20   RoutingProtocol ();
21   virtual ~RoutingProtocol ();
22
23   Ptr<Ipv4Route> RouteOutput (Ptr<Packet> p, const Ipv4Header &header, Ptr<
       NetDevice> oif,
24                               Socket::SocketErrno &sockerr);
25   bool RouteInput (Ptr<const Packet> p, const Ipv4Header &header, Ptr<const
       NetDevice> idev,
26                    UnicastForwardCallback ucb, MulticastForwardCallback mcb,
27                    LocalDeliverCallback lcb, ErrorCallback ecb);
28   virtual void SetIpv4 (Ptr<Ipv4> ipv4);
29   virtual void NotifyInterfaceUp (uint32_t interface);
30   virtual void NotifyInterfaceDown (uint32_t interface);
31   virtual void NotifyAddAddress (uint32_t interface, Ipv4InterfaceAddress address);
32   virtual void NotifyRemoveAddress (uint32_t interface, Ipv4InterfaceAddress
       address);
33   virtual void PrintRoutingTable (Ptr<OutputStreamWrapper> stream) const;
34   bool Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
       UnicastForwardCallback ucb,
35                    ErrorCallback ecb);
36   void SetIndexDest (uint32_t indDst);
37   void CreateEvBroadDst (double tStopApp, double intervBroadDest);
38   void SendDiscovery ();
39   void RecvPackDiscovery (Ptr<Socket> socket);
40   void SendTo (Ptr<Socket> socket, Ptr<Packet> packet, Ipv4Address destination);
41
42 private:
43   uint32_t indexDstApp;
44   Ptr<Ipv4> d_ipv4; // Protocolo IP. Access to the Ipv4 forwarding table,
       interfaces, and configuration. This class defines the API to
45                     // manipulate the following aspects of the Ipv4 implementation.
46   std::map< Ptr<Socket>, Ipv4InterfaceAddress > m_socketAddresses;
47   IdCache d_packsDiscvIdCache; // Este objeto permite detectar si un paquete
       discovery es duplicado.
48   RoutingTable d_routingTable;
49   Time d_timeStartDiscv; // Tiempo en que se inicia cada evento de difusión de un
       paquete discovery desde el destino.
50   double d_broadInterval; // Intervalo de tiempo para la redifusión de un paquete
       discovery desde los nodos intermedios.
```

```
51 };
52 }
53 }
54 #endif
```

5.1.2. discovery-id-cache.cc y .h

El archivo discovery-id-cache.cc contiene los métodos que permiten verificar si un paquete Discovery es un duplicado, evitando su retransmisión. El código de este archivo, que se explicará más adelante, es el siguiente:

Listado 5: Contenido del archivo discovery-id-cache.cc

```
1
2 #include "discovery-id-cache.h"
3 #include <algorithm>
4
5 namespace ns3
6 {
7 namespace discovery
8 {
9 bool
10 IdCache::IsDuplicate (Ipv4Address addr, uint16_t id)
11 {
12   Purge ();
13   for (std::vector<UniqueId>::const_iterator i = m_idCache.begin ();
14        i != m_idCache.end (); ++i)
15     if (i->m_context == addr && i->m_id == id)
16       return true;
17
18   struct UniqueId uniqueId =
19   { addr, id, m_lifetime + Simulator::Now () };
20   m_idCache.push_back (uniqueId);
21
22   return false;
23 }
24
25 void
26 IdCache::Purge ()
27 {
28   m_idCache.erase (remove_if (m_idCache.begin (), m_idCache.end (),
29                               IsExpired ()), m_idCache.end ());
30 }
31
32 void
33 IdCache::RemoveAll (void)
34 {
35   m_idCache.clear ();
36 }
37
38 uint32_t
39 IdCache::GetSize ()
40 {
```

```
41    Purge ();
42    return m_idCache.size ();
43 }
44
45 }
46 }
```

Este código implementa los métodos de la clase `IdCache` (clase a la que pertenece el objeto `d_packsDiscvIdCache`, en la implementación de la clase `RoutingProtocol`), similar a lo que está implementado en el código de AODV provisto como ejemplo en ns-3. El archivo implementa 4 métodos:

El método `IdDuplicate` retorna un booleano indicando si una tupla {dirección IP, Identificador del paquete} se encuentra registrada en un vector. La dirección IP corresponde a la dirección del nodo del cuál se recibe directamente un paquete, y el Identificador del paquete al identificador registrado en su encabezado. El método retorna verdadero si se encuentra (es un duplicado) y falso en caso contrario. Cuando esto último ocurre, el método agregará la tupla al vector.

Al comienzo del método se llama al método `Purge` que remueve todas las entradas del vector `m_idCache` que han expirado (esto es, que han sobrepasado un cierto tiempo de vida, como se indica en el archivo .h del listado 8). El ciclo `for` recorre el vector `m_idCache` verificando que los campos asociados al iterador i, `m_context` y `m_id`, correspondan a la dirección e identificador en cuestión, respectivamente. En caso de ser así, el método retornará verdadero. En caso contrario (si se terminar de recorrer el vector y no se ha encontrado la tupla, se declara la estructura uniqueId, con la dirección de quién se recibió, el identificador y el tiempo (de simulación) en el que debería expirar el paquete. Esta estructura es registrada efectivamente en el vector `m_idCache` mediante el método `push_back`. Luego, el método retorna Falso, indicando que la tupla no estaba registrada.

Los métodos siguientes, Purge, RemoveAll, y GetSize, hacen llamadas a métodos estándares incluidos en la clase Vector, que implementa el objeto `m_idCache` y su comprensión es trivial. El método `remove_if` pertenece al estándar (`std`) de C++ y transforma un rango de valores, en este caso el comienzo y el fin del vector. y retorna un nuevo rango de valores, en este caso, el rango de indices con registros a eliminar, todo esto según una condición determinada (en este caso `IsExpired()`).

Finalmente, el código del archivo con las declaraciones `discovery-id-cache.h` es el siguiente:

Listado 6: Contenido del archivo `discovery-id-cache.h`

