UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

Facultad de Ciencias Físico – Matemáticas
"Luis Manuel Gutiérrez Rivera"

Implementación del árbol de etiquetas en el simulador ns-3

T E S I S

para obtener el título de:
Licenciado en Ciencias Físico – Matemáticas

Autor:
Eduardo López Bolaños

Asesor:
Cuauhtémoc Rivera Loaiza

Co-asesor:
Héctor Tejeda Villela

Diciembre, 2016
Morelia, Michoacán

Implementación del árbol de etiquetas en el simulador ns-3

Eduardo López Bolaños

12 de diciembre de 2016

A mi mamá China
y mi abuelita María

Agradecimientos

Primeramente, agradezco la atención de los sinodales que aceptaron revisar este trabajo. Al profesor Cuauhtémoc, por aceptar ser responsable de la tesis y por las grandes hazañas que se han logrado. Al profesor Mota, por siempre compartir alegría y buen humor en sus labores de docencia e investigador. A la profesora Karina por compartir su conocimiento en los diversos cursos impartidos y proyectos emprendidos. A Chucho y Edgardo por brindar atención, solución e ideas usando computación. Gracias. ¡Sin duda el tiempo que dedicaron es invaluable!

Así mismo, de una manera muy especial quiero agradecer al profesor Héctor Tejeda, con quien surgió la idea de desarrollar el presente tema de tesis. Gracias por los cursos impartidos, por la concesión de prórrogas para la entrega de proyectos finales, por siempre haberme recibido espléndidamente en su cubículo a pesar de las largas ausencias, por las charlas acerca de fútbol y por el *soundtrack* a bajo volumen escuchado al discutir alguna idea. ¡Muchas gracias profe!

No me queda duda de que mis padres están complacidos por haber concluido este reto. Todo lo que soy se los debo a ustedes. Muchas gracias por haberme brindado el regalo de la vida. Gracias por su motivación para buscar ser una persona íntegra y de éxito. Gracias por las clases que me brindaron en el hogar, mi primer escuela. Gracias también por haberme regalado un par de hermanas, Mariana y Andrea, que de alguna forma me han mostrado varias lecciones.

Por otro lado, agradezco la dicha de poder compartir grandes momentos de la vida a lado de Yesi bonita. Sin duda has sido un pilar fundamental en este recorrido desde aquél 2008-2008 en el laboratorio de física general, donde nos conocimos. Gracias por permitirme conocerte y dejarme formar parte de tu vida. Te ofrezco una disculpa por las molestias que he ocasionado. ¡Te amo!

Ha llegado el turno de agradecer a la Facultad de Ciencias Físico-Matemáticas y a la Universidad Michoacana de San Nicolás de Hidalgo por todas las facilidades que ofrece esta casa de estudios y de la cual me siento muy orgulloso egresar. Una de las muchas alegrías que me brindó estudiar en este lugar, fue el conocer a

gratas personas. Gracias por haberme acompañado en el trayecto de la licenciatura. Gracias a (la banda de 'Los del cubo') Augusto, Pelayo, Tero, Griego, Gaby, Bere, Ulises; al profesor Joaquín Estévez, por recibirnos durante tanto tiempo en su oficina, también a Gabino Corona, Ana Chávez, Héctor Alonso, Toño Corona, Oscar Trejo, Rob Lara, Jorge Fuentes, Ana Castro, Jorge Morales, Larissa Malváez y a todos los amiguines y compañeros con los que compartí algún curso o que frecuenté a lo largo de este camino.

Por otro lado gracias a mis camaradas Dago, Chame, Pony, Chús, Buba, Mijail y Canillas por ser ese factor externo con el que de vez en cuando había distracción, fútbol, chascarrillos y buen ambiente.

Es claro que muchas de las personas que mencioné, habían esperado durante bastante tiempo la conclusión de este trabajo de tesis. Sólo me resta decirles... ¡Gracias por su paciencia!

Por último, me gustaría agradecer a la vida, pues me ha dado la oportunidad de llegar hasta este punto, donde valoro que al concluir con este trabajo no sólo me han quedado varias lecciones académicas. Gracias.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea el resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Resumen

Un algoritmo de enrutamiento para redes de computadoras es el proceso que permite buscar la mejor opción dentro del conjunto de caminos disponibles para enlazar dos nodos de una red.

Las redes de tipo ad-hoc se caracterizan por ser infraestructuras de dispositivos con poca o nula planificación, las cuales se despliegan en tiempo y espacio limitado; por lo que no requieren de una administración central o fija para su operación.

En esta tesis se encara el reto de realizar la implementación del algoritmo de enrutamiento para redes ad-hoc *Árbol de etiquetas* tomando como base teórica el algoritmo *Greedy Perimeter Stateless Routing* (GPSR). El objetivo es obtener una optimización del primero sobre el segundo, mediante el uso de rutas más efectivas y obtener una disminución del tiempo de entrega de paquetes entre los nodos en comunicación.

El *software* elegido para realizar dicha implementación es el simulador ns-3, el cual está basado en la programación orientada a objetos y es de licencia tipo GNU. Para alcanzar el objetivo de la implementación se partió sobre un código previo del GPSR desarrollado en una versión antigua del simulador.

La contribución de este trabajo es lograr la adaptación de la implementación del GPSR tomada como base a versiones recientes del ns-3. Así mismo, después de la actualización requerida se logra realizar la modificación de la estrategia de recuperación y como resultado se obtiene la implementación del Árbol de etiquetas. Esto es viable gracias a que el simulador está basado en C++ y hace permeable la adaptación, extensión y creación de las clases necesarias.

De esta manera, una vez implementado el algoritmo es mediante la comparación de resultados obtenidos al correr las simulaciones, que se puede observar y comparar las características y ventajas del Árbol de etiquetas sobre el GPSR.

redes, ad-hoc, implementación, enrutamiento, ns-3

Abstract

A routing algorithm for a network of computers is the operation that determines the best path that allows to connect two routing points (called nodes) in a network. This search is realized inside the set of all availables paths.

An ad-hoc network is a local area network of computers that is characterized by having an infrastructure of little or non-existing plannig, and which are set in a very limited time and space; therefore, they do not require a central or fixed administration to communicate among them to be operational.

In this thesis, the challenge of making possible the implementation of the routing algorithm for ad-hoc networks *Tagtree* is face by taking as a theoretical base the GPSR (Greedy Perimeter Stateless Routing) protocol. The objective is to obtain an optimization of the first algorith over the second one through the use of more effective routes and to obtain a decrease of the delivery time of data packets to and from any one node to another.

The chosen software to do such an implementation was the ns-3 simulator, which is based on object-oriented programming and operated under GNU type of licence. To reach this objetive, we started the development from a previos GPSR code that came from an earlier version of the simulator.

The contribution of this work is to achieve the adaptation of the implementation of the GPSR, taken as a base, to recent versions of the ns-3. Similarly, after the needed update, the modification of the strategy of recovery is possible and as a result the implementation of the Tagtree is obtained. This is capable of development thanks to and because the simulator is based on C++ and makes it flexible to adaptation, extension and creation of the needed classes.

In conclusion, once the algorithm is implemented through the comparison and results after running the simulations, it is possible to observe and compare the characteristics and advantages of the Tagtree over GPSR.

Índice general

Índice de figuras

Introducción

Parte de la motivación en el desarrollo de este trabajo de tesis recae en la importancia que tienen las redes de computadoras en nuestros días; el hecho de poder entablar comunicación desde cualquier rincón del planeta que cuente con la tecnología adecuada resulta fascinante. Sin lugar a dudas, el ser humano debe aprovechar al máximo este tipo de recursos para lograr avances en pro de la conservación y cuidado tanto del planeta como del propio ser humano.

En el lenguaje cotidiano se emplea el uso del término 'red' y se tiene una idea intuitiva de cómo se forma una, todo esto debido al uso de dispositivos como computadoras, teléfonos móviles, consolas de videojuegos, tabletas, televisores inteligentes, etc.; que tienen la capacidad de conectarse a la gran red que es el internet. Pero, plantear formalmente los conceptos clave de todos los elementos involucrados en una red puede resultar un poco más extenso.

Estructura de la tesis

Por lo anterior, en el capítulo 1 'Preliminares' se pretende formalizar el concepto de redes de computadoras. Se intenta explicar los elementos que las forman, la manera práctica en que se pueden representar mediante teoría de grafos, la clasificación de las redes según su topología; todo esto para concluir con la definición de: red ad-hoc. Este tipo de redes son la piedra angular en este documento.

También, en la segunda sección del primer capítulo se da una breve descripción de los distintos tipos de *software* utilizados para desarrollar simulaciones de redes. Es en esta parte donde se presentará el programa utilizado para desarrollar el presente trabajo de tesis, el *ns-3*.

En el capítulo 2, denominado 'Algoritmos de ruteo para redes ad-hoc', se retoma y extiende el concepto de red ad-hoc para dar una descripción acerca de la manera en que funciona el intercambio de información en este tipo de redes.

Además es en el segundo capítulo donde se presenta la descripción completa del objetivo principal de esta tesis: Realizar la implementación del algoritmo de

encaminamiento conocido como **Árbol de etiquetas** en el simulador ns-3 usando como referencia el algoritmo de Encaminamiento Voraz Progresivo Perimétrico Local (GPSR, por sus siglas en inglés).

Para lograr el objetivo que se persigue, se usará el trabajo expuesto en dos publicaciones principalmente. La primera de ellas habla de una implementación del algoritmo GPSR utilizando la versión del simulador ns-3.12, sin embargo, dicho trabajo ya no es funcional para versiones más actuales. En la segunda publicación tomada como referencia, los autores presentan el algoritmo de encaminamiento Árbol de etiquetas, dando la descripción detallada de éste.

Una vez presentado el objetivo que se persigue en este trabajo y conociendo el fundamento teórico del algoritmo árbol de etiquetas, en el capítulo 3, 'Diseño y desarrollo del árbol de etiquetas', se describe el proceso de actualización del algoritmo GPSR a la versión ns-3.22, sobre la cual se desarrolla la presente tesis.

De igual manera, dentro de este tercer capítulo se mencionan algunos desafíos y retos que aparecieron a la hora de implementar el algoritmo. Así mismo, se describen también las adecuaciones que se tomaron para manejar la estructura *PositionTree* requerida en el proceso de recuperación.

Para finalizar, es en el capítulo 'Conclusiones', donde se aterrizan ideas respecto al logro de haber conseguido la implementación del algoritmo que se planteó como objetivo. Además, se mencionan algunas ideas que pueden desarrollarse a manera de un proyecto posterior para mejorar y optimizar este trabajo.

Aportaciones de la tesis

En este trabajo se pueden rescatar un par de aportaciones como novedad. La primera de ellas es el adecuamiento de una implementación del algoritmo GPSR a una versión más reciente del ns-3. De esta manera, se puede completar el trabajo elaborando la documentación requerida por parte de los desarrolladores del simulador y así, una vez revisada la depuración que se planteó, poder integrar este protocolo de enrutamiento de manera oficial.

Otra aportación, considerada la principal de este trabajo, es poder implementar el algoritmo de enrutamiento basado en la posición nombrado por sus autores como árbol de etiquetas. A la par del trabajo que se desarrolló sobre el GPSR, se tiene un avance sobre estos dos tipos de algoritmos que pueden utilizarse en la ejecución de redes ad-hoc en el simulador ns-3.

En pocas palabras, la aportación de este trabajo de tesis es la construcción de un módulo para el simulador ns-3 con el cual se pueda emplear el algoritmo de enrutamiento árbol de etiquetas.

Capítulo 1

Preliminares

En este capítulo se abordarán cuestiones técnicas respecto al concepto de redes de computadoras. En la sección 1.1, se puntualizará sobre su definición, además se dará una pequeña descripción sobre los elementos que componen una red, la manera en que se pueden representar las redes de forma práctica mediante grafos y posteriormente, se enunciarán algunos protocolos de comunicación que aparecen en las redes. Por último, se puntualizará sobre qué son las redes ad-hoc, ya que bajo este concepto se desarrolla este trabajo de tesis.

Enseguida, en la sección 1.2 se hablará acerca de los simuladores de redes. Se describirán algunas de sus características básicas y se enlistarán algunos de ellos. De éstos, el que resaltará será ns-3, ya que es el programa empleado para desarrollar las simulaciones de este trabajo.

1.1. Redes de computadoras

En informática, la definición que puede darse acerca de *'red'*, es:

Definición 1. Conjunto de computadoras o de equipos informáticos conectados entre sí y que pueden intercambiar información. [Esp01, 'red']

Es así como las redes de computadoras desde su aparición, han traído la posibilidad de elevar a altos grados la comunicación en todo el mundo, ya que el propósito de una red es permitir compartir archivos e información entre múltiples sistemas.[Chr16, 'network']

Elementos de una red

Para formar una red es necesario tener varios elementos que interactúan de manera conjunta, estos pueden referirse tanto al hardware como al software.

A continuación se hará una breve descripción acerca de los elementos de las redes, de acuerdo a su funcionamiento lógico[Fou16a]:

- **Servidores:** Son dispositivos o computadoras que mantienen archivos compartidos, programas y soportan el sistema operativo de red. Los servidores proveen del acceso a todos los recursos disponibles a los usuarios de la red. Algunos tipos de servidores que existen son: de archivos, de impresión, de correo, de comunicación, de bases de datos, de web, etc.

- **Clientes:** Los clientes son dispositivos o computadoras que acceden y usan la red y sus recursos compartidos. Estos dispositivos son básicamente los usuarios de la red que solicitan y reciben los servicios proporcionados por el servidor.

Al ser los servidores y clientes elementos de una red, éstos serán denominados nodos.

Definición 2. Los *nodos* se describen como computadoras o dispositivos a los cuales se les puede agregar una funcionalidad y se pueden conectar a una red.[Nat15b, p. 23]

- **Medios de transmisión:** Este apartado se refiere a los medios utilizados para interconectar los dispositivos y computadoras en una red. Algunos de los medios utilizados son: cable coaxial, cable UTP, fibra óptica, conexión inalámbrica y ondas de radiofrecuencia.

- **Datos compartidos:** Son datos que los servidores proporcionan a los clientes, tales como archivos, programas, acceso a la impresora o al correo electrónico.

Dado que el propósito de las redes es intercambiar información y archivos, el concepto utilizado para referirse a la transferencia de datos se denomina recepción y envío de *paquetes*.

- **Periféricos compartidos:** Se definen como recursos *hardware* a los cuales los servidores proporcionan acceso a los usuarios de la red. Algunos ejemplos de estos periféricos son: monitores, impresoras, videocámaras, micrófonos o cualquier otro elemento físico usado por los clientes de la red.

- **Tarjeta de Interface de Red:** Cada dispositivo que se localiza en una red tiene una tarjeta de expansión especial llamada tarjeta de Interface de Red. (*Network Interface Card* [NIC]), la cual prepara, envía, recibe y controla el flujo de datos entre el dispositivo y la red. Su labor de transmisión consiste en pasar el marco o trama de datos de la capa física a través de un medio de transmisión. Su labor de recepción va encaminada en procesar los bits recibidos desde el enlace físico y procesar el mensaje de acuerdo a su contenido.

  Algunos ejemplos de tarjetas de interface de red son: las tarjetas de red alámbricas que reciben los cables UTP (Ethernet [10 Mbps] y Fast Ethernet [100 Mbps]) mediante los conectores RJ-45, las tarjetas de red inalámbricas que permiten las conexiones con la tecnología denominada *Wi-Fi* y los receptores de señal que permiten conexiones inalámbricas de corto alcance denominadas *Bluetooth*

- **Sistema Operativo Local:** Un sistema operativo local permite a los dispositivos acceder a archivos, realizar impresiones y usar dispositivos de almacenamiento localizados en el equipo. Algunos ejemplos de sistemas operativos son: MS-DOS, Unix, Linux, Windows, iOS, Android, etc.

- **Sistema Operativo de red:** Consiste en un programa que corre en los dispositivos tanto clientes como servidores y permite la comunicación en la red.

- **HUB:** Es un dispositivo de red que divide una conexión entre varias computadoras. Simula un centro de distribución. Cuando en una red se requiere información de un dispositivo específico, se manda la requisición al HUB, quien la recibirá y la transmitirá a la red entera. De esta forma corresponde a cada dispositivo determinar si el mensaje es para él o no.

- **Switch:** Es un dispositivo de telecomunicaciones agrupadas. Tiene un funcionamiento similar al HUB, pero con características avanzadas, ya que usa

la dirección del dispositivo físico a quien se le envía un mensaje o archivo para entregarlo justo al destinario que empata con dicha dirección o puerto.

- **Router:** Consiste en un dispositivo que conecta flujo de datos de redes locales a otras redes. Este *hardware* permite sólo la conexión de máquinas o dispositivos que están autorizados para acceder a los sistemas de las computadoras.

A continuación, se muestran los elementos de una red mediante su representación gráfica. Estos íconos son parte de la extensión VRT NetworkEquipment[1] para el *software* LibreOffice, cuyo uso se permite bajo licencia libre. Observe la figura 1.1.

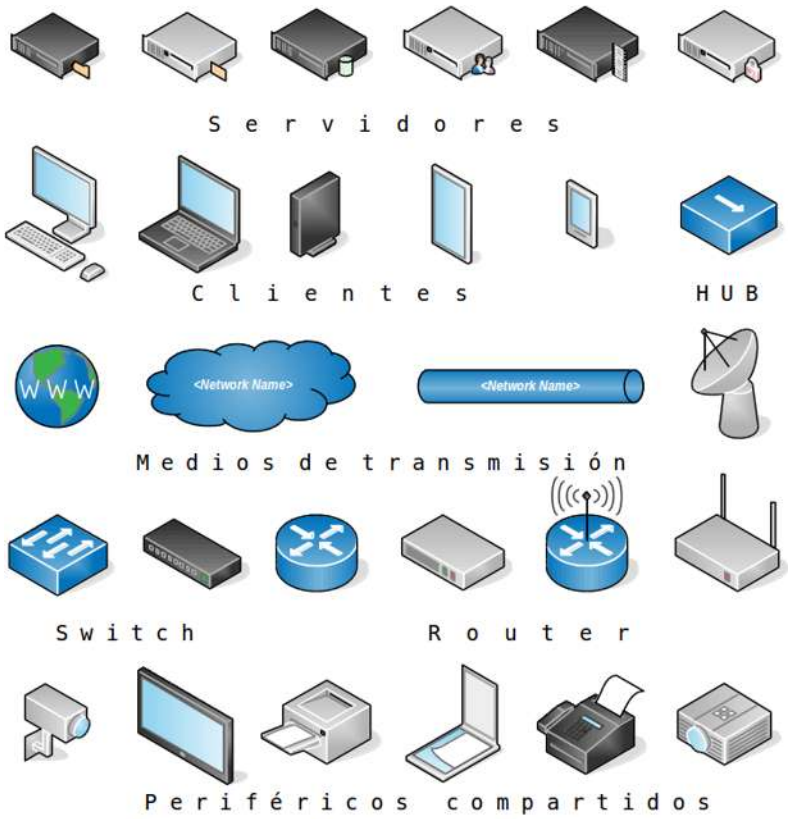


Figura 1.1: Elementos de una red

[1] extensions.libreoffice.org/extension-center/vrt-network-equipment/releases/1.2.0

Representación de las redes

Para llevar a cabo la representación de una red de una manera visual y compacta puede usarse como herramienta la Teoría de Gráficas manejada en matemáticas discretas.[PCL01, p.101]

Definición 3. Una *gráfica* (o *grafo*) $G = (V, A)$ es una estructura combinatoria constituida por un conjunto V de elementos llamados vértices y un conjunto A de pares no ordenados de vértices distintos llamados aristas.

Definición 4. Dada una gráfica G, se dice que el *orden* del grafo $| V(G) |$, corresponde al número de vértices de G y el *tamaño* de la gráfica $| A(G) |$, representa el número de aristas. Por otro lado, para un vértice $u \in V$, el grado del vértice $d(u)$, indica el número de vértices adyacentes.

Sean $u, v \in V$ vértices y $a \in A$ arista. Si $a = (u, v)$, se dice que u y v son vértices adyacentes o también que la arista a es incidente a los vértices u y v, es decir están enlazados.

La mayoría de las veces, la manera útil y práctica de representar una gráfica es mediante un dibujo donde los vértices se representan como puntos y las aristas como líneas que unen los vértices adyacentes.

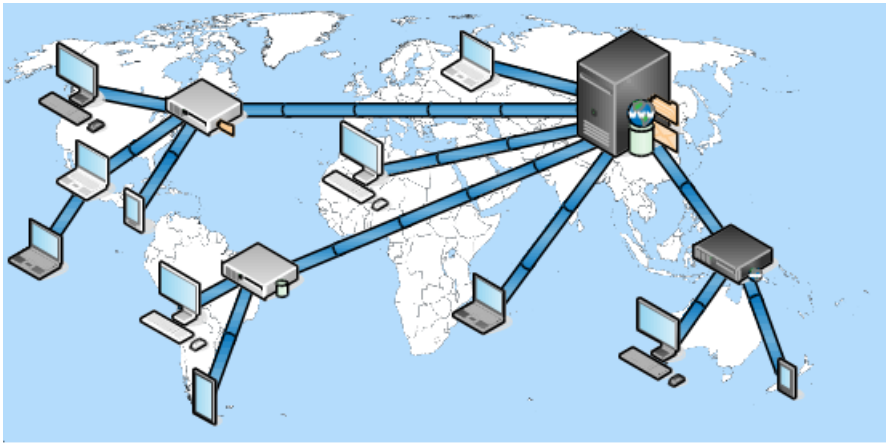


Figura 1.2: Ejemplo de una red de computadoras.

De esta manera un grafo puede modelar una red de computadoras. Los vértices de la gráfica corresponderán a simular los nodos de la red y las aristas los enlaces que se dan a través de los medios de transmisión. Por ejemplo la red de la figura 1.2, se puede interpretar como la gráfica que se muestra en la figura 1.3.

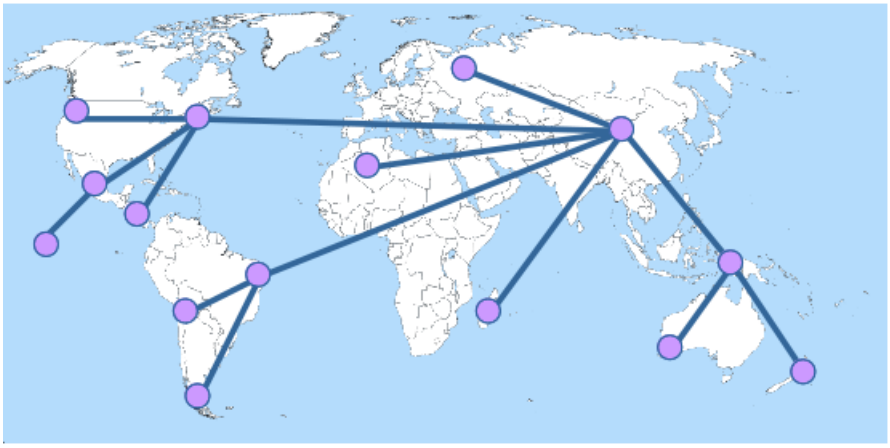


Figura 1.3: Interpretación de una red de computadoras como gráfica.

Para este trabajo se omitirán los sentidos de los enlaces. Es decir si dos nodos se encuentran adyacentes, se dirá que hay un canal de comunicación entre ellos y ambos dispositivos fungirán como emisores y receptores de información.

Otro concepto importante utilizado en la caracterización de una red mediante la teoría de gráficas es el de recorrido.

Definición 5. Dada una gráfica $G = (V, A)$, un *recorrido* S, es una secuencia de vértices $u_0, u_1, ..., u_i \mid (u_{i-1}, u_i) \in A$, además $1 \leq i \leq l$ y $u_{i-1}, u_i \neq u_{j-1}, u_j$. La longitud del recorrido $|R|$ es l y se dice que el recorrido va de u_0 a u_l.

Definición 6. Un *circuito* es un recorrido cerrado, es decir $u_0 = u_l$. En esta secuencia puede haber repetición de nodos. Cuando todos los vértices de un recorrido son distintos se tiene un *camino*. De aquí se desprende el concepto de *ciclo*, el cual es un camino cerrado.

Ejemplo 7. Sea G la gráfica conformada por los vértices: u, v, w, z, t. Visualice las siguientes secuencias en la figura 1.4.

- $S_1 = u, v, w, z, v, t$ es *recorrido*.
- $S_2 = u, v, z, w, t$ es *camino*.
- $S_3 = u, v, w, z, v, t, u$ es *circuito*.
- $S_4 = u, v, z, w, t, u$ es *ciclo*.

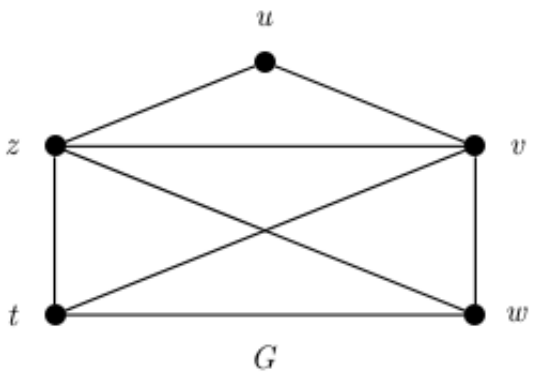


Figura 1.4: Gráfica G de orden 5 y tamaño 8.

Definición 8. Sea G una gráfica. Si para todo par de vértices existe un camino entonces se dice que la gráfica es *conexa*. Esto significa que la gráfica cuenta con sólo un componente de vértices adyacentes.

Definición 9. La *distancia* entre dos vértices adyacentes u y v, denotada por $d(u, v)$, se considera como la longitud mínima de un camino entre estos vértices.

Cuando se afirma que un grafo es conexo, la interpretación que se da es que para todos los nodos que forman parte de la red se puede establecer comunicación entre ellos. Además, para la redes se suele decir que la distancia entre dos nodos es el número de *saltos* que tiene que realizar un paquete para llegar del emisor al receptor.

Definición 10. Un *árbol* es un grafo conexo y sin ciclos. Se suele representar un árbol con la letra T. Si un árbol es de orden n, su tamaño será $n - 1$.

Definición 11. El *diámetro* de una gráfica, es denotado como D, se considera como la más grande de las distancias que hay en el grafo.

$$D = \max_{u,v \in V}\{d(u, v)\}$$

Así mismo se tiene el concepto de *excentricidad* de un vértice, este concepto se representa con el símbolo $e(u)$ y se define como la máxima de las distancias entre $u \in V$ y los otros vértices de G. La mínima de las excentricidades de los vértices de la gráfica recibe el nombre de *radio*, denotado por r.

Definición 12. El *centro* de un grafo, representado con Z, se considera como el conjunto de vértices con excentricidad igual al radio.