```
1
2  #ifndef DISCOVERY_ID_CACHE_H
3  #define DISCOVERY_ID_CACHE_H
4
5  #include "ns3/ipv4-address.h"
6  #include "ns3/simulator.h"
7
8  namespace ns3
9  {
10 namespace discovery
11 {
12
13 class IdCache
14 {
15 public:
16   IdCache (Time lifetime) : m_lifetime (lifetime) {}
17   /// Check that entry (addr, id) exists in cache. Add entry, if it doesn't exist.
18   bool IsDuplicate (Ipv4Address addr, uint16_t id);
19   /// Remove all expired entries
20   void Purge ();
21   /// Return number of entries in cache
```

```
22    uint32_t GetSize ();
23    void RemoveAll (void);
24    /// Set lifetime for future added entries.
25    void SetLifetime (Time lifetime) { m_lifetime = lifetime; }
26    /// Return lifetime for existing entries in cache
27    Time GetLifeTime () const { return m_lifetime; }
28
29  private:
30    /// Unique packet ID
31    struct UniqueId
32    {
33      /// ID is supposed to be unique in single address context (e.g. sender address)
34      Ipv4Address m_context;
35      /// The id
36      uint16_t m_id;
37      /// When record will expire
38      Time m_expire;
39    };
40
41    struct IsExpired
42    {
43      bool operator() (const struct UniqueId & u) const
44      {
45        return (u.m_expire < Simulator::Now ());
46      }
47    };
48
49    std::vector<UniqueId> m_idCache;
50    /// Default lifetime for ID records
51    Time m_lifetime;
52  };
53
54  }
55  }
56  #endif /* DISCOVERY_ID_CACHE_H */
```

5.1.3. `discovery-rtable.cc` y `.h`

El archivo `discovery-rtable.cc` contiene los métodos de las clases `RoutingTableEntry` y `RoutingTable` declaradas en el archivo `discovery-rtable.cc`. Estas clases permiten implementar, respectivamente, una entrada en la tabla de ruteo, y la tabla de ruteo propiamente tal. El código de este archivo, que se explicará más adelante, es el siguiente:

Listado 7: Contenido del archivo `discovery-rtable.cc`

```
1
2  #include "discovery-rtable.h"
3  #include "ns3/log.h"
4
5  NS_LOG_COMPONENT_DEFINE ("DiscoveryRoutingTable");
6
7  namespace ns3
```

```
8  {
9  namespace discovery
10 {
11 
12 RoutingTableEntry::RoutingTableEntry (Ptr<NetDevice> dev, Ipv4Address dst, uint32_t
       uid,
13                                       Ipv4InterfaceAddress iface, uint16_t hops,
                                          Ipv4Address nextHop) :
14   ult_id (uid), m_hops (hops), m_iface (iface)
15 {
16   m_ipv4Route = Create<Ipv4Route> ();
17   m_ipv4Route->SetDestination (dst);
18   m_ipv4Route->SetGateway (nextHop);
19   m_ipv4Route->SetSource (m_iface.GetLocal ());
20   m_ipv4Route->SetOutputDevice (dev);
21 }
22 
23 RoutingTableEntry::~RoutingTableEntry ()
24 {
25 }
26 
27 RoutingTable::RoutingTable (Time t):
28   m_badLinkLifetime (t)
29 {
30 }
31 
32 bool
33 RoutingTable::LookupRoute (Ipv4Address id, RoutingTableEntry & rt)
34 {
35   NS_LOG_FUNCTION (this << id);
36   if (m_ipv4AddressEntry.empty ())
37     {
38       NS_LOG_LOGIC ("Route to " << id << " not found");
39       return false;
40     }
41 
42   std::map<Ipv4Address, RoutingTableEntry>::const_iterator i =
43   m_ipv4AddressEntry.find (id);
44 
45   if (i == m_ipv4AddressEntry.end ())
46     {
47       NS_LOG_LOGIC ("Route to " << id << " not found");
48       return false;
49     }
50 
51   rt = i->second;
52   NS_LOG_LOGIC ("Route to " << id << " found");
53   return true;
54 }
55 
56 bool
57 RoutingTable::AddRoute (RoutingTableEntry & rt)
```