Al utilizar la teoría de gráficas como medio de representación de una red, surgen dos cuestiones de análisis para ésta. El primero es el concepto de fiabilidad, es decir la capacidad que tiene la red para seguir operando a pesar de que haya una falla en sus nodos o enlaces. La segunda característica se enfoca en visualizar si la red permite algoritmos de encaminamiento de los paquetes de manera óptima.

En este trabajo, se hará profundo énfasis en el ruteo de las redes.

Clasificación de las redes

La estructura de las redes puede ser descrita desde dos perspectivas: de acuerdo a su *topología física* o en base a su *topología lógica*.[Fou16d]

Topología física

Comenzaremos a describir los tipos de redes que existen en base a la topología física, es decir de acuerdo a la distribución posicional del hardware que forma la red. Este primer análisis se puede agrupar en 3 categorías:

1. **Redes centralizadas:** En estas redes, existe un nodo al cual están adyacentes todos los dispositivos que componen la red. Este nodo representa el centro de la red, por lo tanto $|Z| = 1$. El resto de los dispositivos se consideran *periféricos*.[2] Toda la comunicación de la red se da a través del nodo central, dando la desventaja de que si surgiera un incidente en éste, se inhabilita el flujo de datos en la red. Es decir la fiabilidad de la red se ve expuesta de manera total si hay un fallo en este nodo. Las conexiones de un HUB con todos sus dispositivos es un ejemplo de una red centralizada. Este tipo de distribución se conoce como: arreglo de red tipo estrella. Véase la figura 1.5. A los nodos que se encuentran en la periferia, también se les conoce como *nodos hoja*, caracterizados por sólo tener una arista incidente.

   Observación 13. Una red centralizada es un grafo de tipo árbol. La configuración más característica de este tipo de redes, es la que se representa en el tipo estrella. Además toda comunicación que se quiera establecer entre nodos periféricos pasa por el nodo central. Este modelo tiene como características: $D = 2$, $r = 1$ y $|Z| = 1$.

[2] Aparato auxiliar e independiente conectado a la unidad central de una computadora.[Esp01, 'periférico']

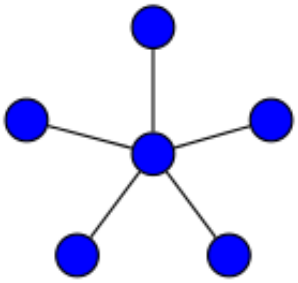


Figura 1.5: Arreglo de red tipo estrella

Existen otros modelos que extienden el concepto de red centralizada. En éstos, se puede dar el caso que $|Z| = 2$ y con cierta característica se preserva que el grafo es un árbol. Más adelante se demostrará dicha característica en la proposición 15. Las topologías que pueden tener dos centros son:

- **Modelo Punto a Punto:** Es el modelo de topología más sencillo que aparece cuando el centro de la gráfica tiene dos nodos. Se suele abreviar con las siglas: **p2p**, por su nomenclatura en inglés *Point To Point*. Consiste en un par de nodos que se encuentran enlazados y éstos pueden compartir información. La única solicitud que se pide es que los nodos tengan el medio de enlace para mostrar su adyacencia mútua, ya que si ésta desapareciera, entonces no habría conexión. Un ejemplo de este pequeño modelo se visualiza en la figura 1.6.

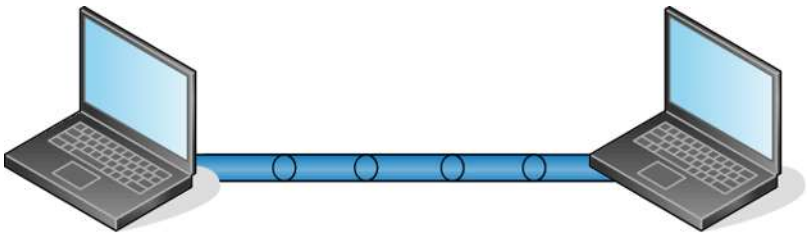


Figura 1.6: Topología del modelo punto a punto

- **Estrella extendida:** Este modelo de topología está basado en el tipo estrella, con la característica de que tiene *nodos repetidores*, es decir dispositivos enlazados con nodos del centro de la gráfica, que permiten la conexión con los nodos de la periferia. Los repetidores son usados

para extender al máximo la cobertura de transmisión entre los nodos. Un nodo que no sea periférico y esté conectado a un repetidor, también se considerará como repetidor. Véase la figura 1.7.

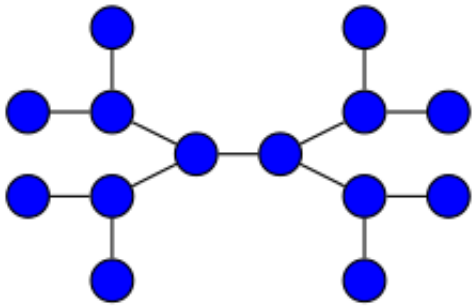


Figura 1.7: Arreglo de red tipo estrella extendida

Una ventaja del modelo estrella extendida es que en caso de alguna incidencia en un nodo central o en alguno de los nodos repetidores, la red quedará parcialmente conectada. Asociando este concepto a la teoría de grafos, los nodos centrales y repetidores pueden ser llamados *vértices de corte*[3].

Observación 14. Al igual que en el modelo tipo estrella, la comunicación entre nodos de la periferia pasa por los nodos centrales o repetidores.

- **Conexión en serie:** También conocida como la topología modelo *cadena*. Es un tipo de red centralizada, la cual tiene dos nodos considerados como extremos y consiste en una serie de nodos que tienen exactamente dos adyacencias, salvo los extremos que fungen como nodos periféricos. Dichos extremos marcan el inicio y fin de la cadena. De esta manera varios nodos se desenvuelven como repetidores para lograr la transmisión de información. Al igual que en la topología de estrella extendida los vértices que no son extremos actúan como vértices de corte. Una idea general de este modelo se encuentra en la figura 1.8.

[3] Aquellos vértices que al ser suprimidos generan más de un componente de la gráfica. En general para una gráfica conexa, la supresión de un vértice de corte desconecta G en dos o más componentes.

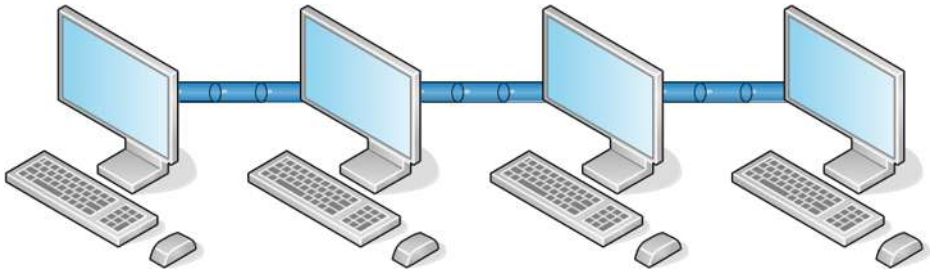


Figura 1.8: Topología del modelo conexión en serie.

- **Estrella distribuida:** En este tipo de modelo, existen los submodelos: *lineal* y *anillo*. Dentro de las redes centralizadas sólo se considerarán las de tipo lineal. Éstas consisten básicamente en grafos disconexos, donde las componentes de la gráfica son modelos del tipo: estrella, punto a punto, estrella extendida o cadena. Así, anexando un enlace entre estos modelos, es decir agregando una arista entre nodos de las componentes de la gráfica que no están conexas; se sigue formando una red centralizada conexa. Se puede verificar que al enlazar dos árboles no conexos mediante una sola arista, la componente resultante es un árbol. Con esta operación se lograría extender la red. El caso más sencillo de una estrella distribuida lineal es el que se muestra en la figura 1.9, en la cual se puede apreciar que agregando un enlace entre dos de los extremos de las cadenas que no están conexas, la componente conexa seguirá siendo árbol y a su vez cadena.

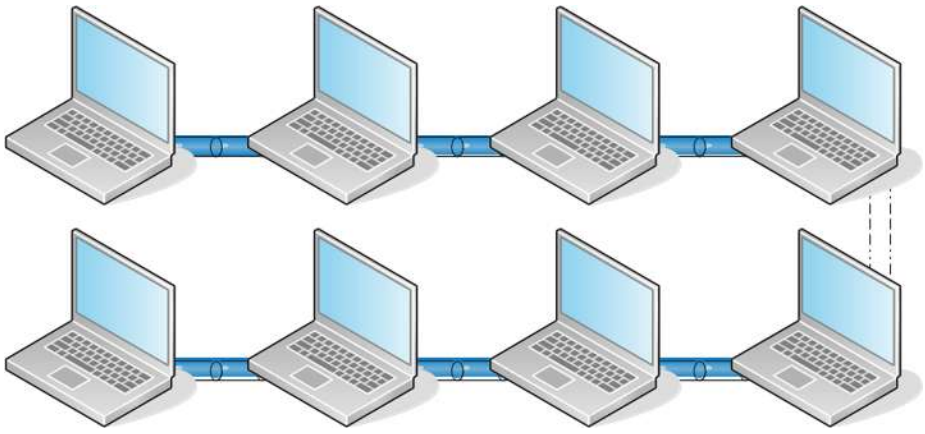


Figura 1.9: Topología del modelo estrella distribuida lineal.

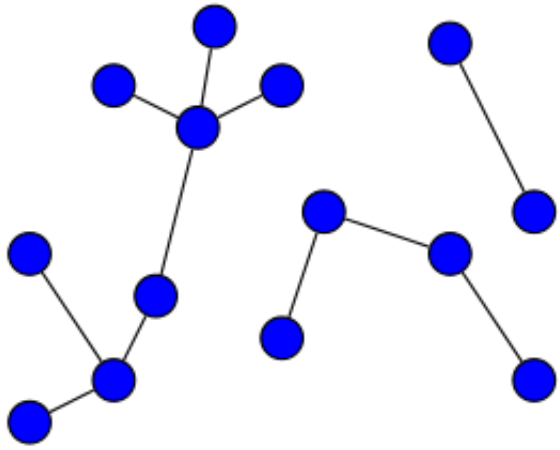


Figura 1.10: Arreglo de red tipo estrella distribuida lineal.

En la figura 1.10, se aprecia una gráfica con tres componentes. El centro de la gráfica se encuentra en la componente que está formada por el modelo p2p, que se aprecia en la parte superior derecha de la figura. Puede apreciarse que $|Z| = 2$. Además, se puede observar cómo agregando una arista entre las componentes disconexas se logra extender la red conexa.

Con los modelos descritos sobre redes centralizadas podemos puntualizar:

a) La representación de las redes centralizadas son gráficas de tipo árbol T.

b) Se caracterizan por tener sólo un nodo central, pero existen casos con dos centros.

A continuación se hablará de la característica que deben tener las gráficas donde $|Z| = 2$, mediante la siguiente:

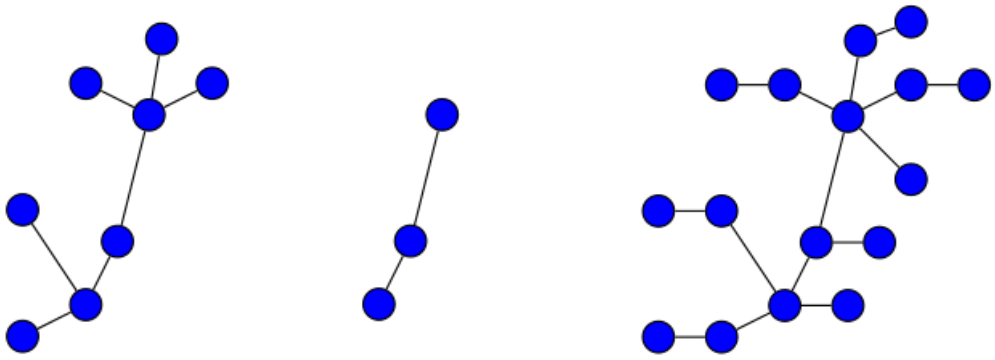
Proposición 15.
- *Un árbol tiene uno o dos centros.*
- *Si el árbol tiene dos centros entonces éstos están adyacentes.*

Demostración. Definamos dos operaciones para un árbol T:

- *Podado de árbol*: Proceso de supresión de todos los nodos hoja de un árbol y las aristas incidentes a éstos. Es decir: $T - \{u \in V \mid d(u) = 1\}$.

- *Reforestación de árbol*: Proceso de anexión de un número de nodos igual al orden del árbol. A cada nodo del árbol se le enlazará uno de los nodos agregados, convirtiéndose éstos en nodos hoja. Por lo que: $\forall v \in V$ se tendrá $v + u_i \mid d(u_i) = 1$ con $0 < i \leq \mid V(T) \mid$.

Gráficamente se aprecian las operaciones anteriores para el árbol T en la figura 1.11.



(a) Árbol T (b) Podado de T (c) Reforestación de T

Figura 1.11: Operaciones en un árbol

Notemos que al realizar una operación de podado o reforestación a un árbol, la excentricidad de todo nodo disminuye o aumenta un entero. Así, puesto que $e(v) > e(z) \Rightarrow e(v) \pm 1 > e(z) \pm 1$ concluimos que los centros se mantienen después de podar y reforestar un árbol.

Ahora procedamos por inducción sobre el radio de un árbol.

Base de inducción: La proposición señalada en 15, se cumple para grafos de orden 2, 3 y 4, donde $r = 1$ o 2.

Hipótesis de inducción: Un árbol T de radio n tiene un centro ó si tiene dos éstos son adyacentes.

Probemos para un radio $n + 1$. Consideremos un árbol T de radio $n + 1$. Apliquemos un podado al árbol, generando así un árbol T' para el cual $r = n$. Por hipótesis de inducción se sabe que T' tiene uno ó dos centros adyacentes. Ahora al aplicar una reforestación para T' se tiene que esta nueva gráfica de radio $n+1$ permanece con los mismos centros, por lo tanto, la proposición es cierta para un árbol de radio $n + 1$. □

2. **Redes descentralizadas:** Son redes que no utilizan un único nodo central, sino que utilizan varios nodos centrales, $|Z| \geq 2$. Analicemos las características de esta categoría de redes. De la primer parte de la proposición 15 tenemos que si se tienen más de dos centros entonces la gráfica no es árbol. Con lo anterior es fácil ver que si $|Z| > 2$ entonces la gráfica tendrá ciclos. Ésta será una característica importante de esta categoría. Ahora restaría ver que existen gráficas descentralizadas cuando $|Z| = 2$. Para esto supongamos una gráfica G con dos centros no adyacentes. Observemos que si G no tuviera ciclos entonces G sería un árbol. Entonces ya que $|Z| = 2$, por la segunda parte de la proposición 15 se tiene que los vértices deben ser adyacentes. Ésto contradice la hipótesis, por lo tanto la gráfica con dos centros no es árbol y los centros no están adyacentes. Véase la figura 1.12. Lo anterior genera el siguiente:

Corolario 16. *Sea G una gráfica conexa tal que $|Z| = 2$. Si los centros no son adyacentes entonces G tiene ciclos.*

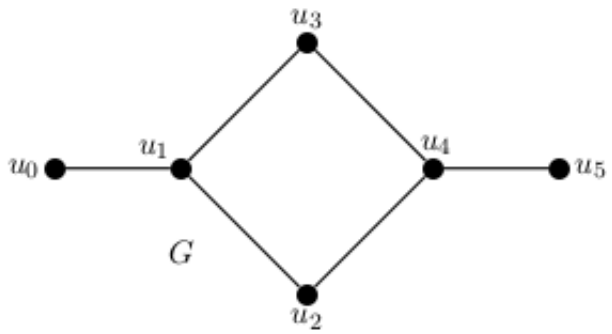


Figura 1.12: Arreglo de red descentralizada con 2 centros

Con lo enunciados anteriores se aprecia que las gráficas descentralizadas se caracterizan por tener ciclos. En la figura 1.12, $Z = \{u_2, u_3\}$. Se muestra una gráfica descentralizada y sus dos centros no están adyacentes. Además demostrando la siguiente:

Proposición 17. *Sea T un árbol. Para cada par de vértices de T existe un único camino que los conecta.*

Demostración. La proposición anterior equivale a probar que:

T es un árbol $\iff$ entre cada par de vértices de T existe un único camino.

$\Longrightarrow$) Sea T un árbol. Supongamos que existen dos caminos distintos R_1 y R_2 para un par de vértices u y v. Supongamos un recorrido de u a v por R_1 y el regreso por R_2 entonces se tendría un ciclo, lo cual contradice la definición de árbol $\therefore$ existe un único camino para cada par de vértices.

$\Longleftarrow$ Sean $u, v \in T$ vértices. Se sabe que para cada par de vértices de T se tiene un único camino, esto lo hace un grafo conexo. Si T tuviera un ciclo existirían al menos dos caminos para u y v. Por hipótesis eso no pasa entonces T es conexo y sin ciclos $\therefore$ T es árbol. □

Se puede apreciar que en las gráficas descentralizadas, hay al menos dos nodos con dos o más caminos entre ellos. Esto provee una ventaja a estas redes, ya que en caso de que hubiera una incidencia en alguno de los nodos que formen parte de un camino y estén en un ciclo, existirá otra opción para mantener la comunicación entre los dispositivos de la red.

Ejemplo 18. Para la figura 1.12, identifique los caminos de la conexión entre u_0 y u_5. Se tiene que $S_1 = u_0, u_1, u_3, u_4, u_5$ y $S_2 = u_0, u_1, u_2, u_4, u_5$. Ahora supongamos que ocurre un fallo en el nodo u_2 entonces la comunicación entre los extremos ya no puede usar S_2, pero no tiene efectos restrictivos para el uso de S_1.

Algunos modelos simples de topología de red que se caracterizan por ser redes descentralizadas son los siguientes tipos:

- **Anillo** *(o circular)*: Este modelo es también conocido como *estrella distribuida anillo*, submodelo mencionado con anterioridad. En este tipo de configuración todos los nodos tienen la misma excentricidad, por lo que todos los vértices pertenecen al centro de la gráfica $|Z| = |V|$. Además se visualiza de forma sencilla como para todo dispositivo de la red se tienen dos caminos para conectarse con los otros nodos. Vea la figura 1.13.
- **Malla**: Este modelo se divide a su vez en dos submodelos: las mallas *parcialmente* y *totalmente* conectadas. Para el caso de redes centralizadas sólo se considerarán las mallas conectadas parcialmente. Éstas representan muy bien las características de las redes centralizadas. Vea la figura 1.14.

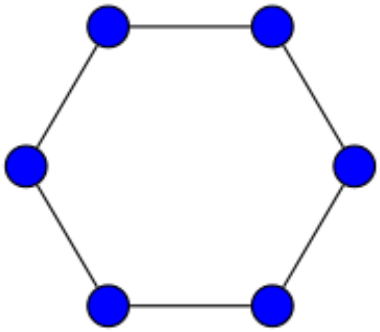


Figura 1.13: Arreglo de red tipo anillo

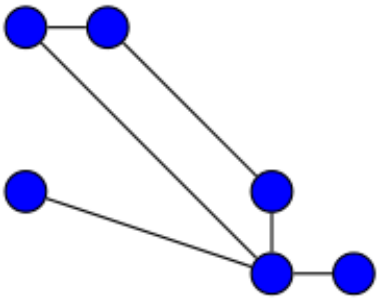


Figura 1.14: Arreglo de red tipo malla parcialmente conectada

- **Híbrido** *(o Mixta)*: Este tipo de arreglo de red surge mediante el enlace y combinación de varios modelos de manera simultánea. Es decir es el resultado de combinar modelos tipo estrella, punto a punto, en serie, anillo y malla. Un ejemplo se puede visualizar en la figura 1.15 donde se tiene una red estrella de anillos.

3. **Redes Distribuidas:** Las redes de este modelo cuentan con más de un nodo central, pero como novedad en estas redes, se pueden encontrar caminos entre nodos sin necesidad de pasar por algún nodo que pertenezca al centro. Esta topología se caracteriza porque al ocasionarse un fallo o incidencia en cualquiera de los nodos, no ocurre la desconexión de algún otro nodo de la red. Además, en estas redes desaparece el concepto de periferia, esto se traduce en que para todo vértice que se encuentre en la red, éste tendrá más de una arista incidente. Sobre este modelo tenemos los siguientes tipos:

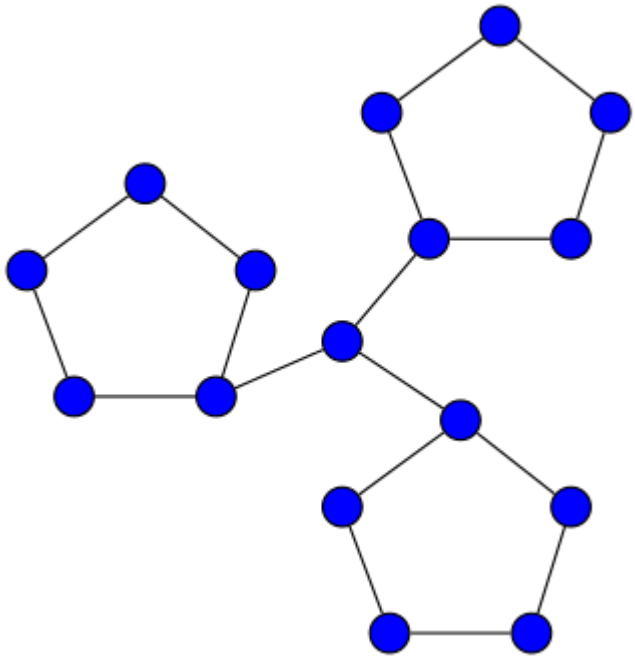


Figura 1.15: Arreglo de red híbrida: estrella de anillos

- **Malla totalmente conectada**: Este tipo de malla tiene como característica que todos los nodos de la red se encuentran adyacentes entre sí. Remover un nodo por alguna falla o incidente deja a los demás nodos formando una malla totalmente conectada. Este tipo de topología lo podemos apreciar en la figura 1.16.
- **Cuadrícula**: A diferencia de las mallas totalmente conectadas, para los nodos de este tipo de red distribuida no se presentan adyacencias hacia todos los demás vértices, pero sí al menos con otros dos vértices de la gráfica. El nombre de *cuadrícula* se debe a que cuando las adyacencias entre los dispositivos no se enciman, las aristas en su representación gráfica forman teselas[4], es decir la región que abarque la distribución de dichos nodos se encontrará cubierta por una teselación[5].

[4]Cada una de las piezas con que se forma un mosaico[Esp01, 'tesela'].

[5][Fou16e] Regularidad o patrón de figuras que cubre o pavimenta completamente una superficie plana que cumple con dos requisitos:

a) Que no queden huecos.

b) Que no se superpongan las figuras.

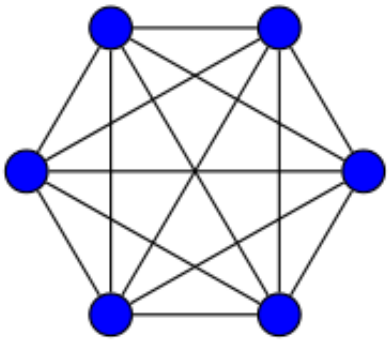


Figura 1.16: Arreglo de red malla totalmente conectada

Podemos observar ejemplos de red distribuida del tipo cuadrícula en la figura 1.17.

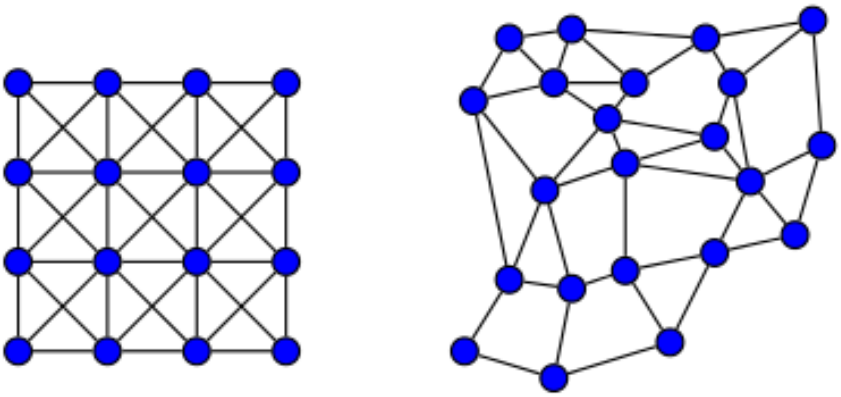


Figura 1.17: Arreglos de red tipo cuadrícula

Del lado izquierdo de la figura 1.17 se puede apreciar que en la representación gráfica de este arreglo de red las aristas se enciman. Sin embargo del lado derecho se visualiza un grafo de tipo cuadrícula donde las aristas no chocan entre ellas, por lo que se forma un teselado.

Topología lógica

La clasificación de la redes en cuanto a su topología lógica hace referencia a cómo se comunican los dispositivos de una red haciendo sólo énfasis a la manera en que se da el flujo de datos independientemente de su diseño físico.

Una manera de hacer énfasis al análisis de una red de acuerdo a la topología lógica es mediante la representación gráfica conocida como tipo **Bus**. Éste se caracteriza por tener un único canal de comunicaciones denominado *bus* o ramal, al cual se conectan todos los dispositivos. Este canal es compartido, por lo que tiene sus ventajas y desventajas. Una representación de la topología lógica se puede visualizar en la figura 1.18.

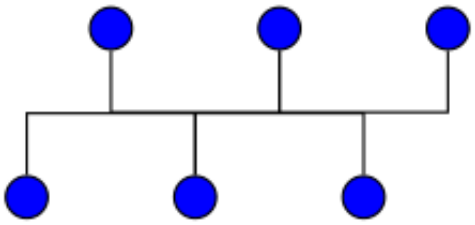


Figura 1.18: Arreglo de red tipo bus

Protocolos de comunicación

La manera en que la información se propaga a través de una red se conoce como: *protocolo*[6]. Existen varias formas en las que se puede llevar a cabo la transmisión de datos. Las tarjetas de red (NIC) están programadas para escuchar los diversos modos de comunicación. Para establecer un protocolo es necesario identificar a un emisor, receptor y paquete. A continuación se analizarán los procesos de envío de datos[CCL03, p.30]

1. **Broadcast** Este modo de transmisión consiste en que a partir del nodo emisor, el mensaje será enviado a todos los nodos que se encuentren a un salto de distancia. Podríamos encontrar dos tipos de broadcast: el *limitado* o *simple* y el *directo* o *total*. El broadcast simple consiste en que un nodo envíe el mensaje a sólo sus vecinos, mientras que el broadcast total consiste en

[6]Conjunto de reglas que se establecen en el proceso de comunicación entre dos sistemas[Esp01, 'protocolo'].

que una vez que se realiza un broadcast simple, los nodos que recibieron el mensaje lo reenviarán de igual manera en modo broadcast y esto continuaría de manera recursiva. Lo anterior puede ocasionar una redundante operación de *re-broadcast* que puede derivar en un problema denominado tormenta *broadcast* que inunda la red en forma catastrófica. Visualicemos un ejemplo de tipo *broadcast* simple en la figura 1.19.

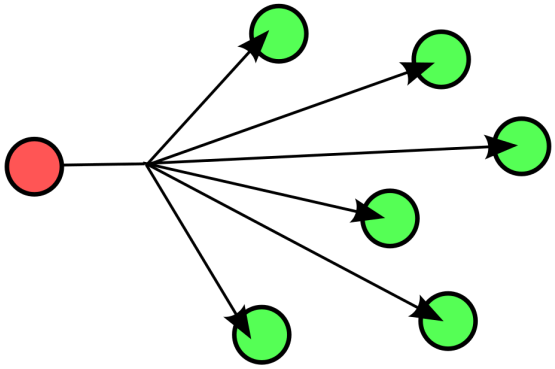


Figura 1.19: Envío de un paquete en modo *broadcast*.

Ejemplo 19. Un caso aplicado para de entender el modo *broadcast*, es la emisión de la señal de un canal televisivo, radiofónico, satelital, etc. La señal propagada es recibida por todos los receptores que sean vecinos al dispositivo que emite la señal. Véase la figura 1.20.

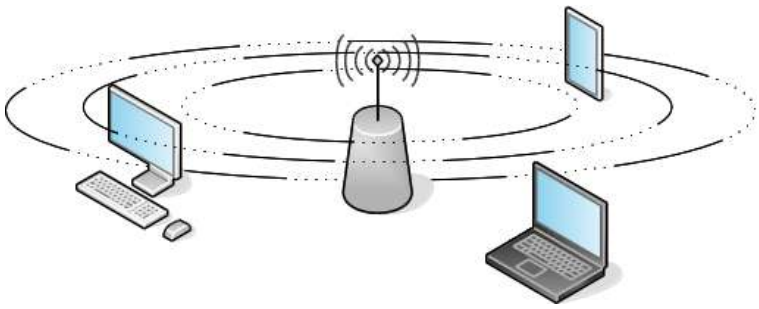


Figura 1.20: Transmisión de tipo *broadcast* simple en una red.

2. **Unicast**: La forma en que se da este modo de transmisión es uno-a-uno. Es decir la fuente emisora envía un paquete de datos a un solo receptor o destino. Este modo de transmisión de datos es la forma más directa de entregar un mensaje. De manera gráfica podemos observar el modo *unicast* en la figura 1.21.

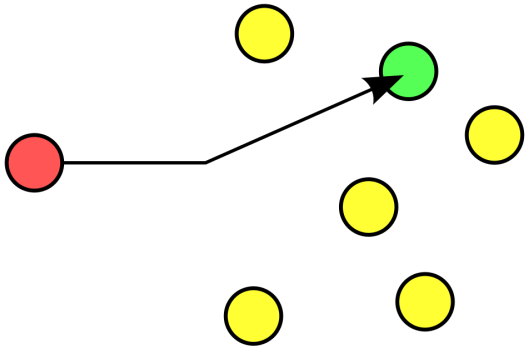


Figura 1.21: Envío de paquete en modo *unicast*.

3. **Multicast**: El protocolo de transmisión *multicast* aparece cuando una fuente emisora necesita enviar el mismo paquete de datos a un grupo o varios destinos de manera simultánea. En la figura 1.22 aparece el modo de envío *multicast* donde el emisor entregará el paquete sólo a ciertos nodos vecinos.

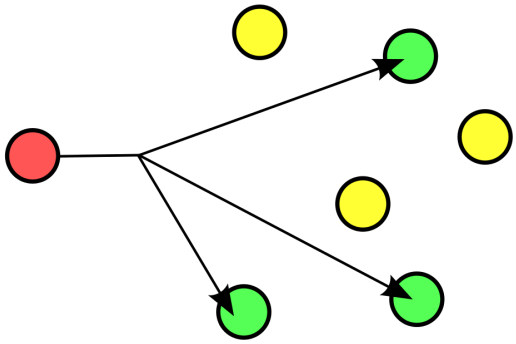


Figura 1.22: Envío de paquete en modo *multicast*.

Un caso especial de emisiones en modo *multicast* es el conocido como *geocast*, el cual es usado para enviar paquetes a un grupo de nodos situados dentro de una región geográfica específica. Véase la figura 1.23.

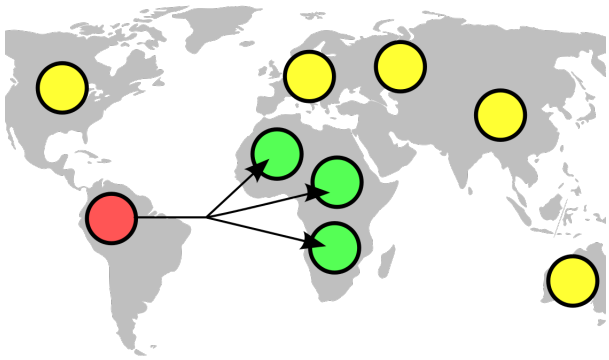


Figura 1.23: Envío de paquete en modo *geocast*.

4. **Anycast**: Este modo de transmisión se caracteriza por ser también del tipo uno-a-muchos, como los modos *broadcast* y *multicast*, ya que se tienen varios candidatos como nodos receptores; sin embargo, el paquete sólo llegará al dispositivo que sea evaluado como el más óptimo de la red, en cuanto a su topología física o lógica. Observe en 1.24 que hay un sólo destino.

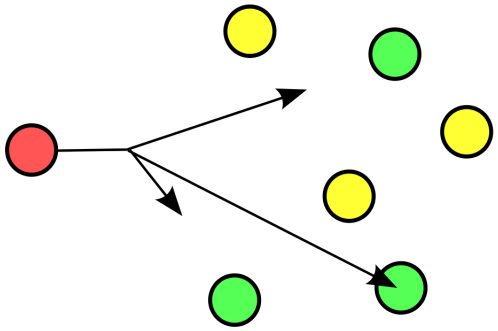


Figura 1.24: Modo de envío *anycast*.

Redes ad-hoc

Ad hoc es una locución latina que se traduce literalmente como “para esto”. Puede entenderse como: ‘Lo que se dice o se hace para un fin determinado’[Esp01, ‘ad hoc’].

En términos de redes de computadoras, una *red ad-hoc* se refiere a una conexión temporal para un propósito específico, de aquí el uso de la locución latina “para esto”.

Técnicamente, las redes ad-hoc requieren de poca o nula planificación, ya que considera a los nodos en igualdad de condiciones, los cuales regularmente utilizan la vía inalámbrica como medio de transmisión. En las redes ad-hoc no hay un único nodo central, es decir son redes sin infraestructura o administración concéntrica específica; que operan en tiempo y espacio limitados. Si una red es configurada para un periodo prolongado de tiempo, ésta pasa a considerarse como una red de área local (LAN)[Chr16, ‘ad hoc network’].

Un ejemplo de red ad-hoc simple puede apreciarse en la transferencia o envío de paquetes de manera inalámbrica entre dos dispositivos, es decir en un modelo p2p como el mostrado en la figura 1.6.

Una extensión del ejemplo anterior, ocurre cuando un grupo de personas se encuentran reunidas y usan la vía inalámbrica para comunicarse y realizar actividades colaborativas, generando así una red espontánea. Estas personas utilizan solamente la capacidad de cada dispositivo para captar las señales cercanas e intercambiar información entre ellas. Véase la figura 1.25.

Figura 1.25: Red ad-hoc.

1.2. Simuladores para redes

En esta sección se describirán algunos *softwares* utilizados para llevar a cabo simulaciones de redes. Lo que se pretende mediante la utilización de estos programas es modelar el comportamiento de una red de computadoras calculando la interacción entre los diferentes elementos que la conforman y la transmisión de datos entre ellos. Lo anterior se logra por la reproducción de observaciones que se producen en la red o por el uso de métodos analíticos como fórmulas matemáticas y procesos de captura de datos.

El comportamiento de una red, así como el funcionamiento de varias aplicaciones y/o servicios pueden ser observados mediante la ejecución de casos prueba, en los que se pueden ir modificando varios atributos del ambiente o llevar la simulación de manera controlada para evaluar la evolución de la red bajo ciertas condiciones[Fou16c].

Algunos simuladores de redes utilizados para la investigación, análisis y uso profesional son[7]:

- Traffic
- Shunra VirtualEnterprise
- GloMoSim
- cnet
- OptSim
- GTNetS
- JiST / SWANS
- OMNeT++
- QualNet Developer
- PARSEC
- NCTuns
- Performance PROPHET
- Riverbed (antes OPNET)
- NetSim
- The Network Simulator - ns

Algunos de los simuladores de la lista previa manejan la Interfaz Gráfica para el Usuario (GUI), por sus siglas en inglés (*Graphical User Interface*); mientras que otros programas lo hacen manejando la Interfaz de Línea de Comandos (CLI), de igual manera por sus siglas en inglés (*Command-Line Interface*).

La mayoría de los simuladores de red utilizan la simulación de eventos discretos, la cual funciona mediante el manejo y almacenamiento de una lista de eventos

[7] `http://people.idsia.ch/~andrea/sim/simnet.html`

pendientes, los cuales se van procesando de manera ordenada. En dicho proceso se van desencadenando eventos para su futuro procesamiento. Para entender como funciona este tipo de simulación, puede suponerse que el arribo de un paquete a un nodo, desencadenará la emisión de la confirmación de recepción por parte de dicho nodo, esto se realizará cuando llegue su turno en la lista de eventos por procesar[Fou16c].

Usos y características de los simuladores

Algunos de los usos que permiten los simuladores de redes son:

1. Permite la implementación de diseños de redes válidos para empresas, centros de datos o redes de sensores.

2. Evaluación de impacto de una red existente, lo que permite realizar modificaciones o adiciones.

3. Desarrollo e investigación en redes de computadoras.

4. Simulación de protocolos de defensa en aplicaciones militares.

Existe una amplia variedad de simuladores de red, desde los más simples hasta los más complejos. Sin embargo, el usuario en cualquiera de ellos es capaz de manipular las siguientes características:

- Modelar la topología de la red, es decir, especificar el número de nodos de la red y los enlaces entre éstos.

- Modelar el flujo de aplicaciones o tráfico entre los nodos.

- Proporcionar métricas de rendimiento como salida de la simulación.

- Visualización del flujo de un paquete.

- Rastreo de paquetes, análisis de profundidad de los eventos y facultad de depuración por pasos (*debugging*[8]).

Enseguida se describen algunos de los simuladores de redes más destacados:

[8]Proceso de identificación y remoción de errores de un *hardware* o *software* de computadora [oO16, debugging]

Riverbed

Este simulador es el sucesor de lo que antes era llamado OPNET. Esto se debe a que en el año 2012 la compañía desarrolladora del simulador OPNET fue adquirida por Riverbed Technology.

Como se comenta en su sitio web[9], Riverbed Modeler es un rápido simulador de eventos discretos que sirve para analizar y diseñar redes de comunicación y trabajo.

El simulador Riverbed comprime una suite de protocolos y tecnologías con un ambiente sofisticado de desarrollo, en el cual se pueden modelar redes y tecnologías del tipo VoIP, TCP, OSPFv3, MPLS, IPv6 entre otras. Riverbed Modeler analiza redes para comparar el impacto de diferentes diseños de tecnología en su comportamiento final, mediante la realización de pruebas y demostraciones antes de llevar a cabo la producción. Además incrementa la investigación y desarrollo en la productividad de las redes, así como los protocolos y tecnologías propiamente inalámbricas para evaluar mejoras en los modelos estándar base.

El simulador Riverbed es un programa de código privativo, por lo que para hacer uso de este *software* es necesario realizar el pago por una licencia para su operación.

NetSim

NetSim es un simulador de tipo GUI, el cual es desarrollado por la compañía Tetcos. A continuación se presenta la descripción acerca de este simulador como aparece en su sitio web [10].

El simulador NetSim es un *software* privativo líder en la simulación de redes, en el cual se puede llevar a cabo modelado y representación de simulacros de protocolos, investigación y desarrollo en el área de redes, así como aplicaciones de defensa de tipo militar. Este simulador permite analizar redes de computadoras sin demasiado poder de cómputo de una manera flexible.

Existen cuatro tipos de versiones para su distribución. Para la *Academic version* se permite la descarga gratuita, ya que está pensada para usos académicos, sin embargo las herramientas que ofrece son limitadas. Observe la figura 1.26.

[9] `http://www.riverbed.com/products/steelcentral/steelcentral-riverbed-modeler.html`
[10] `http://tetcos.com/netsim_gen.html`

NetSim Academic Version

Designed as a virtual networks lab infrastructure on a desktop PC, useful for lab experimentation and teaching at UG and PG level. Available as a single product and includes all the technologies given above.

Features	Academic Version	Standard Version
Technology Coverage Internetworks, Legacy Networks, BGP and MPLS Networks, Advanced Wireless Networks, Cellular Networks, Sensor Networks, Cognitive Radio Networks, LTE Networks	✔	✔
Performance Reporting Performance metrics available for Network and Sub-network	✔	✔
Packet Animator Used to animate the packet flow in network	✔	✔
Packet Trace and Event Trace Available in tab ordered .txt format for easy post processing	✘	✔
Protocol Library Source Codes with Documentation Protocol C source codes and appropriate protocol header files with extensive documentation.	✘	✔
Development Environment Users can write their own code, link their code to NetSim and debug using Visual Studio	✘	✔
Dynamic Metrics Allows users to plot the value of a parameter over simulation time	✘	✔

Figura 1.26: Comparativa de *NetSim Academic Version* vs. *Standard Version*

ns-3

Tal como se muestra en su página oficial, el *ns-3* es un simulador de redes de eventos discretos para sistemas de internet enfocado principalmente para investigación y uso educativo. ns-3 es un software gratuito, bajo la licencia de GNU, GPLv2 [GNU14]; y está disponible para su uso, investigación y desarrollo. [Fou15a]

El simulador ns-3 es un programa de tipo CLI, por lo que el desplegado de los elementos de red, sus enlaces, ejecución de aplicaciones, control de tráfico y análisis de rastreo, se realiza desde una *terminal*[11], donde se compilan y ejecutan los códigos previamente elaborados en un editor de textos o en un desarrollador de programas.

El trabajo efectuado en esta tesis se llevó a cabo en la versión ns-3.22, liberada en febrero de 2015[Fou15b]. Para la fecha en que se escribió este documento la

[11]También conocido como emulador de terminal, emulador de consola o shell, es un programa informático que simula el funcionamiento de una terminal de computadora en cualquier dispositivo de visualización. Funciona como intermediario para procesos de compilación, actualización o instalación de programas[Fou16b].

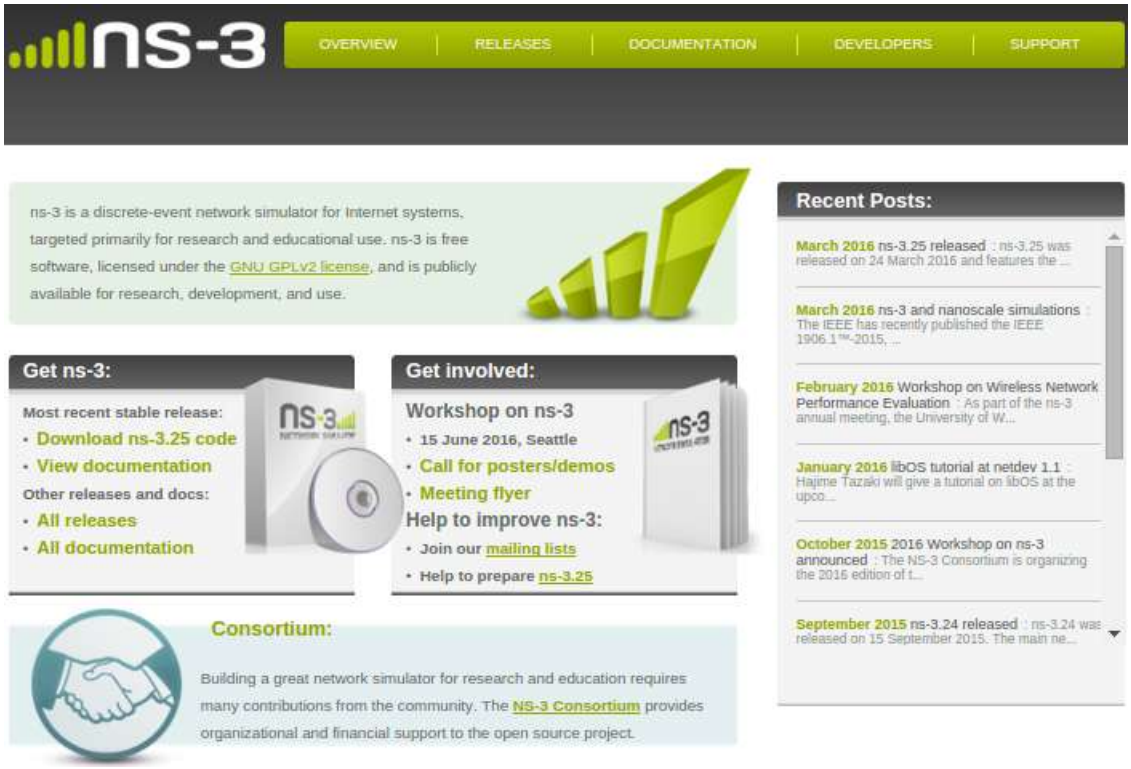


Figura 1.27: Página oficial del simulador ns-3

versión estable más reciente era la versión 25.

En el apéndice A se muestra una guía de la instalación de ns-3.22 para sistemas Linux. El sistema operativo del equipo para realizar las simulaciones fue Ubuntu 12.04.5 LTS, el cual también es un software de distribución gratuita.

El simulador ns-3 utiliza como base para correr *scripts* al lenguaje de programación orientado a objectos: C++.

Capítulo 2

Algoritmos de ruteo para redes ad-hoc

Una vez definidas qué son las redes de computadoras y considerando que la materia prima de este trabajo son las de tipo ad-hoc, en este capítulo se abordará el concepto de algoritmos de enrutamiento o encaminamiento. Para esto, en la sección 1 se extenderá el concepto de redes ad-hoc y se dará la definición de los algoritmos usados para el intercambio de información. Posteriormente se puntualizará sobre algoritmos que funcionan basados en la posición de los nodos o dispositivos. En particular, se describirá el concepto de enrutamiento voraz y cómo se aplica en los algoritmos conocidos como: Encaminamiento Voraz Perimétrico Local y Árbol de Etiquetas.

2.1. Comunicación en redes ad-hoc

Ya que la forma natural en que se comunican las redes ad-hoc es mediante la vía inalámbrica, se debe considerar el *rango de transmisión* de los dispositivos, denotado por R; que se entiende como la máxima *distancia euclidiana*[1] a la que llega el alcance de la señal del dispositivo. Es decir, en la representación gráfica de la red, dos nodos u, v estarán adyacentes si y sólo si la distancia euclidiana de separación entre ellos es menor al rango de transmisión de cada uno de ellos.

El rango de transmisión puede tener alcances distintos dependiendo del tipo de dispositivo, de la intensidad de la señal o potencia.

[1]Longitud del segmento de recta que hay entre dos puntos del espacio. Aquella que se puede cuantificar con un patrón de medida como el metro[Esp01, 'distancia'].

En los nodos de la red ad-hoc también influye el tipo de estándar que utilice la red inalámbrica. Esto está relacionado con los niveles de arquitectura para el uso de la tecnología Wi-Fi, establecidos por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, por sus siglas en inglés). En la actualidad, los estándares de arquitectura que sobresalen para la comunicación vía Wi-Fi son *IEEE 802.11 b/g/n*.

Sin embargo, como ya se dijo, en las redes ad-hoc se consideran los nodos en igualdad de condiciones, por lo que el rango de transmisión para cada uno de ellos se considera del mismo alcance. En la figura 2.1 se aprecian que los dispositivos se encuentran dentro del rango de alcance de manera simultánea.

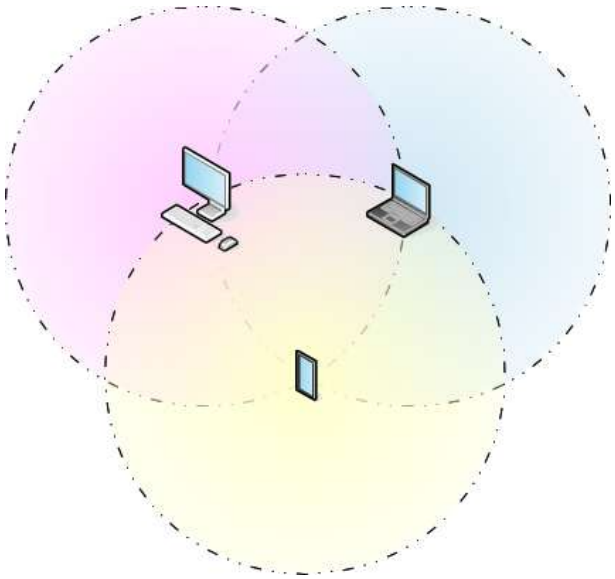


Figura 2.1: Nodos conexos en red ad-hoc.

Uno de los problemas centrales en el estudio e investigación de las redes inalámbricas ad-hoc es tratar de prever la variedad de situaciones posibles que pueden ocurrir.

Como primer caso que se toma de variación en las redes ad-hoc es el análisis sobre la movilidad de los nodos, ya que los dispositivos inalámbricos usados cotidianamente, se pueden desplazar dentro del plano donde se despliega la red. Este tipo de redes reciben el nombre de *red MANET*, por sus siglas en inglés, *Mobile Ad-Hoc Network*. Cuando la velocidad de los nodos es considerable, como en el caso de algún medio de transporte equipado para ser considerado como nodo de una red; se dice que se trata de una *VANET* (*Vehicular Ad-Hoc Network*).

Como consecuencia de la variable de movilidad de los nodos, éstos pueden estarse conectando y desconectando de la red, lo que modifica la topología de la red. Anuado a esto, para establecer la comunicación entre dos nodos u, v de una red los cuáles no se encuentren en su respectivo rango de transmisión, se recurre a enviar el mensaje a través de otros nodos, provocando varios saltos del mensaje para poder llegar a su destino. Esta comunicación se conoce como envío *multisalto*.

Otra cuestión importante es cómo se descubre la secuencia que debe elegirse en el multisalto para que el mensaje llegue de manera óptima, realizando la menor cantidad de reenvíos y se realice en el menor tiempo posible. Para descubrir estas rutas se utilizan los *algoritmos de encaminamiento*.

2.2. Algoritmos de enrutamiento

En redes de computadoras, el *encaminamiento, enrutamiento* o *ruteo* es el proceso que consiste en buscar en el conjunto de caminos posibles para dos nodos u, v de la red; el mejor de ellos o el que cumpla con ciertas condiciones. Esta búsqueda surge, ya que en una red con demasiados enlaces entre los nodos se necesita resolver el problema de cómo elegir la mejor secuencia para el recorrido.

Tal como describe [TV05, cap. 2], los algoritmos tradicionales para enrutar redes ad-hoc se clasifican en: *proactivos* y *reactivos*.

Se considera un algoritmo *proactivo* como aquél en el que se tiene toda la información sobre la topología de la red y se conocen las rutas de todas las fuentes a todos los destinos. Este tipo de protocolos son similares a los utilizados en redes alambradas.

- **Ventajas**: Las rutas se tienen rápidamente disponibles cuando un nodo desea enviar un mensaje a otro.
- **Desventajas**: Requiere memoria proporcional al tamaño de la red. Además, existe una gran cantidad de *mensajes de control*[2] para ordenar la información.

Dos protocolos respresentativos de encaminamiento proactivo son: DSDV, *Destination Sequenced Distance Vector* y OLSR, *Optimized Link State Routing*, ambos por sus siglas en inglés.

[2]Mensajes que requieren ser enviados en la red, para mantener en orden la información[TV05, Cap. 2, Sec. 1].

Un algoritmo *reactivo* o también conocido como 'bajo demanda' es aquél en el que se pretende minimizar el uso de ancho de banda para la funciones de mensajes de control y se quiere dejar en la red mayor afluencia para los datos. Estos algoritmos se caracterizan por descubrir una ruta entre un origen y un destino cuando solamente se quiera establecer comunicación entre ellos.

- **Ventajas**: Almacena información de las rutas que están usando en cierto momento. Así mismo, aparecen menos mensajes para el encaminamiento de información.
- **Desventajas**: Si los nodos no tienen ruta, debe ser descubierta previamente, esto ocasiona un retraso antes de enviar el primer paquete, después de este proceso los paquetes se envían normalmente.

En los algoritmos reactivos, periódicamente los nodos emiten mensajes de tipo *broadcast*, esto con la finalidad de mantener actualizada la información acerca de si los nodos están conectados a la red o si se ha presentado una falla. Un tipo de estos mensajes de control son los llamados *paquetes Hello*, cuya función es informar que cierto nodo sigue activo, además con estos paquetes se actualizan algunos temporalizadores asociados al nodo o en su defecto pueden deshabilitar entradas a la red [TV05, Cap. 2, Sec. 4.2].

Entre los algoritmos destacados que se encuentran clasificados como reactivos encontramos: AODV *Ad-hoc On Demand Distance Vector* y DSR *Dynamic Source Routing*, ambas abreviaturas en el idioma inglés.

Dentro de los algoritmos tradicionales de enrutamiento se puede agregar un tercer tipo de clasificación, éstos se conocen como algoritmos *híbridos*, los cuales son una combinación de los proactivos y reactivos, cuya finalidad es minimizar el número de desventajas que se puedan tener a la hora de enrutar. Un algoritmo de tipo híbrido que destaca es ZRP *Zone Routing Protocol*.

2.3. Encaminamiento basado en la posición

Gracias al avance que ha habido en cuestión de servicios de ubicación, en la actualidad diversos dispositivos pueden contar con hadware que permiten su localización posicional.

Una de las herramientas que predomina en el servicio de localización es el *GPS* (abreviatura de *Global Positioning System*). El Sistema de Posicionamiento Global es un sistema desarrollado por el departamento de defensa de Estados de

Unidos de América que funciona satelitalmente usando como precepto la trilateración inversa; proceso mediante el cual se calcula la distancia respecto a ciertos satélites que son visibles desde cierto punto. Véase la figura 2.2. Con lo anterior, se puede proveer fácilmente de información precisa respecto al posicionamiento de un objecto en casi cualquier lugar de la Tierra.

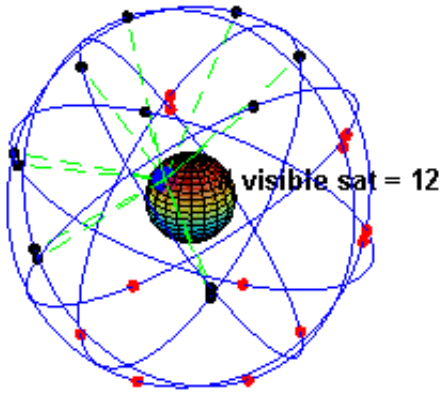


Figura 2.2: Satélites visibles para cierto punto de la Tierra.