```
58  {
59    NS_LOG_FUNCTION (this);
60
61    std::pair<std::map<Ipv4Address, RoutingTableEntry>::iterator, bool> result =
62              m_ipv4AddressEntry.insert (std::make_pair (rt.GetDestination (), rt));
63
64    return result.second;
65  }
66
67  bool
68  RoutingTable::Update (RoutingTableEntry & rt)
69  {
70    NS_LOG_FUNCTION (this);
71
72    std::map<Ipv4Address, RoutingTableEntry>::iterator i =
73      m_ipv4AddressEntry.find (rt.GetDestination ());
74    if (i == m_ipv4AddressEntry.end ())
75      {
76        NS_LOG_LOGIC ("Route update to " << rt.GetDestination () << " fails; not
            found");
77        return false;
78      }
79    i->second = rt;
80
81    return true;
82  }
83
84
85  void
86  RoutingTable::Purge ()
87  {
88    NS_LOG_FUNCTION (this);
89    if (m_ipv4AddressEntry.empty ())
90      {
91        return;
92      }
93
94    for (std::map<Ipv4Address, RoutingTableEntry>::iterator i =
95            m_ipv4AddressEntry.begin (); i != m_ipv4AddressEntry.end ();)
96      {
97         std::map<Ipv4Address, RoutingTableEntry>::iterator tmp = i;
98        ++i;
99         m_ipv4AddressEntry.erase (tmp);
100
101     }
102 }
103
104 }
105 }
```

Este código está implmenetado de forma análoga a lo implementado en el código de AODV provisto como ejemplo en ns-3.

El método `RoutingTableEntry` es el método constructor de la clase homónima. Esta crea un objeto de la clase `Ipv4Route` (apuntado por el puntero `m_ipv4Route`, definido como atributo de la clase `RoutingTableEntry` en el archivo .h) y configura sus campos utilizando los parámetros de entrada del método. Los métodos `SetDestination`, `SetGateway`, `SetSource` y `SetOutputDevice`, pertenecen a la clase `Ipv4Route` de ns-3, y están destinados a indicar, respectivamente, la dirección IP del nodo destino, la dirección IP del siguiente salto, la dirección IP del nodo que originó el paquete, y el dispositivo de salida del nodo que está manejando la tabla de ruteo actual.

Luego de la implementación (sin código) del destructor de `RoutingTableEntry`, se implementan los métodos de la clase `RoutingTable`.

El constructor de la clase `RoutingTable`, sólo inicializa el objeto `m_badLinkLifetime` que está definido en el archivo .h como un atributo utilizado para definir el tiempo que transcurrirá antes de eliminar una ruta inválida en la tabla de ruteo.

El método `LookupRoute` verifica la existencia de una entrada en la tabla de ruteo para la dirección de destino indicada en el primer parámetro `Ipv4Address id`.

En la condición en la línea `if (m_ipv4AddressEntry.empty ())` se hace referencia a un objeto `m_ipv4AddressEntry`. Este objeto es un atributo de la clase `RoutingTable`, declarado como un mapa (objeto clase `map`) que contiene entradas asociando una dirección IP a una entrada en la tabla de ruteo (clase `RoutingTableEntry`). Es, entonces, en este objeto donde se almacenan todos los registros de la tabla de ruteo. La condición `if` consulta si el mapa está vacío, es decir, si no hay entradas registradas en la tabla de ruteo, en cuyo caso se registra un mensaje para y se retorna falso (es decir, no se encontró una ruta).

En caso de existir registros, se busca en el mapa un registro que contenga la dirección indicada por el parámetro `id`, la que se retorna a un iterador `i`. Si este iterador es igual a el elemento pasado el último registro encontrado (método `end` del objeto `map`), significa que se es encuentra ningún elemento por lo que se procede de forma similar a la condición anterior (registra mensaje y retorna Falso).

Si se encuentra un elemento, se asocia el segundo elemento del iterador i (es decir, el objeto `RoutingTableEntry`) al parámetro por referencia `rt` de la misma clase, se registra el evento, y se retorna Verdadero.

En el método AddRoute, la línea `std::pair<std::map<Ipv4Address, RoutingTableEntry>::iterator, bool>result = m_ipv4AddressEntry.insert (std::make_pair (rt.GetDestination (), rt));` permite insertar en la tabla de ruteo, implementada en `m_ipv4AddressEntry`, un registro compuesto de una dirección IP de destino y un objeto `RoutingTableEntry` con la entrada a asociar en la tabla. Detallando más esta línea, el método insert del objeto `m_ipv4AddressEntry` (clase `map`) recibe como parámetro el par `(rt.GetDestination (), rt)`, asociado a un objeto `pair` mediante el método `make_pair`. El método `insert` retorna un objeto `pair` compuesto de:

1. un iterador apuntando, ya sea al nuevo elemento insertado (a un nuevo par `(rt.GetDestination (), rt)`), o a un elemento con una clave equivalente en el mapa (`m_ipv4AddressEntry`). y

2. un booleano, indicando si se insertó o no un nuevo elemento.

Este resultado es retornado al `pair` result. En la línea siguiente se retorna el booleano.

El método `Update` se ejecuta cuando se requiere actualizar una entrada de la tabla de ruteo. La línea `std::map<Ipv4Address, RoutingTableEntry>::iterator i = m_ipv4AddressEntry.find (rt.GetDestination ());` permite buscar en el mapa `m_ipv4AddressEntry` una entrada que contenga como clave la dirección IP del destino, indicada por `rt.GetDestination ()`. El método find retornará un iterador al elemento encontrado (si existe). En caso de no encontrarse (validación en la condición de la línea `if (i == m_ipv4AddressEntry.end ())`) se registra el evento y se retorna Falso indicando que no se realizaron actualizaciones. En caso de encontrarse la entrada correspondiente, se actualiza con el objeto referido por el parámetro `rt` y se retorna Verdadero.

Finalmente, el método `Purge` se implementa para eliminar todos los elementos del mapa `m_ipv4AddressEntry`. En la implementación explicada en este documento, en realidad, este método no se utiliza, pero se dejó disponible como lo está en la implementación de AODV provista por ns-3.

Finalmente, el código del archivo con las declaraciones `discovery-rtable.h` es el siguiente:

Listado 8: Contenido del archivo `discovery-rtable.h`