Esta tecnología, nacida en la segunda mitad del siglo XX bajo programas militares y mejorada para el uso civil para inicios del siglo XXI, se ha proliferado en gran medida, al grado que la integración de un sistema receptor GPS, está disponible en diversos teléfonos móviles, en bases de distintos centros de datos o estaciones dotadas con sensores. Una de las grandes ventajas de este sistema es que brinda una precisión sobre la ubicación geográfica con un rango de fallo de entre 50 a 125 metros[CCL03].

Continuando con la revisión de [TV05] encontramos las siguientes características: Los algoritmos de encaminamiento basados en la posición, aparecen cuando se tienen nodos que cuentan con un servicio de localización, por ser el más accesible en estos días, podremos suponer que se trata de nodos equipados con un GPS. De esta manera, para cada nodo que sea parte de la red, se puede saber la posición de éste en una proyección geográfica. Este parámetro se va actualizando en la medida que el dispositivo vaya teniendo movilidad.

Como precepto importante, el ruteo por posición dispone de un *servicio de localización*, cuya función primordial es poder encontrar la posición del dispositivo que se tiene como destino. Sin este servicio no se podría realizar este tipo de encaminamiento.

Gracias a la ventaja de poder conocer la posición del nodo que se tiene como destino y la posición precisa del nodo emisor, se pueden tomar decisiones acerca de cómo debe irse hilando la secuencia del enrutamiento para entregar el paquete al destinatario de manera eficiente. Para tener una actualización sobre la posición de los nodos, éstos de manera periódica envían señales con el respectivo identificador del nodo y sus coordenadas geográficas.

2.4. Encaminamiento tradicional voraz

Un tipo de algoritmos de ruteo basados en la posición son los llamados *algoritmos voraces*. Algunas características de ellos son:

- Al ser conocida la posición del destino, la toma de decisiones de la ruta se hace sobre dicho parámetro.

- Se utiliza el concepto de progreso[TV05, Cap.3,Sec.2]: Si se tiene un nodo *s* deseando enviar un paquete, el progreso de un nodo vecino *n* es la proyección sobre la recta que forma *s* al destino. Si el nodo *n* proyecta en la dirección de avance, entonces el nodo tiene *avance* o *incremento positivo*, caso contrario se dice que tiene *retroceso* o *incremento negativo*. En la figura 2.3 se muestran las proyecciones de los vecinos del nodo *s*.

- Cada nodo tiene al alcance la información de la posición del destino. Esto se puede interpretar como tener acceso al servicio de localización.

  Algunos protocolos usados para la localización del destino son:

  1. GLS - *(Grid Location Service)*.
  2. RLS - *(Reactive Location Service)*.
  3. HLS - *(Hierchical Location Service)*.

El algoritmo más simple pero efectivo en los protocolos de encaminamiento basados en la posición es GPSR *(Greedy Perimeter Stateless Routing)*, el cual se puede traducir como Encaminamiento Voraz Perimétrico Local. Como comentario a esta traducción los autores de este algoritmo mencionan que *stateless* se refiere a lo pequeño, a que el perímetro se hace puramente local[KK00, p. 245].

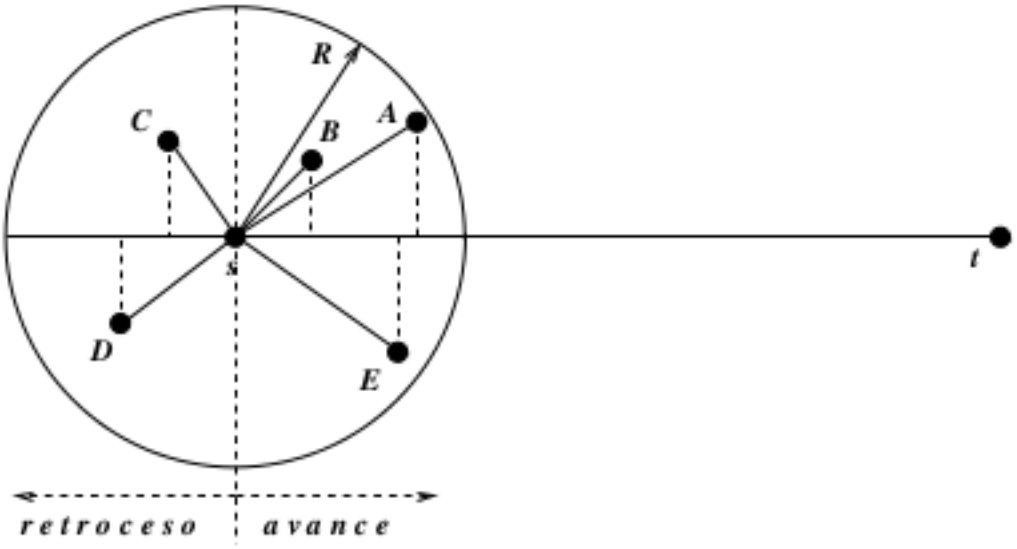


Figura 2.3: Progreso de los nodos vecinos de s.

2.5. Encaminamiento Voraz Perimétrico Local (GPSR)

Sobre el funcionamiento de este algoritmo creado por [KK00] encontramos el resumen de sus características en [FCoVa12]:

- En este algoritmo el ruteo solución se obtiene mediante el reenvío de paquetes en forma voraz.
- Se elige al vecino geográficamente más cercano para enviarle el paquete y a partir de este nodo buscar hacia dónde se debe efectuar el siguiente salto. Es decir, aquél con la menor distancia euclidiana, representada con la notación $d(u, v)$ para la distancia entre los nodos u, v.
- La distancia entre las posiciones se puede calcular mediante mensajes tipo *Hello*.
- Para que no se estén generando mensajes *Hello* de manera simultánea, se utiliza un retrasador de señal (*random jitter*) para evitar colisiones.
- Existen casos en los que se puede caer en una situación de *máximo local* u *óptimo local*, este caso ocurre cuando el paquete se encuentra más cerca

de su destino en comparación de sus demás vecinos, pero no es alcanzable mediante un salto. En este momento se habla de que entrará en vigor una estrategia de recuperación.

- La estrategia de recuperación usada es la regla de la mano derecha (*right-hand rule*). Esta estrategia es conocida como el modo perímetro (*Perimeter Mode*), en este caso si un nodo recibe un paquete por cierta arista y se activa el modo de recuperación, el paquete se reenvía a la siguiente arista en sentido de las manecillas del reloj a partir de la arista por donde llegó el paquete. Para esto se utiliza una herramienta de cálculo de ángulos para poder efectuar el giro en dicho sentido; este reenvío se hará recursivamente hasta que se pueda cambiar al modo de reenvío voraz nuevamente.

Pseudo-código del Encaminamiento Voraz Perimétrico Local

Para describir el pseudo-código del GPSR[FCoVa12, p. 354], usaremos la siguiente notación.

- R es un nodo que recibe un paquete p para un destino D.
- N es el conjunto de vecinos a distancia menor o igual que 1 del nodo R.[3]
- n es un nodo de N que se usa para enviar el paquete.
- D representa el destino del paquete.
- $d(n_1, n_2)$ simboliza la distancia euclidiana entre nodos.

Listado 2.1: Pseudo-código GPSR

```
if ∃ n ∈ N | d(n,D) ≤ d(R,D) then
        {Reenvío Voraz}
        Se escoge n ∈ N | mín d(N,D)
        Se reenvía p → n
        return;
else
        {Máximo Local, se usa regla de la mano derecha}
        Se elige n ∈ N | cumpla con regla de la mano derecha.
        Se reenvía p → n
        return;
end if
```

[3] Sin pérdida de generalidad, ya que todos los nodos se consideran en las mismas condiciones, el rango de transmisión es tomado como si fuera la unidad.

2.6. Árbol de etiquetas

Los autores Chávez, Mitton y Tejeda describen en [CMT07] un algoritmo de encaminamiento basado en la posición que se puede implementar en redes ad-hoc que tengan nula o poca movilidad.

La idea básica de este algoritmo es usar información a nivel local para poder elegir el siguiente salto hacia donde debe dirigirse el paquete. Algunas consideraciones básicas que hacen en este documento es que los nodos tienen un identificador único y además se encuentran dispersos uniformemente.

Además, en su trabajo, los autores introducen un concepto que denominan *PositionTree*. Para definir esta estructura se considera que los nodos pueden almacenar como información una etiqueta, que se entiende como un parámetro de tipo cadena, inicialmente vacío.

Dado que se trata de un algoritmo basado en la posición, la etiqueta se podrá conocer de la misma manera en que se accede a la ubicación de los nodos.

En [CMT07], la estructura *PositionTree* es utilizada para encontrar rutas de encaminamiento en una red de N nodos. En nuestro caso, dicha estructura se definirá como *árbol de etiquetas* respetando el proceso de construcción original:

- Primeramente se elige un nodo aleatoriamente del conjunto que conforma la red. Este nodo recibe el nombre de raíz y se le asignará una etiqueta jerárquica. En este momento el único nodo que tiene un valor en su parámetro cadena es la raíz.

- Para realizar el siguiente paso, se envía un mensaje de tipo *broadcast* para descubrir las adyacencias. Los nodos vecinos responderán enviando una solicitud de etiquetado (*TagRequest*).

- Enseguida, de manera aleatoria o siguiendo una heurística ordenada, se va añadiendo a los nodos adyacentes (los que contestaron), su correspondiente etiqueta tomando como base la etiqueta del padre y agregando un número extra consecutivo ascendente. Este proceso inicia con la raíz etiquetando a sus nodos vecinos, que pasarán a ser sus hijos en el árbol.

- Una vez que todos los nodos adyacentes han sido etiquetados se envía un mensaje al padre para informar que ha culminado el etiquetado de sus hijos.

- Después de que un nodo hijo es etiquetado el proceso continuará pasando un hijo a convertirse en padre y heredando su etiqueta como base para los

nodos adyacentes que aún no reciben etiqueta. Al final, el padre recibirá la notificación de que se ha concluido el proceso para sus vecinos.

- Las etiquetas que se van colocando son construidas agregando el número extra a la derecha de la etiqueta del padre. De esta manera, el camino hacia la raíz lo definirá la jerarquía de la etiqueta del nodo y el tamaño de dicho camino será el número de dígitos que tenga la etiqueta después de la raíz.

- El proceso concluye cuando todos los nodos de la red han sido etiquetados.

Una vez que finaliza la construcción del árbol de etiquetas, como se ha comentado antes, por ser un árbol, existe un único camino para cada par de vértices que formen parte de la red. Este proceso es usado por los autores para encontrar las rutas de encaminamiento.

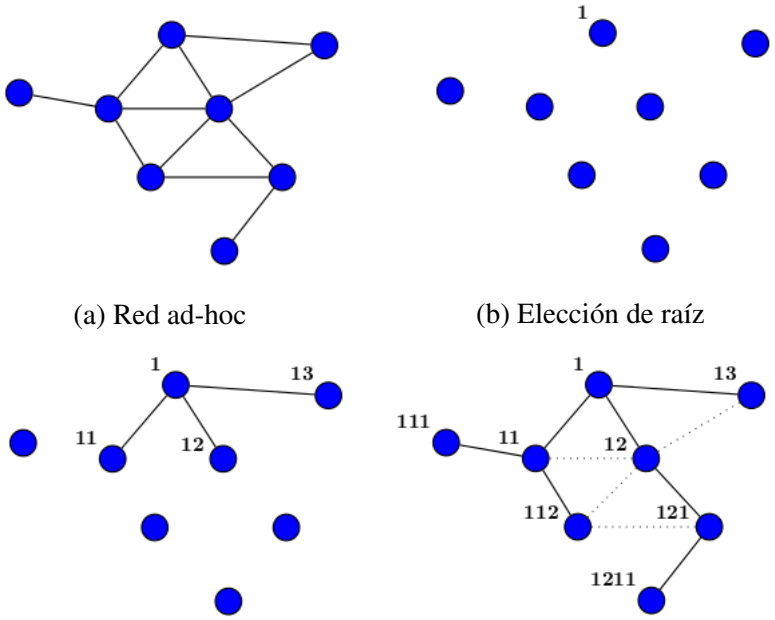
Si la estructura construida resulta ser un *árbol balanceado*[4], entonces se dice que el *tamaño del árbol*[5] de éste es de $O(\log N)$. Por lo general, se puede obtener un árbol balanceado si la raíz se encuentra en una región central de la red y no en la periferia.

En la figura 2.4 se puede observar el proceso de construcción de un árbol de etiquetas para una red ad-hoc formada por 8 nodos. En la figura 2.4a se muestra la red original con sus respectivas conexiones entre nodos. En 2.4b se aprecia como se elige el nodo raíz y se le asigna su etiqueta, la cual se marca como 1. En 2.4c se observa como al realizar el proceso *broadcast* y después de que se recibe la respuesta por parte de los vecinos de la raíz, se asigna de forma ordenada el etiquetado a dichos nodos, éstos quedan marcados como 11, 12 y 13. Por último en 2.4d se puede apreciar cómo al realizar el proceso recursivo de etiquetado y finalizarlo, se llega a formar un árbol. Con líneas punteadas se marcan las conexiones existentes de la red que no fueron tomadas en cuenta para formar el árbol. En este caso se obtiene un árbol de profundidad 3. Esto se debe porque la etiqueta más alejada de la raíz es 1211 y la distancia hacia la raíz a partir del nodo con esta etiqueta es de 3 saltos.

Para conocer la distancia que hay entre un nodo y la raíz, basta con tomar el tamaño de la etiqueta del nodo y restarle el tamaño de la etiqueta que tiene la raíz.

[4] Aunque por lo general esta definición aplica para árboles binarios, se dice que un árbol es balanceado cuando los subárboles que se van generando tienen el mismo número de componentes que el componente inicial.[Sol]

[5] El *tamaño, profundidad* o *altura* de un árbol se define como la excentricidad de la raíz. Como observación se tiene que la raíz puede no fungir como el centro de la gráfica.



(a) Red ad-hoc (b) Elección de raíz

(c) Etiquetado de vecinos (d) Árbol de etiquetas

Figura 2.4: Construcción de árbol de etiquetas

En el caso mostrado en la figura 2.4, el tamaño de la etiqueta que tiene la raíz es 1. De esta manera para conocer la distancia de los nodos marcados como 11, 12 y 13, los cuales llevan una etiqueta de longitud 2 y al restarles el tamaño de la raíz, se tiene que $2 - 1 = 1$, es la distancia que hay entre estos nodos y la raíz.

Sucesivamente, la distancia es de 2 saltos entre la raíz y los nodos 111, 112 y 121, ya que $3 - 1 = 2$. De la misma manera se ve que la distancia entre el nodo etiquetado como 1211 y la raíz es $4 - 1 = 3$.

Envío de paquetes y encaminamiento

Para el envío de paquetes mediante el algoritmo árbol de etiquetas, la manera en que se da la búsqueda de rutas en la estructura construida consiste en la comparación de etiquetas. Para esto se usan las asignadas al nodo emisor y al nodo destino.

Supongamos el caso de un nodo A que quiere enviar un paquete a un nodo B. El paquete debe realizar un ascenso en el árbol hasta encontrar el *ancestro común más coincidente* (a.c.m.c.) de A y B, para luego bajar hasta encontrar al

nodo B. Todas las decisiones de reenvío son hechas de manera local sin necesidad de inundar la red.

Ejemplo 20. Para fijar ideas, suponga que A está etiquetado como 1167895 y B está etiquetado como 116774232, el *prefijo común más largo* es 1167 el cual corresponde al *ancestro común más coincidente* de A y B. Por medio del proceso *broadcast* de la fuente hacia los nodos vecinos, se encuentra sólo un nodo etiquetado como 116789 y es a éste al cual se le reenviará el paquete. Este proceso se repite con la secuencia $A = 1167895, 116789, ..., 1167$ (siendo el último nodo el ancestro común más coincidente) y a partir de la última posición se reenvía a los nodos con etiquetas únicas $11677, 116774, ..., 1167774232 = B$ hasta alcanzar el nodo destino.

Observación 21. El proceso tanto de ascenso/descenso puede ser vacío si el emisor/destino es el ancestro común más coincidente del destino/emisor respectivamente.

En la figura 2.4d se aprecia que si el nodo con la etiqueta 1211 quiere enviar un paquete al nodo 111, la secuencia del camino sería $1211, 121, 12, 1, 11, 111$. Aquí puede observarse que cuando el nodo 121 procede a reenviar el mensaje, los nodos $1211, 112$ y 12 recibirán la petición pero sólo el nodo 12 lo reenviará hasta llegar al ancestro común más largo entre 1211 y 111, el cúal es la raíz marcada con la etiqueta 1.

La estructura del árbol de etiquetas será retomada más adelante para alcanzar los propósitos que persigue este trabajo de tesis.

Capítulo 3

Diseño y desarrollo del árbol de etiquetas

El objetivo principal que se persigue en este trabajo de tesis es poder implementar a través del simulador ns-3 el algoritmo de encaminamiento árbol de etiquetas, presentado en [CMT07]. Haciendo una revisión de este algoritmo encontramos que su funcionamiento es muy parecido al trabajo desarrollo en [KK00]. En ambos algoritmos se emplea el encaminamiento voraz tradicional pero al caer en un máximo local se emplean estrategias de recuperación distintas.

Para llevar a cabo la implementación en ns-3, se usará como base el trabajo de [FCoVa12]. Los autores de este artículo implementan el algoritmo GPSR en la versión ns-3.12; por esta razón, primero se llevarán a cabo algunas modificaciones pertinentes para adaptar la implementación a una versión más reciente del simulador.

Una vez adaptados y probados los ejemplos que se anexan en la implementación de Fonseca, Camões y Vazão, se procede a modificar el algoritmo GPSR. Cuando se llegue al punto donde se alcance un máximo local, el sistema de recuperación que se utilice cambiará de la estrategia *right-hand rule* al uso de la estructura que se emplea en el árbol de etiquetas, con la finalidad de que el proceso de recuperación se realice más ágilmente.

Para hacer uso de la estructura mencionada será necesaria la añadidura de un módulo auxiliar en el simulador. Es por eso que primeramente se revisarán algunas de las bases del ns-3, para conocer la forma de operación de este *software*.

3.1. Revisión conceptual del simulador ns-3

Como se comentó en la sección 1.2, para visualizar cómo se realiza la instalación de la versión del simulador ns-3.22, se puede consultar el apéndice A de este trabajo. A partir de este momento, supondremos que se tiene instalada dicha versión y que es posible ejecutar los *scripts* usados como ejemplos en el tutorial.

Para desarrollar las simulaciones, ns-3 ordena los códigos elementales como modelos, éstos al ser de licencia gratuita pueden ser editados a gusto por el usuario. Los modelos a su vez se agrupan en módulos que representan en conjunto, partes fundamentales de los elementos que componen una red. Los modelos que corresponden a los elementos base de una red según [Nat15b, Cap.4] son:

- **Nodo** (*Node*): Esta definición equivale a la mostrada en la sección 1.1, la cual afirma que un nodo se entiende como una computadora a la que se le puede agregar una funcionalidad y que puede conectarse a una red.
- **Aplicación**: Se puede entender como un programa que adquiere y usa los recursos controlados por el *software* del sistema para alcanzar o completar una meta.
- **Canal** (*Channel*): Comunicación básica de una subred abstracta. Esto es lo análogo a los medios de transmisión descritos en la sección 1.1. Algunos tipos de canal son:
  - *CsmaChannel*: Su significado es *Carrier Sense Multiple Access*. Corresponde al modo empleado cuando se hacen transmisiones de tipo múltiple como *multicast*. Esto para evitar que haya colisiones de paquetes entre emisores y receptores de la red.
  - *PointToPointChannel*: Este modo de comunicación es el que establece en el modelo p2p. Representa la comunicación sencilla entre dos dispositivos. Es de tipo *unicast*.
  - *WifiChannel*: Este canal es utilizado cuando se despliegan elementos en una red que se están comunicando mediante la vía inalámbrica.
- **Dispositivo de Red** (*NetDevice*): Equivale al concepto descrito como Tarjeta de Interface de Red, en la sección 1.1. Así, un dispositivo de red se interpreta como el *hardware* que permite habilitar a los dispositivos la funcionalidad de emitir y recibir señales. En el simulador se hace énfasis a que este dispositivo corresponde a los controladores nombrados como [eth0] o

[wlan0] para conexiones cableadas o inalámbricas, respectivamente, en sistemas Unix y Linux.

En el simulador ns-3 un dispositivo de red es instalado sobre un nodo para habilitar la comunicación con otros nodos, usando como medio los diferentes tipos de canales. Lo podríamos representar gráficamente como lo expresa la figura 3.1.

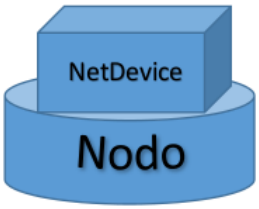


Figura 3.1: Instalación de un *NetDevice* sobre un nodo

Para explicar el funcionamiento del desplegado de elementos de una red a través del ns-3, se describe detalladamente la figura 3.2.

Primeramente, nótese que podríamos separar en dos capas la ilustración. La primer capa que se encuentra a manera de una caja contiene sólo a los nodos, que están representados por cilindros. Esta primer caja es una estructura nombrada contenedor de nodos (*NodeContainer*).

La segunda capa que se encuentra sobrepuesta al contenedor de nodos, recibe el nombre de contenedor de dispositivos de red (*NetDeviceContainer*). En este espacio se encuentran distribuidos los dispositivos de red, sin embargo, se aprecia que el tamaño de este contenedor es similar al contenedor de nodos, ya que como se mencionó, los dispositivos de red se instalan sobre los nodos.

Como argumento importante hay que mencionar que las capas contenedoras de nodos y dispositivos de red pueden tener intersecciones con otros contenedores.

Después de desplegar los nodos y dispositivos de red, sobre la capa contenedora de nodos se instala la pila de internet (*InternetStackHelper*), la cual se puede apreciar como una niebla gris que se encuentra al interior de las cajas contenedoras. Al agregar esta pila, se puede entender que se da la habilitación del conjunto de protocolos y modelos de comunicación para las redes de computadoras.

Para poder establecer comunicación entre los distintos nodos, es necesario asignar direcciones IP (*InternetProtocol*) sobre los dispositivos de red. De esta manera, se tendrá un identificador para los emisores y receptores en los enlaces que se establezcan.

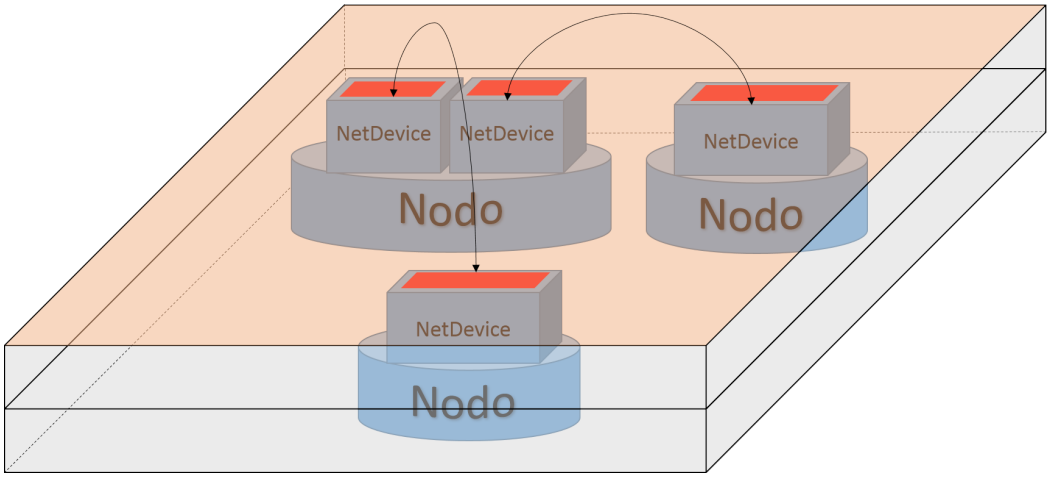


Figura 3.2: Montaje de una red en ns-3

Para que pueda existir el flujo de paquetes a través de la red se debe instalar una última capa contenedora. Esta capa contenedora se aprecia como una tapa que se coloca sobre el contenedor de dispositivos de red; en la figura 3.2, se visualiza como una capa naranja. En esta capa se pueden manejar varios tipos de protocolos. Para esta figura, supongamos que se elige el protocolo Ipv4 entonces su contenedor (la tapa naranja) sería del tipo *Ipv4AddressHelper*.

Con lo anterior, quedan habilitados los canales de comunicación que usarán los protocolos asignados. En la figura, dichos canales se representan mediante las flechas que unen los diversos dispositivos de red y es a través de ellos que fluirá la información por medio de paquetes.

Sin embargo, en el simulador ns-3, el tráfico de paquetes se ha dicho que se da sobre aplicaciones; es por eso que se terminan por instalar sobre los nodos, con toda su estructura que se les ha colocado, las aplicaciones que fungen como *softwares* que simulan servidores o clientes. En la figura 3.2 se visualizan como los recuadros rojos.

En el apéndice B, se agregan los códigos desplegados en la documentación del simulador ns-3 nombrados **`first.cc`** y **`second.cc`** con su respectiva representación gráfica descrita en esta sección.

3.2. Construcción de módulos

La ventaja de utilizar el simulador de código abierto ns-3, es que permite crear nuevas estructuras respetando los modelos de los elementos de red previamente establecidos. La manera en que se crean módulos se encuentra descrita en el manual[Nat15a, p.136]. A continuación se resumirá un poco dicho proceso de construcción:

Se dice que se genera un módulo del ns-3, al tener un conjunto de clases, ejemplos, casos de prueba (*tests*); de uno o varios modelos. Esta agrupación se realiza para que los módulos puedan ser compartidos y usados por otros usuarios. Todos los módulos del simulador se pueden encontrar en la carpeta **`ns-3.22/src`**. En esta ubicación se encuentran agrupados en subcarpetas los diversos modelos. El nombre de la carpeta indica a grandes rasgos qué tipo de modelos puede contener. A continuación se presentan los módulos por defecto que tiene la carpeta **`src`**:

- antenna
- aodv
- applications
- bridge
- brite
- buildings
- click
- config-store
- core
- csma
- csma-layout
- dsdv
- dsr
- energy
- fd-net-device
- flow-monitor
- internet
- lr-wpan
- lte
- mesh
- mobility
- mpi
- netanim
- network
- nix-vector-routing
- olsr
- openflow
- point-to-point
- point-to-point-layout
- propagation
- sixlowpan
- spectrum
- stats
- tap-bridge
- test
- topology-read
- uan
- virtual-net-device
- visualizer
- wave
- wifi
- wimax

Al interior de cada módulo existen subcarpetas, donde se encuentran los modelos, auxiliares(*helpers*), casos de prueba(*tests*) y ejemplos organizados bajo la siguiente estructura:

```
module-name
  |
  +--bindings
  +--doc
  +--examples
  |    |
  |    |__wscript
  +--helper
  +--model
  +--test
  |    |
  |    |__examples-to-run.py
  |__wscript
```

La anterior estructura de directorios es la organización esquemática estándar de cada módulo. Esto se conoce como el esqueleto de los módulos.

Afortunadamente, la creación de un nuevo módulo se puede realizar mediante el *script* auxiliar **create-module.py**, el cual viene integrado en el simulador ns-3. La secuencia de comandos si quisiéramos crear un módulo titulado **nuevo-modulo** sería:

- Ubicados en el directorio contenedor de todos los módulos **src**:

```
$ ./create-module.py nuevo-modulo
```

- Enseguida se ingresa a la carpeta que recién se crea con el comando **cd**:

```
$ cd nuevo-modulo
$ ls
doc examples helper model test wscript
```

Con este primer paso se crean esqueletos iniciales para los archivos *wscript*, *.h*, *.cc* y *.rst* que el simulador requiere como base en el módulo. A continuación podemos visualizar como quedaría el esqueleto del **nuevo-modulo**:

```
nuevo-modulo
  |
  +--doc
  |    |
  |    |__nuevo-modulo.rst
  +--examples
  |    |
  |    |--nuevo-modulo-example.cc
  |    |__wscript
  +--helper
  |    |
  |    |--nuevo-modulo-helper.cc
  |    |__nuevo-modulo-helper.h
  +--model
  |    |
  |    |--nuevo-modulo.cc
  |    |__nuevo-modulo.h
  +--test
  |    |
  |    |__nuevo-modulo-test-suite.cc
  |__wscript
```

El primer archivo que se puede revisar para ver cómo ha quedado su definición es **wscript** que aparece como archivo en la carpeta **nuevo-modulo**.

Dado que la mayoría de los módulos toman como base el módulo **core**, este valor aparece por defecto en su definición, sin embargo es un parámetro editable. Como ejemplo se muestra en [Nat15a, p. 138] la sustitución con otros modelos como: **internet**, **mobility**, **aodv**.

Por otro lado, los archivos modelo con la definición primaria del nuevo módulo generado corresponden a los archivos que se encuentran en la carpeta **model**, los cuales llevan por nombre: **nuevo-modulo.cc** y **nuevo-modulo.h**. Es precisamente en este par de archivos donde se pueden establecer los campos, métodos, estructuras y archivos cabecera requeridos para los fines que se busquen con el módulo.

El método auxiliar de ns-3 usado para crear módulos, permite generar archivos auxiliares en la carpeta **helper**. Si para el manejo del módulo que se construye no son necesarios los archivos de esta carpeta, en ella sólo quedarán los esqueletos generados por defecto.

También se pueden colocar en la carpeta **examples** algunos códigos que ilustren y hagan referencia a la manera en que se opera, usa y maneja el módulo que se añade.

De igual manera, se crean archivos esqueletos, sobre los cuales se pueden agregar *scripts* para ilustrar simulaciones donde se emplee el nuevo modelo. Estos casos de prueba se almacen en la carpeta **test**. A manera de recomendación, el manual menciona que es útil agregar casos de prueba, con la finalidad de que sean revisados y aceptados en la documentación de ns-3 como parte de una contribución.

La manera en que se realiza la configuración y construcción final del módulo se muestra en el listado 3.1. Este paso debe ej ecutarse cada que se realiza una modificación o reconfiguración en alguna de las características de un módulo y se deben ejecutar en el directorio **ns-3.22**.

Listado 3.1: Comandos de compilación y configuración en ns-3

```
$ ./waf configure --enable-examples --enable-tests
$ ./waf build
$ ./test.py
```

Después de realizar la operación anterior, nuestro módulo creado aparece en la carpeta **src** y es anexado a los módulos existentes por defecto, descritos en la página 45. De manera inmediata es posible hacer uso del nuevo módulo en los *scripts* y ejecutar las simulaciones que se deseen con ns-3.

Módulo etiqueta

Antes de trabajar con el algoritmo GPSR de [KK00] y realizar modificaciones al trabajo de [FCoVa12] para su implementación en la versión ns-3.22, se explicará el concepto del módulo **etiqueta** que se agregó al simulador ns-3.

Como se ha descrito en la sección 2.3, los algoritmos basados en la posición requieren de una tecnología para su funcionamiento como el GPS; con el cual se puede conocer la ubicación de un nodo sobre la superficie terreste.

En el *software* ns-3, no se cuenta con un módulo para simular la tecnología GPS como tal. Sin embargo, existe un módulo que permite la localización de un nodo en el espacio para aquellas simulaciones que requieran conocer su ubicación; este módulo implementado en ns-3 se llama **location-service**, el cual se detalla en la sección 3.3.

La finalidad del módulo **etiqueta** es proporcionar una estructura en el simulador que funja como una extensión de los nodos y permita añadir a éstos una

etiqueta que los identifique.

Dicho módulo cuenta con tres campos: Etiqueta (*Tag*), Secuencia (*Sequence*) e Hijos (*Children*). Por otro lado, la clase principal del módulo es **NodeTag**.

El listado 3.2, tiene los comandos para crear el módulo **etiqueta**. Para su ejecución se requiere que el directorio de trabajo sea **ns-3.22/src**.

Listado 3.2: Comandos para la creación del módulo **etiqueta**

```
$ ./create-module.py etiqueta
$ cd ..
$ ./waf configure --enable-examples --enable-tests
$ ./waf build
$ ./test.py
```

Con el proceso anterior, se generan los *scripts* esqueletos del módulo, los cuales se modificaron para integrar los campos y métodos requeridos. Los códigos que conforman al módulo **etiqueta** se encuentran en el apéndice C. En la página 100 se encuentra el código **etiqueta.h**, en el cual se aprecia la declaración de los campos y métodos que se incluyeron. En la tabla 3.1 se muestran los campos de la clase **NodeTag** y el tipo de datos de acuerdo al lenguaje de programación C++:

Campo	Descripción	Tipo	Tamaño (bytes)
m_Tag	Etiqueta	**uint64_t**	8
m_Sec	Estado de etiquetamiento	**uint16_t**	2
m_Chi	Número de hijos	**uint16_t**	2

Tabla 3.1: Campos de la clase **NodeTag**.

Los tipos de datos del módulo **etiqueta** son numéricos. De esta manera, la estructura de dicho módulo tiene un tamaño de 12 bytes.

Los campos del módulo **etiqueta** guardan información requerida en la construcción y uso del árbol de etiquetas. El algoritmo de construcción de la estructura árbol de etiquetas se comenta en la sección 3.4. A continuación se da una breve descripción sobre los campos que se agregaron.

m_Tag: Guarda la etiqueta que fue asignada por un nodo padre en el proceso de construcción del árbol. En este trabajo, se utilizaron etiquetas numéricas, por lo cual este campo está inicializado con cero, lo que significa que aún no se asigna una etiqueta.

m_Sec: Indica la estapa en la cual se encuentra un nodo dentro del proceso de creación del árbol de etiquetas. En el proceso de construcción del árbol hay 3 etapas, las cuales son identificadas con valores enteros:

- **1:** Indica que un nodo no ha sido incluido en el árbol, por lo tanto su etiqueta tiene el valor cero; éste es el valor por defecto para todos los nodos, ya que el árbol de etiquetas todavía no ha sido construido.
- **2:** El nodo ha aceptado ser etiquetado por otro nodo, el cual resultará su padre y será quien le indique la etiqueta que tendrá.
- **3:** Este valor indica que el nodo ha sido etiquetado y por lo tanto el nodo está incorporado al árbol de etiquetas

m_Chi: Indica el número de hijos que tiene cierto nodo. Este valor comienza instaciado con cero para todos los nodos; esto debido a que antes de ejecutar el algoritmo aún no existe estructura de árbol para señalar que un nodo es padre de otro.

Para los campos privados de la clase **NodeTag** se declaran tres métodos accesores y tres métodos mutadores, tal como se observa en el código **etiqueta.h** de la sección C.6. La implementación de los métodos se aprecia en la misma sección del apéndice, pero en el código del archivo **etiqueta.cc**. Los métodos accesores del módulo son: **getTag()**, **getSec()** y **getChi()**; mientras que los métodos mutadores son: **setSec()**, **setTag()** y **setChi()**.

La clase **NodeTag** del fichero **etiqueta.h** se deriva de la clase **Object**, por lo cual, **NodeTag** hereda todas las propiedades del tipo **Object**.

Por el mecanismo de herencia de la programación orientada a objetos, el simulador ns-3 permite la extensión de diversas estructuras que manejen el tipo **Object**. Mediante el método **AggregateObject()**[1] se pueden juntar dos objetos de diferente tipo. De esta manera, con esta asociación desde un objeto x se puede obtener el objeto asociado y usando el método **GetObject()**.

Otra ventaja de heredar a partir de la clase **Object** es el manejo de los Apuntadores Inteligentes (*SmartPointers*)[Nat15a, p. 23]. Estos apuntadores permiten acceder de una manera directa a todos los campos y métodos que están definidos en la clase.

Un *SmartPointer* es una plantilla la cual es parametrizada por un tipo. La plantilla del apuntador inteligente es **Ptr**.

[1] https://www.nsnam.org/doxygen/classns3_1_1_object.html#a79dd435d300f3deca814553f561a2922

Para los fines perseguidos en este trabajo, la creación y uso del módulo **etiqueta** consiste en una estructura que se pueda asociar a la estructura de tipo <**Node**> de ns-3. Visualice la figura 3.3.

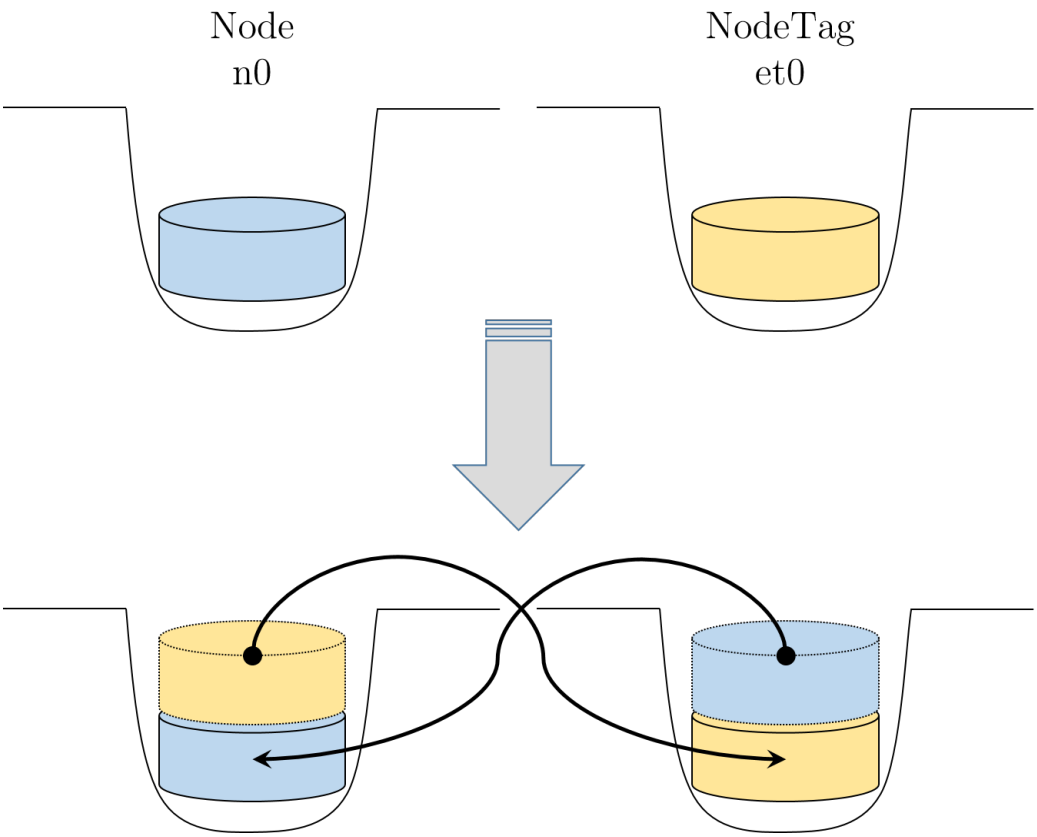


Figura 3.3: Asociación de las estructuras **Node** y **NodeTag**

Al asociar dos tipos de estructuras mediante el método **AggregateObject()**, el resultado es la creación de pilas de prioridad simultáneas[2] mediante las cuales a partir del tipo de estructura base se verifican en orden ascendente las estructuras asociadas usando el método **GetObject()** y especificando el tipo de plantilla que se busca. De esta manera se pueden consultar los campos de cierta estructura. Esta consulta es realizable gracias a que los *SmartPointers* permiten el uso del operador de acceso `->`.

[2] https://www.nsnam.org/doxygen/classns3_1_1_object.html#a1bd7211125185a6cd511c35fea4e500f

Ejemplo 22. En las siguientes líneas de código, las cuales son un extracto de una simulación en ns-3, se muestra la asociación de dos estructuras, una es del tipo <**Node**> y la otra <**NodeTag**>. Además se aprecia cómo después de realizar la asociación entre estas estructuras se puede acceder de forma recíproca a partir de alguna de éstas a cierto campo de la otra para su consulta y de igual forma hacer una modificación.

```
1 for (uint32_t i = 0; i < size; ++i){
2         Ptr<NodeTag> n0 = CreateObject<NodeTag> ();
3         n0->setTag(i);
4         nodes.Get(i)->AggregateObject(n0);
5         std::cout<<nodes.Get(i)->GetObject<NodeTag>()->getTag()<<"\n";
6 }
```

El código anterior recorre mediante un ciclo **for** todos los nodos que forman parte de un contenedor de nodos llamado **nodes** el cual se definió de tamaño **size** al realizar la simulación.

Dentro de las instrucciones a realizar en cada iteración del ciclo , se crea un *SmartPointer* de tipo **NodeTag** usando el método **CreateObject()**[3], el cual es asignado a **n0**.

En la línea 3 se observa la instrucción donde el apuntador inteligente usa el método **setTag()** de la clase **NodeTag** para asignar al campo etiqueta el contador del ciclo.

En la siguiente línea de código se fusiona el modelo nodo y etiqueta usando el método **AggregateObject()** desde el objeto nodo.

Enseguida en la línea 5 se muestra la etiqueta asociada al nodo usando el método **GetObject()** desde un nodo del contenedor; el valor de esta etiqueta será una salida a consola.

Una vez que se ha explicado la manera en que opera el módulo **etiqueta** y como se puede asociar al modelo estándar utilizado en ns-3 para los nodos. Se procederá a la adaptación del trabajo mostrado en [FCoVa12] para usarlo en la versión ns-3.22.

[3]Se comenta en la documentación del simulador ns-3 que para la creación de objetos se utiliza este método cambiando el estilo habitual de C++[Nat15a, p.4].

3.3. Actualización del GPSR a la versión ns-3.22

El trabajo mostrado en [FCoVa12] menciona que los archivos de la implementación que desarrollan se encuentran disponibles para su revisión en el vínculo: `http://codereview.appspot.com/5401042`. La implementación de estos autores fue desarrollada en el simulador ns-3 versión número 12; sin embargo, al ejecutar este trabajo en una edición del *software* más reciente como ns-3.22 se encontró necesario realizar algunos ajustes, los cuales se comentarán en esta sección.

Primeramente, se integrarán los módulos propios para la ejecución del algoritmo GPSR. Una vez que se cuente con los módulos adecuados, éstos se revisarán y se ajustarán para que puedan operar en la versión 22. Posterior a la actualización, se ejecutarán los ejemplos integrados por los autores en su implementación y se verificará que funcione de acuerdo a su descripción teórica.

Al acceder al enlace del trabajo y descargar el código que proporcionan los autores se observa que existen dos carpetas denominadas: **`a`** y **`b`**. En cada una de ellas se encuentran versiones de los módulos que insertan en el simulador ns-3 para poder implementar el algoritmo GPSR. Para el presente trabajo se optó por elegir la carpeta **`b`**, por considerarse la versión más actual del trabajo.

Al interior de la carpeta **`b`** se encuentra una carpeta denominada **`src`** y dentro de ella dos carpetas más que llevan por nombre **`location-service`** y **`gpsr`**. Por lo anterior se entiende que al encontrarse ambas carpetas en la sección **`src`**, éstas deben ser agregadas como módulos nuevos al simulador.

A continuación se describe el proceso requerido para la integración de los módulos señalados, así como los ajustes pertinentes para su actualización.

Integración del módulo location-service

1. Ubicados en **`ns-3/src`**, directorio donde se almacenan todos los módulos, ejecutamos:

```
$ ./create-module.py location-service
```

   De esta manera se generan los esqueletos de los archivos del módulo, como ya se ha explicado previamente. Con esto se logra que el simulador realice todos los procesos necesarios para que incorpore a **`location-service`** como parte del sistema.

2. Enseguida se procedió a reemplazar el contenido de la carpeta **model** y del archivo **wscript** por el material descargado del parche **b** de la implementación de [FCoVa12]. Esta operación se puede realizar mediante un explorador de archivos.

3. Posteriormente, se deben ejecutar el código del listado 3.1, comentadas en la página 48, esto con el fin de que el nuevo módulo esté integrado correctamente para su funcionamiento. En caso de que se presente algún inconveniente con el módulo agregado, se podrá detectar durante la realización de este paso.

   Así, al ejecutar las líneas descritas en la página 48, se detectó la necesidad de realizar un cambio sobre el código debido al desfase de versiones. Ya que, tal como se comenta en [Nat15a, p. 139], la forma de declarar cabeceras públicas cambió entre las ediciones del simulador. De esta manera en el archivo **wscript** que se ha generado en la ubicación **ns-3/src/location-service** se sustituyó la línea 16:

```
16 headers = bld.new_task_gen(features=['ns3header'])
```

   por el código:

```
16 headers = bld(features='ns3header')
```

   Este mismo cambio fue hecho como observación a los autores en el sitio donde colocaron sus códigos para revisión[4].

4. Después de este ajuste, se repite el paso señalado en el número 3, es decir volver a ejecutar el listado 3.1. Como resultado, el simulador marcó como incorporado el módulo **location-service** en la sección de módulos.

La importancia del módulo **location-service** radica en que es por medio de esta herramienta que se puede conocer la localización de todos los nodos que se encuentran en la red. Mediante este módulo se pretende emular la acción de un sistema de ubicación como lo es el GPS, que asociado a los nodos de la red permite que la ubicación de éstos sea una información accesible en todo momento.

Integración del módulo gpsr

1. Se siguió un proceso análogo al utilizado para agregar **location-service**. De igual forma, ubicados en el directorio **ns-3/src** se ejecutó:

[4] https://codereview.appspot.com/5401042/patch/2077/3076

```
$ ./create-module.py gpsr
```

2. Creados los archivos esqueletos, se accedió a la carpeta del módulo y se reemplazaron por el contenido del parche **b** del trabajo [FCoVa12].

3. Se realizó la compilación del simulador ejecutando el listado 3.1.

   Al realizar el paso número 3, se detectaron varios desarreglos con este módulo y dado que sus archivos son de mayor extensión en comparación con **location-service**, fue necesario realizar más modificaciones. A continuación se enlistan dichos cambios:

   - De forma análoga al proceso que se realizó para el módulo de localización se corrigió en la línea 19 del archivo **wscript** la declaración de las cabeceras públicas. Modificando:

```
19 headers = bld.new_task_gen(features=['ns3header'])
```

     por el código:

```
19 headers = bld(features='ns3header')
```

   - En el mismo archivo **wscript** de la carpeta **gpsr**, las líneas 29 y 30 que originalmente contenían:

```
29     if bld.env.ENABLE_EXAMPLES:
30         bld.add_subdirs('examples')
```

     fueron reemplazadas por:

```
29     if bld.env['ENABLE_EXAMPLES']:
30       bld.recurse
```

   - Para los archivos **gpsr.cc** y **gpsr-ptable.h** de la carpeta **model** se reemplazó la llamada a la biblioteca **ns3/random-variable.h** por **ns3/random-variable-stream.h** [Nat15a, Sec. 1.2.1]. En el archivo **gpsr.cc** el reemplazo se da en la línea 12:

```
12 #include "ns3/random-variable.h"
```

     reemplazado por:

```
12 #include "ns3/random-variable-stream.h"
```

     Para el archivo **gpsr-ptable.h** el cambio de la llamada se da en la línea 15. La justificación de este reemplazo es la misma que marca el manual de ns-3.

- En el archivo **gpsr.h** de la carpeta **model** se tuvo que la inclusión de la biblioteca **ipv4-l4-protocol.h** en la línea 14, para la versión ns-3.22 pasó a ser **ip-l4-protocol.h**. Se sustituyó:

```
14 #include "ns3/ipv4-l4-protocol.h"
```

por:

```
14 #include "ns3/ip-l4-protocol.h"
```

Dicha modificación trae como consecuencia que también se corrigieran las declaraciones de los respectivos métodos en las líneas 85, 86 y 132. Así las líneas:

```
85   void SetDownTarget (Ipv4L4Protocol::DownTargetCallback callback);
86   Ipv4L4Protocol::DownTargetCallback GetDownTarget (void) const;
```

fueron reemplazadas por:

```
85   void SetDownTarget (IpL4Protocol::DownTargetCallback callback);
86   IpL4Protocol::DownTargetCallback GetDownTarget (void) const;
```

y la línea 132:

```
132   Ipv4L4Protocol::DownTargetCallback m_downTarget;
```

se cambió a:

```
132   IpL4Protocol::DownTargetCallback m_downTarget;
```

Al cambiar la declaración de los métodos también se procedió a modificar la parte correspondiente a su implementación en el fichero de ejecución **gpsr.cc**, por lo que se realizaron cambios en las líneas originales 964 y 970:

```
963 void
964 RoutingProtocol::SetDownTarget (Ipv4L4Protocol::DownTargetCallback
        callback)
965 {
966   m_downTarget = callback;
967 }
968
969
970 Ipv4L4Protocol::DownTargetCallback
971 RoutingProtocol::GetDownTarget (void) const
972 {
973   return m_downTarget;
974 }
```

y se sustituyeron por:

```
963 void
964 RoutingProtocol::SetDownTarget (IpL4Protocol::DownTargetCallback
        callback)
```

```
965 {
966   m_downTarget = callback;
967 }
968
969
970 IpL4Protocol::DownTargetCallback
971 RoutingProtocol::GetDownTarget (void) const
972 {
973   return m_downTarget;
974 }
```

- Para el archivo **gpsr-rqueue.cc** que se encuentra en la carpeta **model**, se marcó como advertencia que en la línea 50 había una variable que no estaba en uso, por lo que el renglón:

```
50 const Ipv4Address addr = dst;
```

se comentó, usando el operador **//**:

```
50 //  const Ipv4Address addr = dst;
```

- Con los ajustes anteriores, se ejecutó el paso número 3 para revisar el acoplamiento del módulo; sin embargo, se marcó un problema con la clase **UniformVariable**. Revisando la documentación acerca del simulador ns-3.12[5] y la del ns-3.22[6] se apreció que dicha clase ya no estaba vigente, pero la descripción, métodos y referencia a campos se retomaron en la clase **UniformRandomVariable**. Dada esta observación las líneas 82 y 83 del archivo **gpsr.cc** se modificaron:

```
81 /// Random number between [(-GPSR_MAXJITTER)-GPSR_MAXJITTER] used to
       jitter HELLO packet transmission.
82 #define JITTER (Seconds (UniformVariable ().GetValue (-GPSR_MAXJITTER
       , GPSR_MAXJITTER)))
83 #define FIRST_JITTER (Seconds (UniformVariable ().GetValue (0,
       GPSR_MAXJITTER))) //first Hello can not be in the past, used only
        on SetIpv4
```

quedando ajustadas como:

```
81 /// Random number between [(-GPSR_MAXJITTER)-GPSR_MAXJITTER] used to
       jitter HELLO packet transmission.
82 #define JITTER (Seconds (UniformRandomVariable ().GetValue (-
       GPSR_MAXJITTER, GPSR_MAXJITTER)))
83 #define FIRST_JITTER (Seconds (UniformRandomVariable ().GetValue (0,
       GPSR_MAXJITTER))) //first Hello can not be in the past, used only
        on SetIpv4
```

[5] https://www.nsnam.org/docs/release/3.12/doxygen/classns3_1_1_uniform_variable.html

[6] https://www.nsnam.org/doxygen/classns3_1_1_uniform_random_variable.html

4. Realizados todos los cambios enlistados previamente, se procedió a volver a compilar el simulador mediante el ya mencionado paso 3 y se obtuvo como resultado que el módulo **gpsr** se acopló con éxito al simulador.

La importancia de integrar el módulo **gpsr** a ns-3 radica en que es aquí donde se implementa el algoritmo para poderlo usar en el simulador. Así, una vez acoplado el módulo, se procedió a probar los *scripts* que vienen anexos en la carpeta **examples**. Para esto, ubicados en la carpeta **ns-3.22**, se emplean los comandos[7]:

```
$ cp src/gpsr/examples/gpsr-test1.cc scratch/mygpsr-test1.cc
$ ./waf --run scratch/mygpsr-test1
```

Sin embargo, al ejecutar dichos ejemplos se marcaron ciertos errores. Lo anterior llevó al uso del depurador de *scripts*: gdb, para detectar en que parte del código se marcaban las observaciones y realizar las depuraciones pertinentes.

Depuración del modelo gpsr.cc

Al hacer una revisión sobre el archivo **gpsr.cc** de la carpeta **model** con ayuda del depurador gdb, se encontró que por el desfase de versiones del simulador era necesario hacer algunos renombramientos pues se marcaban algunos errores de sintaxis y no permitían la ejecución correcta de los ejemplos.

Para la manipulación del depurador gdb se utilizó el comando:

```
$ ./waf --run scratch/mygpsr-test1 --command-template="gdb --args
    %s <args>"
```

Mediante esta herramienta de depuración pudieron detectarse las siguientes atenciones al código:

- Como consecuencia de actualizar a la biblioteca **UniformRandomVariable**, como se comentó en la página 57, fue necesario además agregar la biblioteca **double.h**; lo anterior debido a que uno de los métodos generadores de números aleatorios devuelve un valor de tipo **double**, esto de acuerdo a la documentación de **UniformRandomVariable**. De esta manera, dicha biblioteca se incluyó mediante la línea:

```
18 #include "ns3/double.h"
```

[7]La carpeta **scratch**, destino a donde se copia el archivo ejemplo bajo el nombre de **mygpsr-test1.cc**, es la carpeta utilizada para correr las simulaciones en ns-3[Nat15b, p. 33].