```
1
2 #ifndef DISCOVERY_RTABLE_H
```

```
3  #define DISCOVERY_RTABLE_H
4
5  #include "ns3/ipv4.h"
6  #include "ns3/timer.h"
7
8  namespace ns3 {
9  namespace discovery {
10
11 class RoutingTableEntry
12 {
13 public:
14   RoutingTableEntry (Ptr<NetDevice> dev = 0, Ipv4Address dst = Ipv4Address (),
        uint32_t ult_id = 0,
15                      Ipv4InterfaceAddress iface = Ipv4InterfaceAddress (), uint16_t
                             hops = 10,
16                      Ipv4Address nextHop = Ipv4Address ());
17
18   ~RoutingTableEntry ();
19   void SetDestination (Ipv4Address addr) { m_ipv4Route->SetDestination (addr); }
20   Ipv4Address GetDestination () const { return m_ipv4Route->GetDestination (); }
21   Ptr<Ipv4Route> GetRoute () const { return m_ipv4Route; }
22   void SetRoute (Ptr<Ipv4Route> r) { m_ipv4Route = r; }
23   void SetNextHop (Ipv4Address nextHop) { m_ipv4Route->SetGateway (nextHop); }
24   Ipv4Address GetNextHop () const { return m_ipv4Route->GetGateway (); }
25   void SetOutputDevice (Ptr<NetDevice> dev) { m_ipv4Route->SetOutputDevice (dev); }
26   Ptr<NetDevice> GetOutputDevice () const { return m_ipv4Route->GetOutputDevice ();
        }
27   Ipv4InterfaceAddress GetInterface () const { return m_iface; }
28   void SetInterface (Ipv4InterfaceAddress iface) { m_iface = iface; }
29   void SetUltId (uint32_t sid) { ult_id = sid; }
30   uint32_t GetUltId () const { return ult_id; }
31   void SetHop (uint16_t hop) { m_hops = hop; }
32   uint16_t GetHop () const { return m_hops; }
33
34   /*bool operator== (Ipv4Address const  dst) const
35   {
36     return (m_ipv4Route->GetDestination () == dst);
37   }*/
38
39 private:
40   uint32_t ult_id;
41   uint16_t m_hops;
42   Ptr<Ipv4Route> m_ipv4Route;
43   /// Output interface address
44   Ipv4InterfaceAddress m_iface;
45 };
46
47 class RoutingTable
48 {
49 public:
50   RoutingTable (Time t);
51   bool AddRoute (RoutingTableEntry & r);
```

```
52   bool LookupRoute (Ipv4Address dst, RoutingTableEntry & rt);
53   bool Update (RoutingTableEntry & rt);
54   void Purge ();
55   std::map<Ipv4Address, RoutingTableEntry> m_ipv4AddressEntry;
56
57 private:
58   /// Deletion time for invalid routes
59   Time m_badLinkLifetime;
60 };
61
62 }
63 }
64
65 #endif /* DISCOVERY_RTABLE_H */
```

5.1.4. `discovery-packet.cc` y `.h`

El archivo `discovery-packet.cc` contiene la implementación de los métodos de las clases `TypeHeader` y `DiscvHeader` declaradas en el archivo `discovery-packet.h`. Ambas clases heredan de la clase `Header` de ns3. Estas clases permiten implementar la estructura del encabezado de los paquetes Discovery, junto con las funciones que permiten su utilización. El código de este archivo, que se explicará más adelante, es el siguiente:

Listado 9: Contenido del archivo `discovery-packet.cc`

```
1
2  #include "discovery-packet.h"
3  #include "ns3/address-utils.h"
4  #include "ns3/packet.h"
5
6  namespace ns3
7  {
8  namespace discovery
9  {
10
11 NS_OBJECT_ENSURE_REGISTERED (TypeHeader);
12
13 TypeHeader::TypeHeader (MessageType t) :
14   m_type (t)
15 {
16 }
17
18 TypeId
19 TypeHeader::GetTypeId ()
20 {
21   static TypeId tid = TypeId ("ns3::discovery::TypeHeader")
22     .SetParent<Header> ()
23     .AddConstructor<TypeHeader> ()
24   ;
25   return tid;
26 }
27
28 TypeId
```

```
29  TypeHeader::GetInstanceTypeId () const
30  {
31    return GetTypeId ();
32  }
33
34  uint32_t
35  TypeHeader::GetSerializedSize () const
36  {
37    return 1;
38  }
39
40  void
41  TypeHeader::Serialize (Buffer::Iterator i) const
42  {
43    i.WriteU8 ((uint8_t) m_type);
44  }
45
46  uint32_t
47  TypeHeader::Deserialize (Buffer::Iterator start)
48  {
49    Buffer::Iterator i = start;
50    uint8_t type = i.ReadU8 ();
51    //m_valid = true;
52    switch (type)
53      {
54      case DISCOVERYTYPE_DISCV:
55        {
56          m_type = (MessageType) type;
57          break;
58        }
59      //default:
60        //m_valid = false;
61      }
62    uint32_t dist = i.GetDistanceFrom (start);
63    NS_ASSERT (dist == GetSerializedSize ());
64    return dist;
65  }
66
67  void
68  TypeHeader::Print (std::ostream &os) const
69  {
70    switch (m_type)
71      {
72      case DISCOVERYTYPE_DISCV:
73        {
74          os << "DISCV";
75          break;
76        }
77      default:
78        os << "UNKNOWN_TYPE";
79      }
80  }
```

```
81
82  /*bool
83  TypeHeader::operator== (TypeHeader const & o) const
84  {
85    return (m_type == o.m_type && m_valid == o.m_valid);
86  }
87
88  std::ostream &
89  operator<< (std::ostream & os, TypeHeader const & h)
90  {
91    h.Print (os);
92    return os;
93  }*/
94
95  DiscvHeader::DiscvHeader (uint32_t hopCount, uint32_t discoveryID, Ipv4Address dest
       ,
96                              Ipv4Address sender) :
97    m_hopCount (hopCount), m_packetID (discoveryID), m_dst (dest),
98    m_sender (sender)
99  {
100 }
101
102 NS_OBJECT_ENSURE_REGISTERED (DiscvHeader);
103
104 TypeId
105 DiscvHeader::GetTypeId ()
106 {
107   static TypeId tid = TypeId ("ns3::discovery::DiscvHeader")
108     .SetParent<Header> ()
109     .AddConstructor<DiscvHeader> ()
110   ;
111   return tid;
112 }
113
114 TypeId
115 DiscvHeader::GetInstanceTypeId () const
116 {
117   return GetTypeId ();
118 }
119
120 uint32_t
121 DiscvHeader::GetSerializedSize () const
122 {
123   return 23;
124 }
125
126 void
127 DiscvHeader::Serialize (Buffer::Iterator i) const
128 {
129   i.WriteHtonU32 (m_hopCount);
130   i.WriteHtonU32 (m_packetID);
131   WriteTo (i, m_dst);
```

```
132   WriteTo (i, m_sender);
133 }
134
135 uint32_t
136 DiscvHeader::Deserialize (Buffer::Iterator start)
137 {
138   Buffer::Iterator i = start;
139   m_hopCount = i.ReadNtohU32 ();
140   m_packetID = i.ReadNtohU32 ();
141   ReadFrom (i, m_dst);
142   ReadFrom (i, m_sender);
143
144   uint32_t dist = i.GetDistanceFrom (start);
145
146   return dist;
147 }
148
149 void
150 DiscvHeader::Print (std::ostream &os) const
151 {
152   os << "Packet discovery Id " << m_packetID << " destination: ipv4 " << m_dst;
153 }
154
155 std::ostream &
156 operator<< (std::ostream & os, DiscvHeader const & h)
157 {
158   h.Print (os);
159   return os;
160 }
161
162 bool
163 DiscvHeader::operator== (DiscvHeader const & o) const
164 {
165   return (m_hopCount == o.m_hopCount && m_packetID == o.m_packetID &&
166           m_dst == o.m_dst && m_sender == o.m_sender);
167 }
168
169 }
170 }
```

El constructor de la clase `TypeHeader`, en la línea 13, simplemente inicializa el tipo de encabezado del paquete Discovery. Esto es similar a lo que está implementado en Aodv. Esto cobra mayor relevancia cuando se manejan diferentes tipos de paquete, en cuyo caso se diferencian por un tipo numerado.

El método `GetTypeId` retorna el `TypeId` de esta subclase, la cual puede poseer la siguiente información referente a la subclase `TypeHeader`: (1) La clase base de la subclase (a través del método `SetParent`), (2) el conjunto de constructores accesible en la subclase (a través del método `AddConstructor`), y (3) el conjunto de atributos accesibles en la subclase. El método `GetInstanceTypeId` retornará el `TypeId` más derivado del objeto implementado (de la clase `TypeHeader`).

El método `GetSerializedSize` es un método virtual puro heredado de `Header` y utilizado por la clase Packet. Retorna el tamaño esperado (en bytes) del encabezado.

El método `Serialize` es un método virtual puro utilizado por AddHeader de la clase Packet para guardar un encabezado en el buffer de conformación de un paquete, comenzando desde el byte indicado por su único parámetro

de entrada. En su implementación, el método `WriteU8` escribe datos en el buffer indicando, incrementando la posición del objeto iterador (en este caso `i`) en 1 byte.

El método `Deserialize` es un método virtual puro utilizado por la clase Packet en el método RemoveHeader. En esta clase, este método es utilizado para recuperar el TypeHeader desde el buffer de bytes. Para esto, se inicializa un iterador en la posición de incio del buffer referida por el parámetro start. Luego, el identificador del tipo de paquete es leído gracias al método ReadU8 y asignado a una variable numérica de 8 bits sin signo. La condición múltiple de casos switch/case está implementada para el caso de múltiples tipos de paquete. Aquí, sólo se utiliza un tipo de paquete (paquete Discovery). El tipo de paquete es retornado al atributo m_type. Las últimas líneas están diseñadas para validar la cantidad de bytes leídos por este concepto (en este caso el tamaño del TypeHeader es de 1). Este método retorna la cantidad de bytes leídas desde el buffer que almacena el identificador del tipo de paquete.

`Print` es un método virtual puro utilizado por el método Print de la clase Packet para imprimir el contenido de una cadena de datos en formato Ascii a través del stream de salida indicado por el parámetro de entrada `os`.

Comprendiendo esto, la comprensión de la implementación de los métodos de la clase DiscvHeader es trivial. Como la primera, esta clase hereda también de la clase Header. La diferencia más notoria es la presencia de varios campos que conforman el encabezado.

Finalmente, el código del archivo con las declaraciones `discovery-packet.h` es el siguiente:

Listado 10: Contenido del archivo `discovery-packet.h`