- Con el desfase de versiones y la inclusión de **UniformRandomVariable**, la macro **FIRST_JITTER** definida como:

```
84 #define FIRST_JITTER (Seconds (UniformRandomVariable ().GetValue (0,
       GPSR_MAXJITTER))) //first Hello can not be in the past, used only on
       SetIpv4
```

se reconocía como errónea para el método **SetIpv4()**, por lo que a este método, cuyas líneas originalmente decían:

```
659 void
660 RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
661 {
662   NS_ASSERT (ipv4 != 0);
663   NS_ASSERT (m_ipv4 == 0);
664
665   m_ipv4 = ipv4;
666
667   HelloIntervalTimer.SetFunction(&RoutingProtocol::HelloTimerExpire, this);
668   HelloIntervalTimer.Schedule (FIRST_JITTER);
669
670   //Schedule only when it has packets on queue
671   CheckQueueTimer.SetFunction (&RoutingProtocol::CheckQueue, this);
672
673   Simulator::ScheduleNow (&RoutingProtocol::Start, this);
674 }
```

le fueron agregadas las siguientes instrucciones:

```
659 void
660 RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
661 {
662   NS_ASSERT (ipv4 != 0);
663   NS_ASSERT (m_ipv4 == 0);
664
665   m_ipv4 = ipv4;
666
667   HelloIntervalTimer.SetFunction(&RoutingProtocol::HelloTimerExpire, this);
668     double min = 0.0;
669     double max = GPSR_MAXJITTER;
670     Ptr<UniformRandomVariable> prueba = CreateObject<UniformRandomVariable
          >();
671     prueba->SetAttribute ("Min", DoubleValue (min));
672     prueba->SetAttribute ("Max", DoubleValue (max));
673
674   HelloIntervalTimer.Schedule  (Seconds (prueba->GetValue (min, max)));
675
676   //Schedule only when it has packets on queue
677   CheckQueueTimer.SetFunction (&RoutingProtocol::CheckQueue, this);
678
679   Simulator::ScheduleNow (&RoutingProtocol::Start, this);
680 }
```

- De la misma manera, en el método **HelloTimerExpire()** se hace un ajuste similar, ya que el depurador gdb señaló una obsservación cuando se hacía

uso de la macro **JITTER**. Por lo que a las líneas que originalmente se marcaban en el modelo **gpsr.cc** como:

```
682 void
683 RoutingProtocol::HelloTimerExpire ()
684 {
685   SendHello ();
686   HelloIntervalTimer.Cancel ();
687   HelloIntervalTimer.Schedule (HelloInterval + JITTER);
688 }
```

fueron reemplazadas por las instrucciones:

```
682 void
683 RoutingProtocol::HelloTimerExpire ()
684 {
685   SendHello ();
686   HelloIntervalTimer.Cancel ();
687   double min = -1*GPSR_MAXJITTER;
688   double max = GPSR_MAXJITTER;
689   Ptr<UniformRandomVariable> p_Jitter = CreateObject<UniformRandomVariable
          >();
690   p_Jitter->SetAttribute ("Min", DoubleValue (min));
691   p_Jitter->SetAttribute ("Max", DoubleValue (max));
692
693   HelloIntervalTimer.Schedule(HelloInterval+Seconds(p_Jitter->GetValue(min,
          max)));
694 }
```

- Después del proceso anterior de depuración, ya no se marcaban observaciones ni errores de sintaxis a la hora de procesar las simulaciones de los ejemplos, sin embargo éstas no concluían. Mediante la herramienta de depuración se dedujo que dentro del método **NotifyInterfaceUp()**, debía realizarse el intercambio en el orden de ejecución de las instrucciones 463 y 464:

```
462   socket->SetRecvCallback (MakeCallback(&RoutingProtocol::RecvGPSR, this));
463   socket->BindToNetDevice (l3->GetNetDevice (interface));
464   socket->Bind (InetSocketAddress (Ipv4Address::GetAny (), GPSR_PORT));
```

para ejecutarse en el orden:

```
462   socket->SetRecvCallback (MakeCallback(&RoutingProtocol::RecvGPSR, this));
463   socket->Bind (InetSocketAddress (Ipv4Address::GetAny (), GPSR_PORT));
464   socket->BindToNetDevice (l3->GetNetDevice (interface));
```

Con el proceso de depuración y los ajustes comentados fue posible realizar la ejecución completa de los siete ejemplos con los que cuenta el parche de la implementación del trabajo [FCoVa12]. En la siguiente sección se describirán a detalle éstos.

Ejemplos disponibles para GPSR

En los ejemplos mostrados a continuación, es posible modificar los argumentos del *script*, sin embargo, se ejecutará la versión tal cual viene cargada en el parche manipulado. Además, los parámetros usados por defecto para las simulaciones se muestran en la tabla 3.2[FCoVa12, p.355].

Parámetro	**Valor**
Tiempo de simulación:	30 s
Separación entre nodos:	100 m
Topología de los nodos:	Cuadrícula de 10 x 10
Rango de transmisión:	100 m
Número máximo de paquetes:	100

Tabla 3.2: Parámetros de simulación por defecto

En lo que se refiere al acomodo de los nodos, éstos forman una cuadrícula que se sitúa sobre un plano cartesiano, abarcando el primer cuadrante, comenzando en la posición (0,0) y desde ese punto distan la separación por defecto hacia la derecha.

La colocación de nodos en la cuadrícula es por filas y se va llenando de acuerdo al valor indicado en el parámetro **`gridWidth`** (Ancho de cuadrícula) del método **`SetPositionAllocator()`** integrado en el auxiliar *MobilityHelper*.

Alcanzado el ancho de fila, los nodos que resten por situarse en la topología, se ubicarán tomando como referencia la posición del primer nodo de la fila que acaba de llenarse y se le suma la distancia de separación en la coordenada *y*.

Además, el primer nodo que se coloca tiene como valor de identificación el número cero (Nodo-0) y a partir de él la secuencia de numeración es entera consecutiva ascendente.

Se debe destacar que, las simulaciones buscan mostrar el uso del algoritmo GPSR para el tráfico de paquetes que se genere entre dos nodos: un servidor y un cliente. Se señalará la posición de estos elementos mediante la notación Pos.(x,y).

Por último, el número máximo de paquetes que puede transmitir un cliente está definido por defecto, como lo señala la tabla 3.2. Para cada paquete que se envíe, la simulación realizará un recorrido cliente-servidor y enseguida se emitirá una respuesta mediante el recorrido servidor-cliente. El tiempo intermitente entre cada emisión de paquetes por parte del cliente se señala en cada ejemplo. Puede suceder que no se envíe el máximo número de paquetes, ya que el tiempo de simulación es fijo a 30 segundos para todos las pruebas.

Las topologías de estos ejemplos se muestran en la lista siguiente:

1. Ejemplo: **`gpsr-test1.cc`**. Observe la figura 3.4.

   Nodos en la simulación: 2. Cliente: Nodo-0 Pos.(0,0).
   Ancho de cuadrícula: 2.
   Servidor: Nodo-1 Pos.(0,100). Intervalo entre paquetes: 1 s.

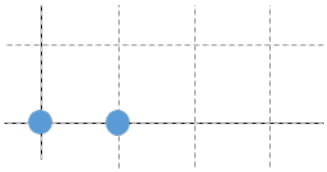


Figura 3.4: Topología de **`gpsr-test1.cc`**

2. Ejemplo: **`gpsr-test2.cc`**. Visualice la figura 3.5.

   Nodos en la simulación: 2. Servidor: Nodo-1 Pos.(0,200).
   Separación entre nodos: 200 m. (Modificación al valor como prueba) Cliente: Nodo-0 Pos.(0,0).
   Ancho de cuadrícula: 2. Intervalo entre paquetes: 1 s.

   Dado que la separación fue mayor que el rango de transmisión, no se realiza comunicación entre los dispositivos.

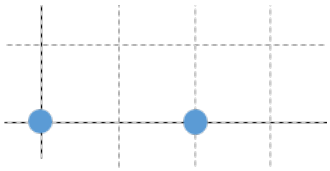


Figura 3.5: Topología de **`gpsr-test2.cc`**

3. Ejemplo: **`gpsr-test3.cc`**. Los detalles se aprecian en la figura 3.6.

   Nodos en la simulación: 4. Cliente: Nodo-0 Pos.(0,0).
   Ancho de cuadrícula: 2. Intervalo entre paquetes: 0.5 s.
   Servidor: Nodo-3 Pos.(100,100). Máximo número de paquetes: 50 (Reducción del valor como prueba).

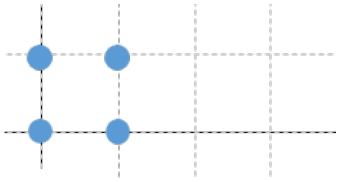


Figura 3.6: Topología de **gpsr-test3.cc**

4. Ejemplo: **gpsr-test4.cc**. En esta simulación se realiza una prueba extra, se colocan dos servidores y dos clientes y se establece comunicación entre dos parejas ajenas cliente-servidor de manera simultánea. Vea la figura 3.7.

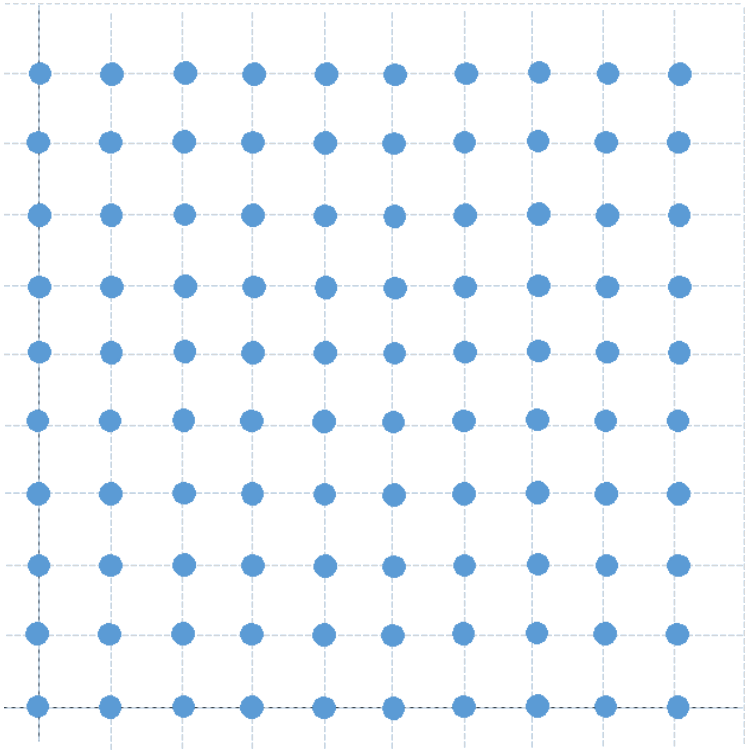


Figura 3.7: Topología de **gpsr-test4.cc**

Nodos en la simulación: 100.
Ancho de cuadrícula: 10.
Servidor_1: Nodo-99 Pos.(900,900).
Cliente_1: Nodo-0 Pos.(0,0).
Servidor_2: Nodo-9 Pos.(900,0).
Cliente_2: Nodo-91 Pos.(100,900).
Intervalo entre paquetes: 1 s.

5. Ejemplo: **`gpsr-test5.cc`**. El despliegue de la topología se puede observar en la figura 3.8.

   Nodos en la simulación: 8.
   Cliente: Nodo-4 Pos.(0,100).
   Ancho de cuadrícula: 4.
   Servidor: Nodo-7 Pos.(300,100).
   Intervalo entre paquetes: 0.5 s.

   Para comenzar con los casos de prueba en los que el algoritmo GPSR alcanza máximos locales, en este ejemplo se remueven los nodos 0 y 6 antes de comenzar la simulación. Con esta acción se pretende generar un panorama para apreciar el funcionamiento de la estrategia de recuperación.

   Nodos removidos:

   - Nodo-0 Pos.(0,0).
   - Nodo-6 Pos.(200,100).

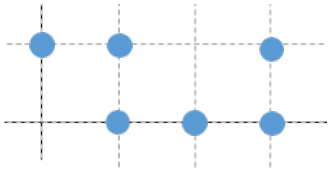


Figura 3.8: Topología de **`gpsr-test5.cc`**

6. Ejemplo: **`gpsr-test6.cc`**. Los detalles de esta simulación se aprecian en la figura 3.9.

   Nodos en la simulación: 20.
   Cliente: Nodo-8 Pos.(0,200).
   Ancho de cuadrícula: 4.
   Servidor: Nodo-15 Pos.(300,300).
   Intervalo entre paquetes: 0.5 s.

   Nodos removidos:

   - Nodo-5 Pos.(100,100).
   - Nodo-6 Pos.(200,100).
   - Nodo-10 Pos.(200,200).
   - Nodo-12 Pos.(0,300).
   - Nodo-13 Pos.(100,300).
   - Nodo-14 Pos.(200,300).

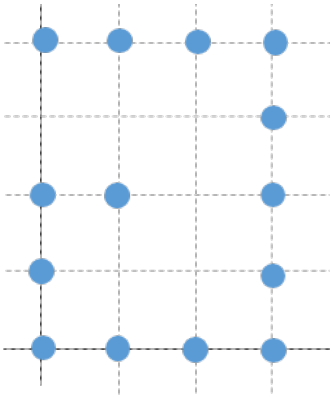


Figura 3.9: Topología de **`gpsr-test6.cc`**

7. Ejemplo: **`gpsr-test7.cc`**. El arreglo para la topología de red de esta simulación se visualiza en la figura 3.10.

   Nodos en la simulación: 100. Cliente: Nodo-0 Pos.(0,0).

   Ancho de cuadrícula: 10.

   Servidor: Nodo-99 Pos.(900,900). Intervalo entre paquetes: 0.5 s.

   Nodos removidos:

   - Nodo-33 Pos.(300,300).
   - Nodo-34 Pos.(400,300).
   - Nodo-35 Pos.(500,300).
   - Nodo-36 Pos.(600,300).
   - Nodo-43 Pos.(300,400).
   - Nodo-44 Pos.(400,400).
   - Nodo-45 Pos.(500,400).
   - Nodo-46 Pos.(600,400).
   - Nodo-53 Pos.(300,500).
   - Nodo-54 Pos.(400,500).
   - Nodo-55 Pos.(500,500).
   - Nodo-56 Pos.(600,500).
   - Nodo-63 Pos.(300,600).
   - Nodo-64 Pos.(400,600).
   - Nodo-65 Pos.(500,600).
   - Nodo-66 Pos.(600,600).

Para los ejemplos que se han descrito el simulador ns-3 señalaba la ejecución completa de éstos. Sin embargo, para determinar si cada ejemplo se realizaba de acuerdo a la descripción teórica del algoritmo fue requerida otra herramienta para esta verificación. En la siguiente sección se describirá esta corroboración.

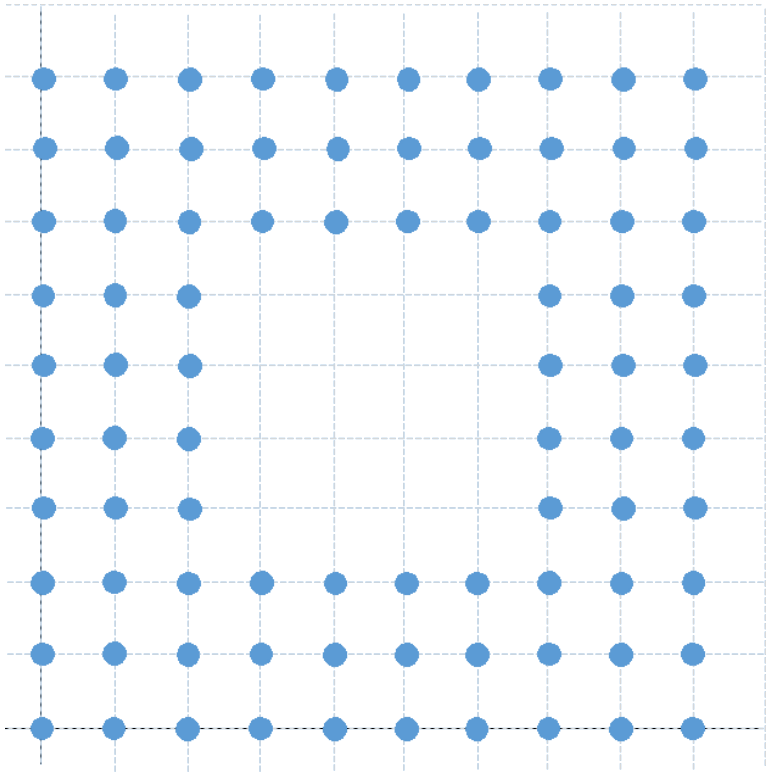


Figura 3.10: Topología de **`gpsr-test7.cc`**

Verificación de ejecución correcta

El simulador ns-3 utiliza un útil componente especial para imprimir mensajes a la consola conocido como el módulo de registro (*Logging Module*)[Nat15b, p.15]. Se dice que dicho componente es multinivel, ya que posee la ventaja de mostrar en la salida de la consola el nivel que se desee de estos mensajes de registro, lo cuales pueden ser mensajes de error, mensajes de advertencia o mensajes con información detallada.

La solicitud para obtener los mensajes de registro puede ser definida por el usuario en cualquier *script*. Existen siete niveles de mensajes de registro:

LOG_ERROR: Muestra mensajes de error producidos en la simulación.

LOG_WARN: Imprime en la salida mensajes de tipo advertencia. Raramente aparecen en las simulaciones.

LOG_DEBUG: Permite mostrar mensajes para realizar *debugging*.

LOG_INFO: Señala mensajes de información acerca del progreso del programa.

LOG_FUNCTION: Muestra mensajes que describen que función está actuando.

LOG_LOGIC: Imprime mensajes que describen el flujo lógico de la simulación sin especificar la función que actúa.

LOG_ALL: Muestra todos los niveles de registro anteriores.

Para visualizar los mensajes de registro que emite una simulación es necesario habilitar este módulo previamente a la ejecución de la simulación. Dado que el caso que se desea analizar en los ejemplos del algoritmo GPSR corresponde a la manera en que la información de los paquetes va siendo transmitida, se habilitará la impresión de registros a nivel lógico mediante la siguiente instrucción, que debe ser ejecutada antes de correr la simulación deseada:

```
$ export NS_LOG=GpsrRoutingProtocol=logic
$ ./waf --run scratch/mygpsr-test1
```

Se pudo apreciar mediante la utilización de la impresión de registros a nivel lógico que en las simulaciones de los ejemplos no se obtenía el correcto funcionamiento del algoritmo GPSR como lo describen en [KK00]. Al hacer un análisis del código del algoritmo se detectó que el error se encontraba en el método **`Forwarding()`**, en un enunciado **`if-else`** de doble alternativa ubicado en la línea 925, pues no se permitía la ejecución de las líneas 929 a la 957 porque la condición siempre era verdadera:

```
925 if(m_neighbors.isNeighbour (dst){
926   nextHop=dst
927   }
928 else{
929 ...

958   }
```

Por lo que se procedió a comentar estas líneas.

```
925 /* if(m_neighbors.isNeighbour (dst){
926   nextHop=dst
927   }
928 else{ */
929 ...

958 //  }
```

El ajuste anterior permitió la ejecución de la implementación del algoritmo GPSR con el uso correcto de la estrategia de recuperación, haciendo uso de la regla de la mano derecha como lo señalan en [KK00].

Con la finalidad de mejorar la implementación comentadada se hicieron algunas modificaciones en los mensajes de registros. Primeramente, se anexó la línea:

```
438   NS_LOG_LOGIC (route->GetOutputDevice () << "␣forwarding␣in␣Recovery␣to␣" << dst
          << "␣through␣" << route->GetGateway () << "␣packet␣" << p->GetUid ());
```

ya que el código original no marcaba el trazo que seguían los paquetes al entrar al modo de recuperación, con la línea anterior se habilita dicha impresión.

Además, se comentó la línea 207:

```
207           NS_LOG_LOGIC ("Broadcast␣local␣delivery␣to␣" << dst);
```

dado que dicha línea generaba demasiadas impresiones a la consola, debido a que hace referencia a la parte en que se realiza una operación de tipo *broadcast*; así, se prefirió omitir este tipo de mensajes por la excesiva cantidad de estos procesos y se complicaba seguir el flujo de los paquetes.

```
207 //          NS_LOG_LOGIC ("Broadcast local delivery to " << dst);
```

La última modificación que se estableció respecto a las salidas de registro **NS_LOG**, fue el intercambio de orden de las líneas 1051 y 1053:

```
1051  RecoveryMode (dst, p, ucb, header);
1052
1053  NS_LOG_LOGIC ("Entering␣recovery-mode␣to␣"<<dst<<"␣in␣"<<m_ipv4->GetAddress
          (1,0).GetLocal());
```

esto con la intención de que el mensaje de registro apareciera antes de hacer la llamada al método **RecoveryMode()**.

```
1051  NS_LOG_LOGIC ("Entering␣recovery-mode␣to␣"<<dst<<"␣in␣"<<m_ipv4->GetAddress
          (1,0).GetLocal());
1052
1053  RecoveryMode (dst, p, ucb, header);
```

Con lo comentado a lo largo de esta sección se da la habilitación del módulo **gpsr** y esto permite el simulador haga uso de este algoritmo de encaminamiento basado en la posición. De esta manera, una vez acoplado el GPSR con un correcto funcionamiento en el despliegue de redes ad-hoc; la intención es tomarlo como base para optimizar la parte correspondiente a la herramienta de recuperación con la que cuenta y con esto tener la implementación del algoritmo árbol de etiquetas.

Hasta este punto, realizando las actualizaciones y depuraciones que se comentaron, se afirma que el algoritmo de Encaminamiento Voraz Perimétrico Local, presentado en [FCoVa12], queda totalmente funcional para la versión ns-3.22, de acuerdo a la parte descrita originalmente en [KK00].

3.4. Estructura PositionTree

En esta sección se partirá de que el GPSR ha quedado acoplado en el ns-3 y se continuará con la modificación necesaria para lograr la implementación del algoritmo árbol de etiquetas.

Primeramente, se comentará la manera en que se construye la estructura árbol de etiquetas, explicando cómo se desarrolla el proceso de etiquetado; para posteriormente abordar la implementación de la estrategia de recuperación que requiere el algoritmo descrito en [CMT07].

Para implementar la estructura árbol de etiquetas se hará uso del módulo **etiqueta** que se comentó en la sección 3.2. Recordemos que en este módulo existen 3 campos que se definieron: **m_Tag**, **m_Sec** y **m_Chi**. Además, recordemos que cada uno de los nodos o dispositivos de la red cuenta con una asociación directa a este módulo construido, mecanismo fundamental que permite la construcción de la estructura y del funcionamiento del algoritmo.

Inicio del árbol

El proceso de generación de esta estructura comienza, como se comentó en la página 37, con la elección aleatoria de un nodo del conjunto de dispositivos con los que cuenta la red. Este nodo será la raíz del árbol y se le asignará el entero 1 en su campo **m_Tag**. Así mismo, a su campo **m_Sec** se le asigna el entero 3, pues como se señaló en la descripción del módulo **etiqueta**, página 50, el nodo habrá quedado etiquetado. En este caso por ser la base de la estructura estaremos omitiendo el estado 2 de la secuencia.

Enseguida este nodo consultará quiénes son los nodos que están dentro de su rango de transmisión, es decir cuáles son los vecinos con los que se pueden crear enlaces. De esta manera comienza el proceso de etiquetado de los vecinos, siguiendo el proceso que se describe a continuación.

Etiquetado recursivo

1. Se verifica que los nodos vecinos tengan marcado en su campo **m_Sec** el valor correspondiente al estado 1, lo cual implica que su etiqueta está inicializada en 0.

2. Una vez que se han identificado los nodos vecinos que son candidatos a

recibir etiqueta, éstos responden enviando su solicitud de etiquetado y modifican su campo **m_Sec** al estado 2.

3. De acuerdo al orden en que vayan llegando la solicitudes de etiquetado de los nodos mencionados en el paso anterior, el nodo que pasará a ser el padre modifica de manera creciente en una unidad el valor de su campo **m_Chi** de manera previa a cada proceso de etiquetado que realice. Dicho campo indica el número de hijos e inicialmente está asignado con el valor 0. De esta manera se responde a cada solicitud enviando la etiqueta que tiene asignada el padre, guardada en **m_Tag** y el valor momentáneo que tenga **m_Chi**.

4. Enseguida, todos los nodos que enviaron su solicitud de etiquetado habrán recibido en el mensaje respuesta que envío el nodo que será el padre en el paso anterior, la información de su etiqueta y además el valor de **m_Chi** distinto para cada uno de ellos. Para asignar la etiqueta que le corresponde a cada nodo, recordando que **m_Tag** se trata de un campo numérico; se tomará el valor de la etiqueta recibida, se multiplicará por 10 y se le sumará el valor **m_Chi** que se recibió . De esta manera, los nodos quedan etiquetados y resta modificar el campo **m_Sec** al valor entero 3, que indica que finalizó la asignación de etiqueta para ese nodo.[8]

5. Los nodos que acaban de recibir su etiqueta en el paso anterior, llevarán a cabo un proceso recursivo tomando ahora el papel de nodos con capacidad para etiquetar, es decir se vuelve a comenzar con el paso 1 y los nodos que aún tengan marcado el campo **m_Sec** en estado 1 serán los nodos que se puedan etiquetar, para cualquier otro estado se omitirá el proceso de etiquetado.

Modificación del algoritmo GPSR

En la sección 3.3, se realizó la actualización necesaria al trabajo [FCoVa12] para que se ejecutara correctamente en el simulador ns-3 en su versión 22.

Recordemos que la importancia de que el algoritmo GPSR sea implementado correctamente radica en que el objetivo principal de este trabajo de tesis es la implementación del algoritmo árbol de etiquetas y gracias al gran parecido que

[8]Se puede apreciar que siguiendo este procedimiento, el algoritmo presenta una falla si un dispositivo fuera a etiquetar a más de 9 nodos, sin embargo se ha dicho que esta estructura y el algoritmo de árbol de etiquetas operará para redes distribuidas uniformemente por lo que no se esperan cúmulos o aglomeraciones de nodos en ciertas regiones.

tiene con el trabajo [KK00] se realizará una modificación a la implementación en la parte correspondiente a la estrategia de recuperación.

Como se comentó en el listado 2.1 en la página 36, cuando se alcanza un máximo local se ejecuta una estrategia de recuperación para continuar con la búsqueda de la ruta hacia el destino.

El modo de recuperación entra en operación cuando un paquete llega a un nodo y éste busca una opción para reenviar el contenido para llegar al destino, esto porque se encuentra en el modo de reenvío voraz progresivo; pero resulta que las distancias de las opciones que tienen como nodos vecinos con respecto al nodo destino es mayor que la distancia de la posición actual con el destino. Es en este momento que los autores [KK00] aplican la regla de la mano derecha, que consiste en ubicar cual fue la arista por la que el paquete incidió y a partir de una medición de ángulos se reenviará el paquete al primer nodo que se encuentre en sentido de las manecillas del reloj.