```
1
2  #ifndef DISCOVERYPACKET_H
3  #define DISCOVERYPACKET_H
4
5  #include "ns3/header.h"
6  #include "ns3/ipv4-address.h"
7
8  namespace ns3 {
9  namespace discovery {
10
11 enum MessageType
12 {
13   DISCOVERYTYPE_DISCV  = 1,
14 };
15
16 class TypeHeader : public Header
17 {
18 public:
19
20   TypeHeader (MessageType t = DISCOVERYTYPE_DISCV);
21   static TypeId GetTypeId ();
22   TypeId GetInstanceTypeId () const;
23   uint32_t GetSerializedSize () const;
24   void Serialize (Buffer::Iterator start) const;
25   uint32_t Deserialize (Buffer::Iterator start);
26   void Print (std::ostream &os) const;
27
28   // Return type
29   MessageType Get () const { return m_type; }
30   // Check that type if valid
31   //bool IsValid () const { return m_valid; }
32   //bool operator== (TypeHeader const & o) const;
33
```

```
34 private:
35   MessageType m_type;
36   //bool m_valid;
37 };
38
39 //std::ostream & operator<< (std::ostream & os, TypeHeader const & h);
40
41 class DiscvHeader : public Header
42 {
43 public:
44   DiscvHeader (uint32_t hopCount = 0, uint32_t discoveryID = 0, Ipv4Address dest =
       Ipv4Address (),
45                Ipv4Address sender = Ipv4Address ());
46
47   // name Header serialization/deserialization
48   static TypeId GetTypeId ();
49   TypeId GetInstanceTypeId () const;
50   uint32_t GetSerializedSize () const;
51   void Serialize (Buffer::Iterator start) const;
52   uint32_t Deserialize (Buffer::Iterator start);
53   void Print (std::ostream &os) const;
54
55   // name Fields
56   void SetHopCount (uint32_t count) { m_hopCount = count; }
57   uint32_t GetHopCount () const { return m_hopCount; }
58   void SetId (uint32_t id) { m_packetID = id; }
59   uint32_t GetId () const { return m_packetID; }
60   void SetDst (Ipv4Address a) { m_dst = a; }
61   Ipv4Address GetDst () const { return m_dst; }
62   void SetSender (Ipv4Address a) { m_sender = a; }
63   Ipv4Address GetSender () const { return m_sender; }
64
65   bool operator== (DiscvHeader const & o) const;
66
67 private:
68   uint32_t        m_hopCount;
69   uint32_t        m_packetID;
70   Ipv4Address     m_dst;
71   Ipv4Address     m_sender;
72 };
73
74 std::ostream & operator<< (std::ostream & os, DiscvHeader const &);
75 }
76 }
77 #endif /* DISCOVERYPACKET_H */
```

5.1.5. discovery.cc

En la carpeta `examples` se encuentra el archivo de ejemplo con la utilización de este protocolo. Este archivo se llama, en nuestro caso de ejemplo, `discovery.cc`. Este mismo archivo es el que, más adelante, se debe copiar en la carpeta scratch para su ejecución. El código del archivo `discovery.cc` se presenta comentado a continuación:

Listado 11: Contenido del archivo `discovery.cc`

```
1  #include "ns3/discovery-module.h"
2  #include "ns3/core-module.h"
3  #include "ns3/network-module.h"
4  #include "ns3/mobility-module.h"
5  #include "ns3/config-store-module.h"
6  #include "ns3/wifi-module.h"
7  #include "ns3/netanim-module.h"
8  #include "ns3/internet-module.h"
9  #include "ns3/point-to-point-module.h"
10 #include "ns3/applications-module.h"
11 #include "ns3/discovery-helper.h"
12
13 #include <iostream>
14 #include <fstream>
15 #include <vector>
16 #include <string>
17
18 using namespace ns3;
19
20 int main (int argc, char *argv[])
21 {
22   double totalTime = 15.5;      // Se define el tiempo total de simulación (en
        segundos)
23   double interval = 1.0;        // Es el intérvalo de tiempo entre dos
        transmisiones de paquetes de aplicación (en segundos).
24
25   //Estas líneas son para activar la visualización de los mensajes log de las
        aplicaciones cliente y servidor.
26   LogComponentEnable ("UdpClient", LOG_LEVEL_INFO);
27   LogComponentEnable ("UdpServer", LOG_LEVEL_INFO);
28
29   //std::string animFile = "discovery-animation.xml" ;
30   std::string phyMode ("DsssRate1Mbps");        // Se definen los parámetros de la
        capa física. En este caso, método DSSS, y bitrate a 1Mbps.
31
32   WifiHelper wifi;      // Se declara el objeto wifi de la clase WifiHelper.
33   YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();   // Se declara un
        objeto Helper para la utilización del modelo Yans de capa física.
34   YansWifiChannelHelper wifiChannel;                            // Se declara un
        objeto Helper para la utilización del modelo de canal Yans.
35   wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");  //
        Se define el modelo de velocidad de propagación de señales. En este caso, se
        utilizará el modelo ConstantSpeedPropagationDelayModel.
36   wifiChannel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel");      //
        Se define el modelo de pérdida por propagación. En este caso, el modelo
        LogDistancePropagationLossModel.
37   wifiPhy.SetChannel (wifiChannel.Create ());                                   //
        Se crea un canal de comunicación utilizando los parámetros antes descritos y
        se asignan para ser utilizados por el objeto de la clase WifiHelper.
38
```

```
39    NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();            // Se
          declara un objeto helper para la definición de la capa Mac a utilizar, en este
          caso, de la clase NqosWifiMacHelper.
40    wifi.SetStandard (WIFI_PHY_STANDARD_80211b);                           // Se
          declara el estándar de comunicación a utilizar en la capa física. En este caso
          Wifi, implementando la norma IEEE 802.11b.
41    wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",          //
          SetRemoteStationManager sirve para definir varios parámetros del objeto de
          clase WifiHelper. "ns3::ConstantRateWifiManager" indica que se utilizarán
          tasas de transmisión constantes para los datos y control de transmisión.
42                                  "DataMode",StringValue (phyMode),        // "
                                        DataMode" es el modo de transmisión utilizado
                                        para transmitir cada paquete de datos (por
                                        defecto es OFDM). El valor es obtenido desde el
                                        string phyMode definido más arriba, con ayuda de
                                        StringValue.
43                                  "ControlMode",StringValue (phyMode));    // "
                                        ControlMode" es el modo de transmisión a
                                        utilizar para cada transmisión de paquete RTS.
44    wifiMac.SetType ("ns3::AdhocWifiMac");                                 // Se
          define que se utilizará un modelo AdhocWifiMac para la implementación de la
          capa MAC (red Ad Hoc WiFi).
45
46    NodeContainer c;        // Se define el contenedor de nodos c
47    c.Create(6);            // Se crean 6 nodos en el contenedor
48
49    NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);      // Se crean
          dispositivos de red implementando las capas física y MAC definidas antes en
          cada nodo contenido en c. El contenedor de los dispositivos de red es
          retornado en el NetDeviceContainer devices.
50
51    MobilityHelper mobility;                                               // Se crea
          un objeto Helper para el modelo de movilidad.
52    mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");     // Se
          define que el modelo es un modelo de posición fija (posición a definir
          manualmente).
53    mobility.Install (c);                                                  // Se
          instala el modelo de movilidad en cada nodo.
54
55    // Las siguientes líneas asocian punteros a las posiciones de cada nodo y asignan
          valores a estas (con ayuda del método SetPosition)...
56
57    Ptr<ConstantPositionMobilityModel> mm0 = c.Get (0)->GetObject<
          ConstantPositionMobilityModel> ();
58    Ptr<ConstantPositionMobilityModel> mm1 = c.Get (1)->GetObject<
          ConstantPositionMobilityModel> ();
59    Ptr<ConstantPositionMobilityModel> mm2 = c.Get (2)->GetObject<
          ConstantPositionMobilityModel> ();
60    Ptr<ConstantPositionMobilityModel> mm3 = c.Get (3)->GetObject<
          ConstantPositionMobilityModel> ();
61    Ptr<ConstantPositionMobilityModel> mm4 = c.Get (4)->GetObject<
          ConstantPositionMobilityModel> ();
```