En la figura 3.11 se muestra al nodo *s* transmitiendo un paquete al destino *D*. Es en el reenvío de paquetes por parte del nodo *R* que se alcanza un máximo local y debe aplicarse la regla de la mano derecha, reenviando el paquete hacia el nodo n_1.

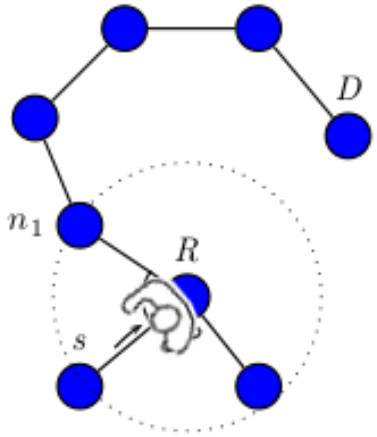


Figura 3.11: Estrategia de recuperación *right-hand rule*.

Sin embargo, la estrategia de recuperación que se acaba de comentar fue reemplazada por otra que utilizaba la estructura del árbol de etiquetas y de esta manera conseguir la implementación del trabajo de [CMT07]. A continuación se describirá el modo empleado cuando hay un máximo local:

1. Una vez que no se pueda reenviar el paquete en modo voraz progresivo, se procederá a consultar la etiqueta del nodo destino. Esta consulta puede realizarse, ya que hemos afirmado que en el algoritmo árbol de etiquetas, la ubicación y la información correspondiente al módulo **`etiqueta`** son accesibles en todo momento.

2. Conocida la etiqueta del destino, se procede a tomar la etiqueta del nodo que debe reenviar el paquete y se realiza una comparación entre éstas; esto para buscar, como se comentó en la página 39, el *ancestro común más coincidente*. Como primera acción se trabaja con los tamaños de las etiquetas, ya que por la estructura que se tiene en el árbol de etiquetas, la longitud de la etiqueta raíz es uno y conforme se vaya descendiendo por los niveles de los hijos del árbol, el tamaño de las etiquetas va aumentando en una unidad por nivel descendido. Conocer el tamaño de las etiquetas puede ser visto como un proceso en el que se cuenta el número de dígitos que tiene la etiqueta de cada nodo. Al realizar la comparación entre etiquetas se ejecutarán dos acciones:

   a) Ascenso: Si el tamaño de la etiqueta del nodo que reenvía fuera mayor que el tamaño de la etiqueta del nodo destino entonces para encaminar debemos acudir al ancestro. Es decir, utilizando la estructura del árbol de etiquetas, se elegirá al padre del nodo que reenvía y con esto se sube un nivel en el árbol. La finalidad de esta operación es llegar al *ancestro común más coincidente*.

   b) Descenso: Cuando el tamaño de la etiqueta del nodo que reenvía sea menor que el de la etiqueta destino y además se cumpla que la etiqueta completa del nodo que está reenviando sea prefijo del destino, ésta funcionará como *ancestro común más coincidente*; entonces se procede a bajar, es decir descender en la estructura del árbol de etiquetas buscando como nodo de encaminamiento aquel que empate como prefijo del destino pero de tamaño mayor en una unidad. En caso de que no se cumpla que la etiqueta del nodo que reenvía sea prefijo del destino, entonces se debe ascender hasta encontrar al *ancestro común más coincidente*.

3. El proceso de ascenso o descenso sobre el árbol de etiquetas, concluye en el momento en que el algoritmo de reenvío de paquetes pueda volver al modo voraz progresivo, es decir que se salga de la etapa de recuperación.

A continuación, en el listado 3.3 se muestra el pseudo-código que describe el modo de recuperación del árbol de etiquetas. Para esto se manejará la siguiente notación:

- R es un nodo que recibe un paquete p para un destino D.
- N es el conjunto de vecinos a distancia menor o igual que 1 del nodo R.
- n es un nodo de N que se usa para enviar el paquete.
- D representa el destino del paquete.
- $d(n_1, n_2)$ simboliza la distancia euclidiana entre nodos.
- $m\_Tag(n)$ Etiqueta asignada a un nodo n del conjunto N
- $\mid m\_Tag(n) \mid$ Longitud de la etiqueta del nodo n. (Número de dígitos).

Listado 3.3: Pseudo-código: Árbol de etiquetas

```
if ∃ n ∈ N | d(n,D) ≤ d(R,D) then
  {Reenvío Voraz}
  Se escoge n ∈ N | mín d(N,D)
  Se reenvía p → n
  return;
else
  {Máximo Local, usar estructura de arbol de etiquetas}
  Se comparan |m_Tag(R)| y |m_Tag(D)|
    if |m_Tag(R)| > |m_Tag(D)|
      {Ascenso}
      Se reenvía p → n | m_Tag(n) es el padre de m_Tag(R)
    else
      if m_Tag(R) es prefijo de m_Tag(D)
        {Descenso}
        Se reenvía p → n | |m_Tag(R)|+1=|m_Tag(n)| y m_Tag(n) es prefijo de m_Tag(D)
      else
        {Ascenso}
        Se reenvía p → n | m_Tag(n) es el padre de m_Tag(R)
      end if
      return;
    return;
    end if
  return;
end if
```

En la siguiente sección se hablará acerca de la manera en que se trabajó en el ns-3 para lograr la implementación del algoritmo árbol de etiquetas y hacer posible su ejecución en las simulaciones.

3.5. Implementación y experimentos

En las secciones previas, se indicó que la implementación del árbol de etiquetas se haría modificando el trabajo de [FCoVa12] para el algoritmo GPSR. Para realizar la modificación se presentaron algunos desafíos que en esta sección se comentará cómo fueron resueltos.

Se ha dicho que para trabajar con el algoritmo árbol de etiquetas, es necesaria una estructura de datos conocida como *PositionTree*. Dicha estructura debe ser creada y puesta en operación a manera de un preproceso ejecutado antes de comenzar el tráfico en la red; sin embargo, las simulaciones que se corren en ns-3 son marcadas como un proceso de ejecución continua, por lo que lograr la generación del árbol de etiquetas representó un reto.

Lo anterior trajo como consecuencia que el proceso de construcción del árbol de etiquetas tuviera que adaptarse al momento en que el GPSR usa los paquetes *HELLO* para el descubrimiento de rutas y vecinos, lo cual ocurre hasta que se inicia el proceso de envío y recepción de paquetes. La diferencia que se tuvo en esta adaptación fue que la raíz del árbol se fijó de manera previa al comienzo de la simulación.

Para lograr la implementación de la herramienta de recuperación descrita en la sección 3.4, se modificaron los métodos originales del GPSR: `RecvGPSR()` y `Forwarding()`.

Además, se crearon y agregaron al algoritmo tres nuevos métodos: `Inunda()`, `mejor_etiqueta()` y `sizeOfTag()`.

En la siguiente sección se describe el proceso de modificación y añadidura que se comenta.

Ejecución de PositionTree

Dado que se emplea como base el algoritmo GPSR y teniendo en cuenta el reto de prescindir de una etapa de ejecución previa; el armado de la estructura árbol de etiquetas se da una vez iniciada la simulación y que los nodos van descubriendo a sus vecinos. Todo esto se hace posible gracias al intercambio y flujo de paquetes que se utilizan para el descubrimiento de rutas y vecinos.

La parte fundamental para que la implementación del algoritmo árbol de etiquetas sea posible, es contar con toda la estructura *PositionTree* armada antes de caer en un caso de máximo local y poder hacer uso del árbol al momento de lanzar la estrategia de recuperación. Con base en el algoritmo pretendido, se realizan los siguientes ajustes y modificaciones.

Primeramente, dado que el proceso de armado del árbol de etiquetas debe ejecutarse una vez, en el modelo central del algoritmo **gpsr.cc** se colocó en la línea 94, una variable global de tipo **bool** inicializada con valor verdadero:

```
94 bool RoutingProtocol::etiqueta = true;
```

con este valor se indica que aún no se construye el árbol.

Anuado a lo anterior, se modifica el método **Forwarding()**, para esto se agrega el siguiente código al original a partir de la línea 936:

```
936   if((m_ipv4->GetObject<Node>()->GetId ())+1==2&& etiqueta==true){
937     std::cout <<"La dirección marcada como origen sería: "<< origin <<" destino:"
            <<dst<< "\n";
938     //etiqueta PERMITE LANZAR LA FUNCIÓN INUNDA SÓLO UNA VEZ, CAMBIA A false
939     etiqueta=false;
940     std::cout<< "INICIO Árbol--> \n";
941     std::cout<< "Nodo_control: " << (m_ipv4->GetObject<Node> ()->GetId ())+1 << "
            \n";
942 //    Se configura el valor de la etiqueta Raíz del árbol
943     m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->setTag(1);
944 //    Se configura el valor del modo a 3, porque ya está etiquetado el nodo raíz
945     m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->setSec(3);
946     Inunda();
947   }
```

y se deja el contenido restante del método intacto.

De esta manera, mediante la condicional agregada, cuando se detecte alguna acción de reenvío de paquetes por parte del nodo con el identificador dos[9], y además, la variable **etiqueta** esté marcada como verdadera, se desencadenará el proceso de construcción del árbol de etiquetas. Observe que en la línea 939, la variable global **etiqueta** cambia su valor a **false**, esto para indicar que el proceso de armado del árbol se realiza una sola vez, así para cualquier otro caso de reenvío por parte del nodo con el identificador dos, no se lanzará el proceso.

La diferencia que se tuvo según los pasos que se describieron en la sección 3.4, en la página 69, fue que el nodo donde se inicia la construcción del árbol queda fijo, pues corresponde al nodo necesario que haga verdadero el enunciado **if** y es en la última línea de la secuencia de enunciados donde se hace la llamada respectiva para crear la estructura.

El proceso de construcción del árbol de etiquetas lo realiza el método **Inunda()**, el cual fue añadido a los métodos originales del GPSR. La llamada a este método se da en la línea 946, el cual está definido en la línea 1170 y cuyas instrucciones que contiene son:

[9]Se eligió este nodo debido a que la mayoría de los ejemplos del GPSR utilizan al primer nodo colocado en la topología como cliente; entonces se busca un nodo cercano.

```
1170 void RoutingProtocol::Inunda(){
1171   /*Iterador y obtención de la tabla de vecinos que calcula GPSR*/
1172   std::map<Ipv4Address, std::pair<Vector, Time> >::iterator i;
1173   /*En la parte first del mapa viene la dirección de los vecinos*/
1174   std::map<Ipv4Address, std::pair<Vector, Time> > aux = *(m_neighbors.get_NTable
          ());
1175   for (i = aux.begin (); !(i == aux.end ()); i++){
1176     for (std::map<Ptr<Socket>, Ipv4InterfaceAddress>::const_iterator j =
            m_socketAddresses.begin (); j != m_socketAddresses.end (); ++j){
1177
1178       Ptr<Socket> socket = j->first;
1179
1180 //    Estructura de cabecera de etiquetas TagHeader::TagHeader (uint16_t mode,
      uint64_t tag, uint16_t children)
1181 //    Se tiene la etiqueta inicial en m_ipv4->GetObject<Node>()->GetObject<
      NodeTag>()->getTag()
1182       TagHeader tagHeader (1, m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->
              getTag(),m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->getChi());
1183
1184       //Sub-modo 1 - Pregunta si se quiere etiquetar
1185       //Sub-modo 2 - Contesta que debe ser etiquetado
1186       //Sub-modo 3 - Realiza el etiquetado
1187
1188       Ptr<Packet> packet = Create<Packet> ();
1189       packet->AddHeader (tagHeader);
1190       TypeHeader tHeader (GPSRTYPE_TAG);
1191       packet->AddHeader (tHeader);
1192       // Send to all-hosts broadcast if on /32 addr, subnet-directed otherwise
1193       Ipv4Address destination=i->first;
1194       //La llamada del socket mediante SendTo se detecta en RecvGPSR
1195       socket->SendTo (packet, 0, InetSocketAddress (destination, GPSR_PORT));
1196     }
1197   }
1198 }
```

De esta manera el objetivo del método **Inunda()** es que a partir del nodo donde se le llamó, se construya el árbol de etiquetas y tome a dicho nodo como la raíz de la estructura.

Se puede apreciar en la línea 1174 que la variable **aux** está siendo inicializada con los valores de la tabla donde el algoritmo GPSR guarda los vecinos que ha descubierto para el respectivo nodo. Para poder obtener dichos valores se modificaron los ficheros parte del modelo del GPSR: **gpsr-ptable.cc** y **gpsr-ptable.h** donde se declaró e instanció, respectivamente, el método **get_NTable()**.

Enseguida, se puede observar que en las líneas 1175 y 1176 se ejecutan un par de ciclos que ayudan a recorrer la tabla de vecinos del respectivo nodo. Además, se arma un paquete con un tipo de cabecera especial: **TagHeader**.

La cabecera que se comenta, se agregó y declaró de manera análoga a las que ya contaba el modelo GPSR en sus ficheros **gpsr-packet.cc** y **gpsr-packet.h**. Es mediante el uso de **TagHeader**, donde se añade la información de la etiqueta que tiene el padre, así mismo se agrega el estado en que se encuentra el etiquetado

y se anexa el número de hijos que tiene hasta ese momento el nodo padre.

El envío de la información que sirve para etiquetar se realiza concretamente en la línea 1195. Este paquete será interpretado por cada vecino del nodo etiquetador. Para dicha interpretación, se llevó a cabo otra modificación en un método que se activa cuando un nodo recibe un paquete, en el algoritmo GPSR. Dicho ajuste se describirá a continuación.

El método que interpreta los mensajes es **RecvGPSR()**, el cual originalmente sólo manejaba dos tipos de cabeceras: **HelloHeader** y **PositionHeader**. La modificación correspondió a que si se detectaba que un nodo recibía un paquete con cabecera del tipo **TagHeader** entonces se verificaba si se debía realizar el proceso de etiquetado o no.

Las líneas del método de recepción modificado comienzan en la línea 491, las cuales se presentan a continuación:

```
491 void
492 RoutingProtocol::RecvGPSR (Ptr<Socket> socket)
493 {
494   NS_LOG_FUNCTION (this << socket);
495   Address sourceAddress;
496   Ptr<Packet> packet = socket->RecvFrom (sourceAddress);
497
498   TypeHeader tHeader (GPSRTYPE_HELLO);
499   packet->RemoveHeader (tHeader);
500   if (!tHeader.IsValid ())
501     {
502       NS_LOG_DEBUG ("GPSR␣message␣" << packet->GetUid () << "␣with␣unknown␣type␣
          received:␣" << tHeader.Get () << ".␣Ignored");
503       return;
504     }
505
506   switch(tHeader.Get()){
507     case GPSRTYPE_HELLO:{
508       HelloHeader hdr;
509       packet->RemoveHeader (hdr);
510       Vector Position;
511       Position.x = hdr.GetOriginPosx ();
512       Position.y = hdr.GetOriginPosy ();
513       InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (
          sourceAddress);
514       Ipv4Address sender = inetSourceAddr.GetIpv4 ();
515       Ipv4Address receiver = m_socketAddresses[socket].GetLocal ();
516
517       UpdateRouteToNeighbor (sender, receiver, Position);
518       break;
519     }
520     case GPSRTYPE_POS:{
521       break;
522     }
523     case GPSRTYPE_TAG:{
524       TagHeader hdr;
525       packet->RemoveHeader(hdr);
```

```
526       uint16_t i_mode = hdr.GetMode();
527       uint64_t i_tag = hdr.GetTag();
528       uint16_t i_children = hdr.GetChildren();
529       if(i_mode==1&&m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->getSec()==1)
              {
530         InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (
                sourceAddress);
531         Ipv4Address sender = inetSourceAddr.GetIpv4 ();
532
533 //        NS_LOG_LOGIC ("Quiero que me etiqueten SubMode1 "<< sender);
534
535         i_mode++;                              //Aumenta para el submodo 2
536         m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->setSec(i_mode);
537         TagHeader tagHeader (i_mode, i_tag, i_children); //Mantiene la etiqueta
                que llegó
538         Ptr<Packet> packet = Create<Packet> ();
539         packet->AddHeader (tagHeader);
540         TypeHeader tHeader (GPSRTYPE_TAG);
541         packet->AddHeader (tHeader);
542         // Enviamos respuesta al emisor
543         socket->SendTo (packet, 0, InetSocketAddress (sender, GPSR_PORT));
544       }
545       else if(i_mode==2){
546         InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (
                sourceAddress);
547         Ipv4Address sender = inetSourceAddr.GetIpv4 ();
548 //        NS_LOG_LOGIC ("Te voy a etiquetar- SubMode2 "<<sender);
549
550         //Coloca secuencia en 2 y aumenta el hijo en 1 también
551         i_mode++;
552         //El nodo etiquetador tendrá ahora un hijo más
553         uint16_t M_children = m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->
                getChi();
554         M_children++;
555         m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->setChi(M_children);
556         /*v----- En esta cabecera irán los valores (modo 3 [etiquetador],
                etiqueta del padre) */
557         TagHeader tagHeader (i_mode, i_tag, M_children);//Ya contiene los valores
                 ready para etiquetar
558         Ptr<Packet> packet = Create<Packet> ();
559         packet->AddHeader (tagHeader);
560         TypeHeader tHeader (GPSRTYPE_TAG);
561         packet->AddHeader (tHeader);
562         // Enviamos respuesta a quien etiquetará
563         socket->SendTo (packet, 0, InetSocketAddress (sender, GPSR_PORT));
564       }
565       else if(i_mode==3){
566         InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (
                sourceAddress);
567         Ipv4Address sender = inetSourceAddr.GetIpv4 ();
568 //        NS_LOG_LOGIC ("Me están etiquetando- SubMode3 "<<sender);
569
570         m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->setTag(i_tag*10+
                i_children);
571
572         //Se quiere ver que etiqueta tendría después de etiquetar
573         NS_LOG_LOGIC ("Me␣están etiquetando- SubMode3 "<<sender << "␣Eti_POST:␣"
```

```
            << m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->getTag());
574         //Pondrá a ese nodo en sub-modo 3 ya está etiquetado y puede etiquetar
575         m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->setSec(i_mode);
576         Inunda();
577       }
578       else{
579       }
580       break;
581     }
582   }
583
584 }
```

Con esta modificaciones se logra tener implementada la estructura del *Position-Tree*. A continuación, se revisará un ejemplo de una red de nodos donde se aplica el algoritmo de encaminamiento árbol de etiquetas.

El ejemplo que se presenta lleva por nombre **`mygpsr-test14.cc`**. Para éste, se manejaron los parámetros por defecto que se muestran en la tabla 3.2, en la página 61.

Observación 23. En este ejemplo, los nombres identificadores de los nodos se ajustan para que coincidan con el orden en que éstos se colocan en la topología del arreglo.

La topología que se manejó en la simulación del ejemplo se puede observar en la figura 3.12.

Nodos en la simulación: 49.
Ancho de cuadrícula: 7.
Servidor: Nodo-49 Pos.(600,600).

Cliente: Nodo-1 Pos.(0,0).
Intervalo entre paquetes: 0.5 s.

Nodos removidos:

- Nodo-3 Pos.(200,0).
- Nodo-4 Pos.(300,0).
- Nodo-5 Pos.(400,0).
- Nodo-6 Pos.(500,0).
- Nodo-7 Pos.(600,0).
- Nodo-8 Pos.(0,100).
- Nodo-12 Pos.(400,100).
- Nodo-13 Pos.(500,100).
- Nodo-14 Pos.(600,100).
- Nodo-15 Pos.(0,200).
- Nodo-17 Pos.(200,200).
- Nodo-22 Pos.(0,300).

- Nodo-24 Pos.(200,300).
- Nodo-25 Pos.(300,300).
- Nodo-26 Pos.(400,300).
- Nodo-27 Pos.(500,300).
- Nodo-29 Pos.(0,400).
- Nodo-36 Pos.(0,500).
- Nodo-38 Pos.(200,500).
- Nodo-39 Pos.(300,500).
- Nodo-40 Pos.(400,500).
- Nodo-41 Pos.(500,500).
- Nodo-42 Pos.(600,500).
- Nodo-43 Pos.(0,600).

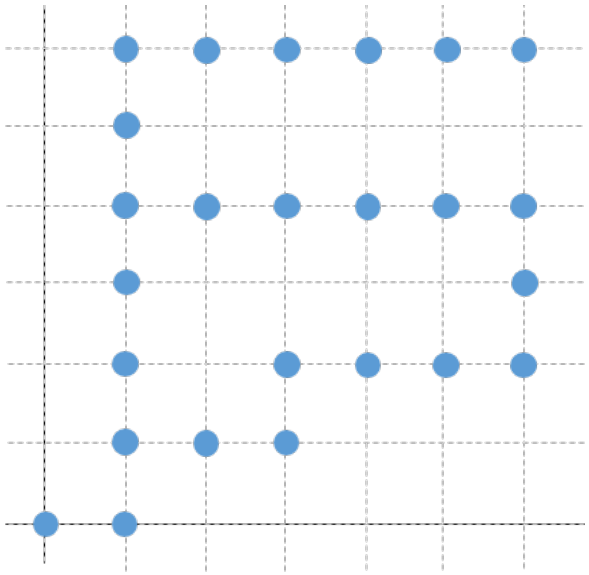


Figura 3.12: Topología de **`mygpsr-test14.cc`**

En la siguiente sección se comentará la manera en que se realiza el proceso de recuperación usando esta estructura de datos.

Estrategia de recuperación

Cuando se alcanza un máximo local en el reenvío de paquetes, el algoritmo implementado llama al método **`RecoveryMode()`**. En el ejemplo que se describió en la sección anterior, **`mygpsr-test14.cc`**, la secuencia de reenvío, según los identificadores de los nodos, que se siguió en el modo tradicional voraz fue:

$1 -> 2 -> 9 -> 10 -> 11 -> 18 -> 19 -> 20 -> 21 -> 28 -> 35$

Se aprecia en la figura 3.12, que al seguirse la secuencia anterior es en el nodo-35 con posición (600,400) que se alcanza un máximo local y debe activarse la estrategia de recuperación correspondiente.

Fue por esto que, para el método **RecoveryMode()** la única modificación que se realizó fue en la línea donde se debía elegir cual dirección debía ser el destino de acuerdo a la estrategia de recuperación que se describió en el código 3.3. Originalmente se tenía algo como:

```
Ipv4Address nextHop = m_neighbors.BestAngle (previousHop, myPos);
```

La línea anterior hacía uso del método **BestAngle()** con el que se calculaba cual era el primer nodo que se hallaba en sentido de las manecillas del reloj utilizando la regla de la mano derecha. Esta línea se sustituyó por:

```
Ipv4Address nextHop = mejor_etiqueta(m_locationService->GetTag(dst));
```

Con la sustitución anterior se hace uso del método **mejor_etiqueta()** que con base en el algoritmo utiliza como parámetro la etiqueta que tiene el nodo que llama al método y así tomar la decisión del encaminamiento. Las líneas de código del método se describen a continuación, éstas son agregadas al modelo GPSR:

```
1200 /* Mediante la siguiente función mejor_etiqueta se busca devolver el Ipv4Address
1201 que debe usarse como nodo intermediaria, para que a través de ella se haga
1202 el reenvío en recovery usando el árbol de etiquetas*/
1203 Ipv4Address RoutingProtocol::mejor_etiqueta(uint64_t dstTag){
1204   uint64_t aux_control = m_ipv4->GetObject<Node>()->GetObject<NodeTag>()->getTag
         ();
1205   uint64_t aux_dstTag = dstTag;
1206   uint64_t aux_dstTag2 =dstTag;
1207   int tc = sizeOfTag(aux_control);
1208   int td = sizeOfTag(dstTag);
1209   int dif;
1210   uint64_t find;
1211   if(td<=tc){                          //ASCENSO - Sube al padre
1212     find = aux_control/10;
1213   }else{      /*td>tc*/
1214     /*Checando si vamos a subir o no*/
1215     dif=td-tc;
1216     for(int d=0;d<dif;d++)
1217       aux_dstTag=aux_dstTag/10;
1218     if(aux_dstTag==aux_control){      //DESCENSO -  Es prefijo común
1219       for(int c=1; c<dif;c++)
1220         aux_dstTag2=aux_dstTag2/10;
1221       find = aux_dstTag2;
1222     }else{                           //ASCENSO - Sube al padre
1223       find = aux_control/10;
1224     }
1225   }
1226   //Iterador y obtención de la tabla de vecinos que calcula GPSR
1227   std::map<Ipv4Address, std::pair<Vector, Time> >::iterator i;
1228   //En la parte first del mapa viene la dirección de los vecinos
```

```
1229   std::map<Ipv4Address, std::pair<Vector, Time> > aux = *(m_neighbors.get_NTable
           ());
1230
1231   for (i = aux.begin (); !(i == aux.end ()); i++){
1232     //abbreviation nbr = neighbor
1233     Ipv4Address nbr = i->first;
1234     if(m_locationService->GetTag(nbr)==find)
1235     return nbr;
1236   }
1237
1238   Ipv4Address eti = Ipv4Address::GetZero();
1239   return eti;
1240 }
```

Las líneas anteriores implementan la estrategia de recuperación que se describió en el listado 3.3. Este modo empleado para salir del máximo local se vale de las etiquetas que se han colocado previamente con la estructura *PositionTree* y es sobre la consulta de las etiquetas de los nodos vecinos que se toma la decisión para el reenvío de paquetes.

Se puede apreciar en las líneas 1207 y 1208 del método **mejor_etiqueta()** que se hace uso de otro método que se agregó, **sizeOfTag()**; el cual tiene como finalidad obtener la longitud de la etiqueta del correspondiente nodo que haga el llamado a este método. Recordemos que dicha longitud expresa a qué nivel debajo de la raíz se encuentra cierto nodo. El código del método se expresa a continuación:

```
1242 int RoutingProtocol::sizeOfTag(uint64_t n){
1243   int size=0;
1244   while(n/10>=1){
1245     size++;
1246     n=n/10;
1247   }
1248   return size+1;
1249 }
```

Con las modificaciones y añadiduras que se han mostrado en esta sección, queda implementado el algoritmo árbol de etiquetas en el simulador ns-3.

Conclusiones

Con el material que se revisó en el capítulo anterior, se mostró una forma de implementar el algoritmo árbol de etiquetas haciendo una modificación al GPSR. De esta manera, se contribuye a que en ns-3.22 se pueda hacer uso del trabajo propuesto en [CMT07].

Experiencias y aprendizaje

Gran parte de la experiencia que dejó el trabajar con el tema de tesis desarrollado fue primeramente el acercamiento hacia las redes de computadoras. Como se abordó un poco en la introducción, muchas veces se tiene o conoce la idea intuitiva de lo que es una red de computadoras, pero llevar a cabo la revisión minuciosa de cada uno de los elementos, protocolos, algoritmos y teorías de redes resulta un poco más laboriosa.