```
62    Ptr<ConstantPositionMobilityModel> mm5 = c.Get (5)->GetObject<
          ConstantPositionMobilityModel> ();
63
64    mm0->SetPosition (Vector (50, 185, 0));
65    mm1->SetPosition (Vector (220, 185, 0));
66    mm2->SetPosition (Vector (300, 270, 0));
67    mm3->SetPosition (Vector (300, 100, 0));
68    mm4->SetPosition (Vector (370, 185, 0));
69    mm5->SetPosition (Vector (540, 185, 0));
70
71    // Estas posiciones fueron encontradas experimentalmente, de tal forma de llegar
          a una cierta topología donde haya ruteo.
72
73    //-- (fin comentario anterior)
74
75    ns3::discovery::RoutingProtocol discovery;    // Se declara un objeto de la clase
           RoutingProtocol implementada.
76
77    DiscoveryHelper discvHelper;                  // Se declara un objeto Helper para
           la uitlización del protocolo Discovery (para definir qué clase de
          enrutamiento se utilizará).
78    InternetStackHelper internet;                 // Se declara un objeto Helper para
           la declaración del stack de protocolos de Internet a utilizar.
79    internet.SetRoutingHelper (discvHelper);      // Se asocia el protocolo Discovery
           al stack de protocolos Internet a utilizar.
80    internet.Install (c);                         // Se instalan los protocolos de
          Interent al stack a utilizar.
81
82    Ipv4AddressHelper ipv4;                                       // Se declara un
          objeto helper para la utilización de direcciones IPv4.
83    ipv4.SetBase ("10.1.1.0", "255.255.255.0");                   // Se configura la
          dirección IP base y la máscara de la red a crear.
84    Ipv4InterfaceContainer ifCont = ipv4.Assign (devices);        // Se asignan
          direcciones IP a cada dispositivo, y se retorna un contenedor de interfaces
          con estos dispositivos configurados.
85
86    double stopApp = 15.0;                  // Es el tiempo en que la aplicación
          termina de ejecutarse (en segundos).
87    double intervalBroadDst = 2.0;          // Intervalo de tiempo después del cual se
          enviará un nuevo msje Discovery desde el destino (en segundos).
88    discovery.CreateEvBroadDst (stopApp, intervalBroadDst);       // Se planifican
          los eventos de difusión de paquetes discovery, con los parámetros definidos
          antes.
89    discovery.SetIndexDest (6);                                   // Se asigna el
          nodo 6 como nodo destino para el algoritmo de ruteo. Recordar que este
          algoritmo de ruteo considera sólo un destino posible una vez comenzado.
90
91    // Crear una aplicación UdpServer en el nodo 5
92
93    uint16_t port = 8000;                                     // Se define una variable
          para el número de puerto de la aplicación servidora.
94    UdpServerHelper server (port);                            // Se declara un objeto
```

```
        Helper para la implementación de la aplicación servidora , en este caso
        utilizando el protocolo de transporte UDP, la que escuchará solicitudes de
        conexión a través del puerto 8000.
95
96    ApplicationContainer apps;                              // Se crea un contenedor de
        aplicaciones
97    apps = server.Install (c.Get (5));                      // Se instala la aplicación
        servidora UDP en el nodo 5 contenido en el contenedor c y se agrega al
        contenedor de aplicaciones.
98
99    apps.Start (Seconds (6));              // Se configura el tiempo de inicio de las
        aplicaciones contenidas en apps. En este caso, en el segundo 6 de simulación.
100   apps.Stop (Seconds (stopApp));         // Se configura el tiempo de detención de
        las aplicaciones contenidas en apps. En este caso, en el segundo 15 de
        simulación.
101
102   // Crear una aplicación UdpClient para enviar datagramas UDP desde el nodo 0
103
104   uint32_t MaxPacketSize = 1024;                          // Se define la cantidad m
        áxima de datos a transmitir en un paquete UDP (sin considerar los encabezados)
        .
105   Time interPacketInterval = Seconds (interval);          // Se define un objeto de
        la clase Time con el tiempo entre dos comunicaciones de paquete consecutivas.
106   uint32_t maxPacketCount = 8;                            // Se define el máximo n
        úmero de paquetes a transmitir por la aplicación.
107   UdpClientHelper client (ifCont.GetAddress (5), port); // Se declara un objeto
        Helper para la implementación de la aplicación cliente utilizando el protocolo
         de transporte UDP, la que se conectará al servidor ubicado en la dirección
        obtenida gracias al método GetAddress del objeto ifCont, a través del puerto
        8000.
108
109   client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));   // Se
        define el máximo de paquetes a transmitir por la aplicación
110   client.SetAttribute ("Interval", TimeValue (interPacketInterval));    // Se
        define el intérvalo de tiempo entre dos comunicaciones de paquetes de dato
        consecutivas.
111   client.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));    // Se
        define el tamaño máximo de datos a transmitir en un paquete UDP (sin
        encabezado).
112
113   apps = client.Install (c.Get (0));                      // Se instala la aplicación
        cliente en el nodo 0.
114
115   // Tracing
116   wifiPhy.EnablePcap ("discovery-pcap", devices);         // Se activa el registro de
        tramas en un archivo Pcap, compatible con el sniffer Wireshark.
117
118   apps.Start (Seconds (7.0));                             // Se configura el tiempo
        de inicio de la aplicación cliente
119   apps.Stop (Seconds (stopApp));                          // Se configura el tiempo
        de término de la aplicación cliente
120
```

```
121    //AnimationInterface anim (animFile);
122
123
124    Simulator::Stop (Seconds (totalTime));          // Se configura el tiempo de t
           érmino de la simulación.
125    Simulator::Run ();                               // Se ejecuta la simulación.
126    Simulator::Destroy ();                           // Se liberan los recursos
           utilizados por la simulación.
127
128    return 0;      // Fin!!
129  }
```


Agradecimientos

Este tutorial fue financiado por el Fondo de Desarrollo de la Docencia (FDD) de la Universidad del Bío-Bío (Proyecto FDD2011-02: "Tutorial para la simulación de algoritmos de ruteo en redes alámbricas e inalámbricas utilizando ns-3").

Referencias

[1] Rfc 3626: Optimized link state routing protocol (olsr). `https://www.ietf.org/rfc/rfc3626.txt`, 2003.

[2] Lacage, M., and Henderson, T. R. Yet another network simulator. In *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator* (New York, NY, USA, 2006), WNS2 '06, ACM.