Una vez que se abordó el marco teórico sobre el tema expuesto, aparecieron una serie de retos al revisar algunas publicaciones que desarrollan conceptos similares. En esta parte se requirió serenidad y paciencia para comprender los trabajos en los que se basa el árbol de etiquetas. De igual manera, seguir el flujo de los códigos donde se implementaban los conceptos teóricos del algoritmo GPSR requirió de una profunda revisión. Sin embargo, una vez realizado dicho proceso se pudieron localizar de manera práctica los lugares donde se debían crear o modificar las clases de los objetos necesarios.

De la misma forma, cuando se revisaron contribuciones acerca de los distintos algoritmos que se han desarrollado para el ruteo de paquetes en redes, saltaron a la vista los diversos simuladores de redes que existen. Se pudo analizar que existían varios tipos de *software*, desde los de código privativo que requerían de la adquisición de una patente, hasta los de licencia libre con opción de modificación y desarrollo por parte de los usuarios.

Al revisar los diversos tipos de simuladores se encontró que en todos ellos no

era necesario partir de ceros para crear una simulación; sino que en cada *software* existían trabajos que se podían tomar como base para dichas reproducciones. Por esta razón se consideró el trabajo presentado en [FCoVa12], el cual al ser desarrollado en ns-3 ofrecía gran ventaja por ser un simulador con licencia del tipo *General Public License* (GNU GPL) y facilitaba la modificación del algoritmo referenciado.

Desarrollo mediante el simulador ns-3

La opción de implementar en ns-3, resultó bastante cómoda, ya que al ser basado el simulador en C++, la idea de un lenguaje con Programación Orientada a Objetos permite una abstracción sencilla para asociar las diversas clases con los correspondientes dispositivos y elementos de una red. A la par de esto se aprovechó el concepto de extensión y herencia de clases para asociarlo con la elaboración de diversas topologías de red.

De igual manera, el simulador ns-3 cuenta con una amplia comunidad que trabaja en dar soporte periódico mediante una serie de actualizaciones, por lo que el *software* se encuentra en continuo desarrollo. Así, la razón principal por la cual el trabajo usado del GPSR tuvo primeramente que adaptarse a la versión 22 del simulador, ocurrió por el desfase de versiones, por lo que el código de [FCoVa12] no se ejecutaba tenazmente en la edición más actual.

Objetivos alcanzados

A manera de un breve sumario sobre los objetivos que se obtuvieron al desarrollar el trabajo de tesis presentado encontramos los siguientes:

- En el simulador ns-3.22 queda implementado el árbol de etiquetas.
- Se da la metodología de cómo contribuir al proyecto ns-3 para implementar un algoritmo.
- De manera práctica, queda justificado que el árbol de etiquetas puede recuperarse cuando se alcanza un máximo local.

Trabajo futuro

Algunos detalles que se pueden mejorar para que el algoritmo árbol de etiquetas sea más óptimo son:

1. Buscar la factibilidad de poder construir el árbol de etiquetas de manera previa a la ejecución de las simulaciones. Lo deseable es que la estructura se cree como si fuera una aplicación de tipo *broadcast* que exclusivamente permita descubrir las rutas y vecinos de los nodos.

2. En relación con lo comentado en el punto anterior, la elección de la raíz del *PositionTree* quedará a disposición para que pueda ser elegida arbitrariamente.

3. Se puede mejorar el tipo del campo **`m_Tag`**, ya que al ser de tipo numérico **`uint_64`**, el máximo número que puede expresarse es: $2^{64} - 1$, con lo cual aproximadamente se pueden emplear etiquetas de hasta 20 dígitos. Esto significa que la estructura del árbol de etiquetas usando este tipo de datos, no podrá tener más de 20 niveles. Para esto, podría manejarse un etiquetado de tipo alfabético mediante el uso de arreglos de caracteres creados de manera dinámica, de acuerdo al tamaño de la etiqueta necesaria y guardar la referencia a la dirección de memoria en el campo **`m_Tag`**.

4. Se puede trabajar en un proceso de acortamiento de rutas cuando se esté en la estrategia de recuperación y se apliquen las operaciones de ascenso. Se debe aprovechar que si el nodo que entra al modo de recuperación tiene como vecino a otro cuya etiqueta tenga mayor coincidencia con la del nodo destino, entonces se debe reenviar el paquete a este nodo y evitar subir al ancestro.

El correspondiente código funcional del Encamiento Voraz Perimétrico Local, adaptado a la versión 22 del ns-3, que se obtiene al realizar las modificaciones y ajustes mostradas hasta la página 68, se colocó en el *link* `https://github.com/edlobo127/gpsr-ns-3.22`, donde puede descargarse e incorporarse para ejecutar el algoritmo GPSR en simulaciones de ns-3.

Una vez revisado el código compartido se puede trabajar en la elaboración de su correspondiente documentación para que se pueda anexar al simulador; ya que, haciendo una revisión sobre los algoritmos de encaminamiento que maneja ns-3, resulta que este *software* sólo tiene integrados de manera oficial cuatro protocolos

de enrutamiento: OLSR, AODV, DSDV y DSR, los cuales fueron brevemente descritos en la introducción del capítulo 2.

Por lo anterior, cuando se realizan simulaciones y debe elegirse el tipo de encaminamiento deseado, los parámetros disponibles para el método **SetRoutingHelper()** del auxiliar *InternetStackHelper* son: **olsr**, **aodv**, **dsdv** o **dsr**.

Sin embargo, al momento de ejecutar los ejemplos que se han comentado en este documento fue posible utilizar **gpsr** como parámetro para el método **SetRoutingHelper()**, como se muestra en el siguiente código:

```
InternetStackHelper stack;
stack.SetRoutingHelper (gpsr);
```

Por lo anterior, con el objetivo de contribuir al proyecto ns-3.22, un trabajo a futuro es renombrar los códigos en la definición de sus clases para que al usar el algoritmo árbol de etiquetas se haga de manera adecuada, ya que en la implementación conseguida se utilizó como parámetro **gpsr**.

La finalidad de la implementación presentada como objetivo de este trabajo de tesis es que se pueda construir un nuevo módulo titulado *tagTree*, ya que este algoritmo es novedoso respecto a los ya incluidos en ns-3. De esta manera, se puede trabajar en una depuración y proceso de optimización, para que se someta a revisión y junto a su correspondiente documentación se reconozca a este tipo de protocolo como uno más del simulador y otros usuarios puedan usar el algoritmo.

Así, cuando se efectúen simulaciones y se desee usar al árbol de etiquetas, se pueda colocar **tagTree** como parámetro. Se pretende entonces tener:

```
InternetStackHelper stack;
stack.SetRoutingHelper (tagTree);
```

Apéndice A

Instalación del simulador ns-3.22

El simulador ns-3 es desarollado principalmente para plataformas GNU/Linux. Los mínimos requisitos para instalar el simulador son: el compilador **g++** y el intérprete de Python.

Para el equipo en el que se instaló el ns-3, sólo se contaba con el intérprete de Python, por lo que tuvo que realizarse el paso que se indica en el número 4. En caso de que el usuario tenga en su dispositivo el compilador gcc, dicho paso puede omitirse.

A continuación se describirá el procedimiento usado para instalar el simulador ns-3 en su versión 22. Se asume que el usuario está trabajando en Linux o un ambiente de emulación de Linux. Para realizar el proceso, se usará una Terminal o Consola. Estos pasos están tomados del Tutorial del ns-3. [Nat15b, p.10]

1. Estando en algún directorio preferido, (como recomendación la carpeta personal), se procede a crear una carpeta mediante el comando *mkdir* bajo el nombre que se desee y se accede a ella con *cd*.

```
$ mkdir simulador
$ cd simulador
```

2. Haciendo uso del comando *wget* y con la opción *–no-check-certificate*, se procede a realizar la descarga del sitio desarrollador del simulador `http://www.nsam.org/release/ns-allinone-3.22.tar.bz2`. El tamaño de esta versión es de 23 MB aproximadamente.

```
$ wget --no-check-certificate http://www.nsnam.org/release/
    ns-allinone-3.22.tar.bz2
```

** En caso de que no se pueda realizar la descarga utilizando el paso anterior, es posible acceder a través de un navegador de internet al enlace `https://www.nsnam.org./ns-3-22/download/` y obtener el archivo comprimido dando clic en la sección indicada. Una vez obtenido el archivo, hay que colocarlo en la carpeta que se creó, en este caso **simulador**.

3. Dado que el archivo se ha guardado en la carpeta **simulador** en un formato comprimido, procedemos a realizar la descompresión mediante el comando *tar* y las opciones *xjf*, las cuales significan *x* - Extraer, *j* - Usando el algoritmo *LZMA* y *f* - El archivo.

```
$ tar xjf ns-allinone-3.22.tar.bz2
```

4. * Este paso puede omitirse si el equipo donde se está realizando la instalación cuenta previamente con el compilador de C++. De lo contrario, para obtener el compilador *g++* debe ejectuarse:

```
$ sudo apt-get install g++
```

   Ahora el equipo dispondrá de este compilador y podrá ejecutar *scripts* que se creen en el lenguaje orientado a objetos C++.

5. Hasta este punto, antes de ejecutar el siguiente paso, podemos considerar que se cuenta con todos los ficheros necesarios y que se está listo para construir la base de la distribución *ns-3*. Para esto, se accede mediante *cd* a la carpeta donde se ha descomprimido el archivo y se ejecuta el *script* de instalación basado en el lenguaje *python* que se denomina **build.py**.

```
$ cd ns-allinone-3.22
$ ./build.py --enable-examples --enable-tests
```

   Este proceso puede demorar algunos minutos dependiendo de las características que tenga el equipo. Una vez finalizado este proceso de contrucción[Nat15b, p.12], aparece el siguiente mensaje:

```
Waf: Leaving directory `/home/elopez/simulador/ns-allinone
    -3.22/ns-3.22/build'
'build' finished successfully (16m34.520s)
```

   Además se muestran los módulos que trae por defecto el simulador ns-3 y que se describen en 3.2.

6. Para llevar a cabo el siguiente paso, el cual podemos describir como un proceso de examinación (*testing*)[Nat15b, p.18], debe accederse a la carpeta que se ha creado **ns-3.22** y se procede a correr un *script* que realizará pruebas sobre el simulador, para esto se ejecuta:

```
$ cd ns-3.22
$ ./test.py -c core
```

   Nuevamente este proceso puede demorar algunos minutos. Después de un tiempo el mensaje que aparece es el siguiente:

```
193 of 196 tests passed (193 passed, 3 skipped, 0 failed, 0
    crashed, 0 valgrind errors)
List of SKIPped tests: ns3-tcp-cwnd
    ns3-tcp-interoperability
    nsc-tcp-loss
```

   Nótese que se indicó que tres pruebas fueron omitidas, sin embargo el resto de ellas ($\frac{193}{196}$) se realizaron correctamente, por lo que el simulador se encuentra disponible para correr *scripts*.

7. Como último paso, se procede a ejecutar el primer programa en ns-3, esto mediante el comando:

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

   Lo que imprime en la salida de la consola:

```
Hello Simulator
```

   ¡Enhorabuena! El ns-3 se ha instalado correctamente y es posible realizar simulaciones con esta herramienta.

   "*Congratulations! You are now an ns-3 user!*"[Nat15b, p. 19]

Apéndice B

Ejemplos del tutorial de ns-3

Los ejemplos utilizados para ilustrar los primeros pasos en el trabajo del simulador *ns-3* son los códigos anexados en la carpeta: **.../ns-3.22/examples/tutorial**.

first.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
```

```
  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

second.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
```

```
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/csma-module.h"
20 #include "ns3/internet-module.h"
21 #include "ns3/point-to-point-module.h"
22 #include "ns3/applications-module.h"
23 #include "ns3/ipv4-global-routing-helper.h"
24
25 // Default Network Topology
26 //
27 //       10.1.1.0
28 // n0 -------------- n1   n2   n3   n4
29 //    point-to-point  |    |    |    |
30 //                    ================
31 //                      LAN 10.1.2.0
32
33
34 using namespace ns3;
35
36 NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");
37
38 int
39 main (int argc, char *argv[])
40 {
41   bool verbose = true;
42   uint32_t nCsma = 3;
43
44   CommandLine cmd;
45   cmd.AddValue ("nCsma", "Number␣of␣\"extra\"␣CSMA␣nodes/devices", nCsma);
46   cmd.AddValue ("verbose", "Tell␣echo␣applications␣to␣log␣if␣true", verbose);
47
48   cmd.Parse (argc,argv);
49
50   if (verbose)
51     {
52       LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
53       LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
54     }
55
56   nCsma = nCsma == 0 ? 1 : nCsma;
57
58   NodeContainer p2pNodes;
59   p2pNodes.Create (2);
60
61   NodeContainer csmaNodes;
62   csmaNodes.Add (p2pNodes.Get (1));
63   csmaNodes.Create (nCsma);
64
65   PointToPointHelper pointToPoint;
66   pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
67   pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
68
```

```
69   NetDeviceContainer p2pDevices;
70   p2pDevices = pointToPoint.Install (p2pNodes);
71
72   CsmaHelper csma;
73   csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
74   csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
75
76   NetDeviceContainer csmaDevices;
77   csmaDevices = csma.Install (csmaNodes);
78
79   InternetStackHelper stack;
80   stack.Install (p2pNodes.Get (0));
81   stack.Install (csmaNodes);
82
83   Ipv4AddressHelper address;
84   address.SetBase ("10.1.1.0", "255.255.255.0");
85   Ipv4InterfaceContainer p2pInterfaces;
86   p2pInterfaces = address.Assign (p2pDevices);
87
88   address.SetBase ("10.1.2.0", "255.255.255.0");
89   Ipv4InterfaceContainer csmaInterfaces;
90   csmaInterfaces = address.Assign (csmaDevices);
91
92   UdpEchoServerHelper echoServer (9);
93
94   ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
95   serverApps.Start (Seconds (1.0));
96   serverApps.Stop (Seconds (10.0));
97
98   UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
99   echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
100  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
101  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
102
103  ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
104  clientApps.Start (Seconds (2.0));
105  clientApps.Stop (Seconds (10.0));
106
107  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
108
109  pointToPoint.EnablePcapAll ("second");
110  csma.EnablePcap ("second", csmaDevices.Get (1), true);
111
112  Simulator::Run ();
113  Simulator::Destroy ();
114  return 0;
115 }
```

Apéndice C

Contenido del módulo etiqueta

La estructura esquemática que se genera al crear el módulo **etiqueta** es la siguiente:

```
+etiqueta
  +-doc
     |_etiqueta.rst
  +-examples
     |-etiqueta-example.cc
     |_wscript
  +-helper
     |-etiqueta-helper.cc
     |_etiqueta-helper.h
  +-model
     |-etiqueta.cc
     |_etiqueta.h
  +-test
     |_etiqueta-test-suite.cc
     |_wscript
```

A continuación se despliegan los códigos generados, los cuales permanecieron con su estructura esquelética generada por defecto, a excepción de **etiqueta.cc** y **etiqueta.h** que fueron manipulados agregándoles los campos necesarios descritos en 3.2.

C.1. doc

etiqueta.rst

```
Example Module Documentation
----------------------------

.. include:: replace.txt
.. highlight:: cpp

.. heading hierarchy:
   ------------- Chapter
   ************* Section (#.#)
   ============= Subsection (#.#.#)
   ############# Paragraph (no number)

This is a suggested outline for adding new module documentation to |ns3|.
See ``src/click/doc/click.rst`` for an example.

The introductory paragraph is for describing what this code is trying to
model.

For consistency (italicized formatting), please use |ns3| to refer to
ns-3 in the documentation (and likewise, |ns2| for ns-2).  These macros
are defined in the file ``replace.txt``.

Model Description
*****************

The source code for the new module lives in the directory ``src/etiqueta``.

Add here a basic description of what is being modeled.

Design
======

Briefly describe the software design of the model and how it fits into
the existing ns-3 architecture.

Scope and Limitations
=====================

What can the model do?  What can it not do?  Please use this section to
describe the scope and limitations of the model.

References
==========

Add academic citations here, such as if you published a paper on this
model, or if readers should read a particular specification or other work.

Usage
*****

This section is principally concerned with the usage of your model, using
```

```
the public API.  Focus first on most common usage patterns, then go
into more advanced topics.

Building New Module
===================

Include this subsection only if there are special build instructions or
platform limitations.

Helpers
=======

What helper API will users typically use?  Describe it here.

Attributes
==========

What classes hold attributes, and what are the key ones worth mentioning?

Output
======

What kind of data does the model generate?  What are the key trace
sources?   What kind of logging output can be enabled?

Advanced Usage
==============

Go into further details (such as using the API outside of the helpers)
in additional sections, as needed.

Examples
========

What examples using this new code are available?  Describe them here.

Troubleshooting
===============

Add any tips for avoiding pitfalls, etc.

Validation
**********

Describe how the model has been tested/validated.  What tests run in the
test suite?  How much API and code is covered by the tests?  Again,
references to outside published work may help here.
```

C.2. examples

etiqueta-example.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */

#include "ns3/core-module.h"
#include "ns3/etiqueta-helper.h"

using namespace ns3;


int
main (int argc, char *argv[])
{
  bool verbose = true;

  CommandLine cmd;
  cmd.AddValue ("verbose", "Tell␣application␣to␣log␣if␣true", verbose);

  cmd.Parse (argc,argv);

  /* ... */

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

wscript

```
# -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8;
    -*-

def build(bld):
    obj = bld.create_ns3_program('etiqueta-example', ['etiqueta'])
    obj.source = 'etiqueta-example.cc'
```

C.3. helper

etiqueta-helper.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */

#include "etiqueta-helper.h"

namespace ns3 {

/* ... */


}
```

etiqueta-helper.h

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
#ifndef ETIQUETA_HELPER_H
#define ETIQUETA_HELPER_H

#include "ns3/etiqueta.h"

namespace ns3 {

/* ... */

}

#endif /* ETIQUETA_HELPER_H */
```

C.4. model

etiqueta.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */

#include "etiqueta.h"
#include "ns3/uinteger.h"

namespace ns3 {

NS_OBJECT_ENSURE_REGISTERED (NodeTag);

TypeId
NodeTag::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::NodeTag")
    .SetParent<Object> ()
    .AddConstructor<NodeTag> ()
    .AddAttribute ("DefaultTag","Default␣tag␣set␣in␣the␣nodes,␣'0'",
                   UintegerValue (0),
                   MakeUintegerAccessor (&NodeTag::m_Tag),
                   MakeUintegerChecker<uint64_t> ())
    .AddAttribute ("DefaultSequence","Default␣sequence␣in␣the␣nodes,␣'1'␣no␣
        tagged␣even[I␣want␣to␣be␣labeled]",
                   UintegerValue (1),
                   MakeUintegerAccessor (&NodeTag::m_Sec),
                   MakeUintegerChecker<uint16_t> ())
    .AddAttribute ("DefaultChildren","Default␣number␣of␣children,␣'0'",
                   UintegerValue (0),
                   MakeUintegerAccessor (&NodeTag::m_Chi),
                   MakeUintegerChecker<uint16_t> ())
    ;

  return tid;
}

NodeTag::NodeTag ()
```

```
{
}

NodeTag::~NodeTag ()
{
}

uint64_t
NodeTag::getTag(){
  return m_Tag;
}
void
NodeTag::setTag(uint64_t valueTag){
    m_Tag=valueTag;
}
uint16_t
NodeTag::getSec(){
  return m_Sec;
}
void
NodeTag::setSec(uint16_t valueTag){
    m_Sec=valueTag;
}
uint16_t
NodeTag::getChi(){
  return m_Chi;
}
void
NodeTag::setChi(uint16_t valueTag){
    m_Chi=valueTag;
}

}
```

etiqueta.h

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
#ifndef ETIQUETA_H
#define ETIQUETA_H

#include "ns3/object.h"

namespace ns3 {

class NodeTag : public Object
{
public:
  static TypeId GetTypeId (void);

  NodeTag ();
  virtual ~NodeTag ();

  uint64_t getTag();
  void setTag(uint64_t valueTag);
  uint16_t getSec();
  void setSec(uint16_t valueSec);
```

```
21   uint16_t getChi();
22   void setChi(uint16_t valueChi);
23
24 private:
25   uint64_t        m_Tag;          ///<  Indica la etiqueta que tiene el nodo
26   uint16_t        m_Sec;          ///<  Indica la secuencia de etiquetado
27   uint16_t        m_Chi;          ///<  Indica el número de hijos de un nodo
28
29 };
30
31 /* ... */
32
33 }
34
35 #endif /* ETIQUETA_H */
```

C.5. test

etiqueta-test-suite.cc

```
1  /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2
3  // Include a header file from your module to test.
4  #include "ns3/etiqueta.h"
5
6  // An essential include is test.h
7  #include "ns3/test.h"
8
9  // Do not put your test classes in namespace ns3.  You may find it useful
10 // to use the using directive to access the ns3 namespace directly
11 using namespace ns3;
12
13 // This is an example TestCase.
14 class EtiquetaTestCase1 : public TestCase
15 {
16 public:
17   EtiquetaTestCase1 ();
18   virtual ~EtiquetaTestCase1 ();
19
20 private:
21   virtual void DoRun (void);
22 };
23
24 // Add some help text to this case to describe what it is intended to test
25 EtiquetaTestCase1::EtiquetaTestCase1 ()
26   : TestCase ("Etiqueta␣test␣case␣(does␣nothing)")
27 {
28 }
29
30 // This destructor does nothing but we include it as a reminder that
31 // the test case should clean up after itself
32 EtiquetaTestCase1::~EtiquetaTestCase1 ()
33 {
34 }
35
```

```
36 //
37 // This method is the pure virtual method from class TestCase that every
38 // TestCase must implement
39 //
40 void
41 EtiquetaTestCase1::DoRun (void)
42 {
43   // A wide variety of test macros are available in src/core/test.h
44   NS_TEST_ASSERT_MSG_EQ (true, true, "true␣doesn't␣equal␣true␣for␣some␣reason");
45   // Use this one for floating point comparisons
46   NS_TEST_ASSERT_MSG_EQ_TOL (0.01, 0.01, 0.001, "Numbers␣are␣not␣equal␣within␣
         tolerance");
47 }
48
49 // The TestSuite class names the TestSuite, identifies what type of TestSuite,
50 // and enables the TestCases to be run.  Typically, only the constructor for
51 // this class must be defined
52 //
53 class EtiquetaTestSuite : public TestSuite
54 {
55 public:
56   EtiquetaTestSuite ();
57 };
58
59 EtiquetaTestSuite::EtiquetaTestSuite ()
60   : TestSuite ("etiqueta", UNIT)
61 {
62   // TestDuration for TestCase can be QUICK, EXTENSIVE or TAKES_FOREVER
63   AddTestCase (new EtiquetaTestCase1, TestCase::QUICK);
64 }
65
66 // Do not forget to allocate an instance of this TestSuite
67 static EtiquetaTestSuite etiquetaTestSuite;
```

C.6. Archivo wscript en etiqueta

```
1  # -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8;
       -*-
2
3  # def options(opt):
4  #     pass
5
6  # def configure(conf):
7  #     conf.check_nonfatal(header_name='stdint.h', define_name='HAVE_STDINT_H')
8
9  def build(bld):
10     module = bld.create_ns3_module('etiqueta', ['core'])
11     module.source = [
12         'model/etiqueta.cc',
13         'helper/etiqueta-helper.cc',
14         ]
15
16     module_test = bld.create_ns3_module_test_library('etiqueta')
17     module_test.source = [
18         'test/etiqueta-test-suite.cc',
```

```
19          ]
20
21      headers = bld(features='ns3header')
22      headers.module = 'etiqueta'
23      headers.source = [
24          'model/etiqueta.h',
25          'helper/etiqueta-helper.h',
26          ]
27
28      if bld.env.ENABLE_EXAMPLES:
29          bld.recurse('examples')
30
31      # bld.ns3_python_bindings()
```

Bibliografía

[CCL03] Imrich Chlamtac, Marco Conti, and Jennifer J.-N. Liu. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks*, 1(1):13 – 64, 2003.

[Chr16] Per Christensson. Techterms. `http://techterms.com/definition/`, 2005-2016. Accedido el 13-03-2016.

[CMT07] Edgar Chávez, Nathalie Mitton, and Héctor Tejeda. *Ad-Hoc, Mobile, and Wireless Networks: 6th International Conference, ADHOC-NOW 2007, Morelia, Mexico, September 24-26, 2007, Proceeedings*, chapter Routing in Wireless Networks with Position Trees, pages 32–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Esp01] Real Academia Española. *Diccionario de la lengua española*. 2001.

[FCoVa12] António Fonseca, André Camões, and Teresa Vazão. Geographical routing implementation in ns3. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 353–358, ICST, Brussels, Belgium, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Fou15a] National Science Foundation. ns-3. `https://www.nsnam.org/`, 2011-2015. Accedido el 11-03-2016.

[Fou15b] National Science Foundation. Older releases. `https://www.nsnam.org/releases/older/`, 2011-2015. Accedido el 13-03-2016.

[Fou16a] Wikimedia Foundation. Basic computer network components. `https://en.wikiversity.org/wiki/Basic_computer_network_components`, 2016. Accedido el 13-03-2016.

[Fou16b] Wikimedia Foundation. Emulador de terminal. https://es.wikipedia.org/wiki/Emulador_de_terminal, 2016. Accedido el 15-07-2016.

[Fou16c] Wikimedia Foundation. Network simulation. https://en.wikipedia.org/wiki/Network_simulation, 2016. Accedido el 15-07-2016.

[Fou16d] Wikimedia Foundation. Network topology. https://en.wikipedia.org/wiki/Network_topology, 2016. Accedido el 14-03-2016.

[Fou16e] Wikimedia Foundation. Teselado. https://es.wikipedia.org/wiki/Teselado, 2016. Accedido el 14-03-2016.

[GNU14] GNU. Gnu general public license, version 2. http://www.gnu.org/licenses/gpl-2.0.html, 2014. Accedido el 11-03-2016.

[KK00] Brad Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM.

[Nat15a] National Science Foundation. *ns-3 Manual*, February 2015. Accedido el 12-04-2016.

[Nat15b] National Science Foundation. *ns-3 Tutorial*, February 2015. Accedido el 13-03-2016.

[oO16] University of Oxford. Oxford english dictionarie. www.oxforddictionaries.com/, 2016. Accedido el 15-07-2016.

[PCL01] F.C. Padró, J.F. Canudas, and A.S. Lladó. *Matemática discreta*. Politext (Universidad Politécnica de Cataluña). Edicions UPC, 2001.

[Sol] Claudia Marina Vicario Solórzano. Estructura de datos. http://148.204.211.134/polilibros/Portal/Polilibros/P_terminados/EstrRepreDat/Files/arboles_balanceados.html. Accedido el 18-07-2016.

[TV05] Héctor Tejeda Villela. *El grafo virtual para ruteo geométrico en una red inalámbrica ad-hoc*. PhD thesis, Universidad Michoacana de San Nicolás de Hidalgo, 2005.

.

Índice alfabético