Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Redes Móviles, Inalámbricas y Distribuidas - ICARO

Análisis de Protocolos de Enrutamiento en Redes Malladas Inalámbricas con Restricciones en el Acceso a Internet

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
por el Bachiller:

Cordero Báez, Carlos Alberto
C.I.: 16.905.245
E-Mail: carlos.cordero.0@gmail.com

Para optar al título de: Licenciado en Computación

Tutora: Profesora María Elena Villapol

Caracas, Octubre 2014

Caracas, Octubre 2014

Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Laboratorio de Redes Móviles, Inalámbricas y Distribuidas (ICARO)

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por el Bachiller Carlos Alberto Cordero Báez C.I.: 16.905.245, con el título “Análisis de Protocolos de Enrutamiento en Redes Malladas Inalámbricas con Restricciones en el Acceso a Internet”, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 27 de Octubre de 2014, a las 11:00 am, para que su autor lo defendiera en forma pública, en la Sala I de la Escuela de Computación, mediante una exposición oral de su contenido, y luego respondiera satisfactoriamente a las preguntas que les fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente acta, en Caracas el 27 de Octubre de 2014, dejándose también constancia de que actuó como Coordinador del Jurado la Profesora Tutora María Elena Villapol.

_____

Prof. María Elena Villapol
(Tutora)

_____

Prof. Robinson Rivas
(Jurado Principal)

_____

Prof. Fernando Crema
(Jurado Principal)

“Vivir en el corazón de los que

dejamos atrás no es morir.”

Thomas Campbell

A ti hermano, siempre con nosotros Frank.

Agradecimientos

A mi padre y madre, Gabriel y Cecilia:

“Jamás en la vida encontrarás ternura mejor y más desinteresada que la de tu madre." - Honoré de Balzac.

"El sueño del héroe, es ser grande en todas partes y pequeño al lado de su padre." - Victor Hugo.

A ustedes, que me enseñaron a ser fuerte en la vida, que lo más importante es la familia y la paz. Ustedes, que me mostraron que el cariño no tiene límites cuando es verdadero. Por ser las dos principales razones por quien soy hoy en día. No hay palabras para agradecerles, ni vida para devolverles el favor. ¡Gracias!

A mis hermanas, Gabriela y Alexandra:

"Una familia feliz no es sino un paraíso anticipado." - Sir John Bowring.

Por estar en las buenas y las malas, por ayudarme a crecer y ahora compartir un montón de años juntos. Por ser mis segundas madres todo este tiempo. Son un ejemplo de familia, y un orgullo para mí tenerlas a mi lado. Tampoco tengo palabras para darles las gracias, ni tiempo para mostrarles todo lo que han significado en la vida. ¡Gracias!

A mis sobrinos, Christian, Jorge, Rafael y Santiago:

"Sólo dos legados duraderos podemos dejar a nuestros hijos: uno, raíces; otro, alas." - Hodding Carter

Por darme miles de razones de salir para adelante, el solo verlos crecer y saber que están allí, con un gran futuro por delante, me ha llevado a ser cada dia mejor persona y darles todo el buen ejemplo posible. Ustedes son los colores a los días de la familia. ¡Gracias!

A mis cuñados, Jorge y Christian:

Por apoyarme en todo lo que han podido, años después de conocerlos han sido mis nuevos hermanos. ¡Gracias!

A mi familia:

Quienes me han enseñado que el cariño sincero transpasa más allá de las paredes del hogar. ¡Gracias!

A mis amigos y amigas:

Ustedes, con los que he vivido innumerables experiencias, y que juntos hemos estado en este tren de la vida. ¡Gracias!

Y a todas esas personas que han pasado por mi vida, que han creido en mi, de las que he aprendido, y/o me han apoyado en un momento u otro.

¡Gracias a todos ustedes!

Resumen

Debido a la cantidad de protocolos de enrutamiento que actualmente existen para las plataformas móviles inalámbricas, cada uno con sus diferentes acercamientos al problema que es enrutar un paquete a través de una red, es necesario establecer comparaciones entre ellos para verificar su funcionamiento en un ambiente de trabajo definido. Actualmente con el desarrollo de las tecnologías de redes malladas inalámbricas, estas comparaciones permiten verificar el funcionamiento de estos protocolos con el fin establecer cómo se comportan estos, y si alguno sobresale sobre los demás.

Esta investigación surge de la idea de expandir la señal de los puntos de accesos con conexión a Internet que se encuentran en las urbanizaciones limitantes a barriadas que no cuentan con este tipo de acceso al Internet. Muchos de los hogares de estas urbanizaciones cuentan con Internet y un punto de acceso que se propaga mucho más allá del hogar. Uno de los elementos base para desarrollar este tipo de solución es el elegir el protocolo de enrutamiento a utilizar, ya que en el cae un buen peso del funcionamiento de una red mallada inalámbrica.

En este desarrollo se utilizó como banco de pruebas (o *testbed*) un simulador de redes, en este caso *Network Simulator NS-3*. Este simulador es basado en Linux, y usa como lenguaje principal C++. Siguiendo las tendencias tecnológicas, se decidió utilizar los protocolos Optimized Link State Routing Protocol (OLSR) y Hybrid Wireless Mesh Protocol (HWMP) junto a IEEE 802.11s.

Estos protocolos fueron evaluados sobre un escenario base al que se les variaron diferentes parámetros, como la cantidad de clientes o tipo de tráfico, para así verificar posibles diferencias entre estos.

Al finalizar, una vez estudiados los protocolos seleccionados, su comportamiento y ventajas propias, se determinó que la mejor opción para las condiciones presentadas fue HWMP en conjunto con 802.11s

Palabras Claves: red mallada, red inalámbrica, protocolo de enrutamiento, OLSR, HWMP, IEEE 802.11s, Network Simulator, NS3.

Índice General

Índice de Ilustraciones

Índice de Tablas

1. Introducción

Las redes inalámbricas son parte fundamental de la infraestructura tecnológica moderna. Han permitido llevar tanto a Internet como otros servicios móviles a una cantidad cada vez mayor de equipos y personas. Hoy en día, es común encontrar equipos electrónicos que cuentan con alguna conectividad inalámbrica hacia algún tipo servicio de telecomunicación.

El gran crecimiento las telecomunicaciones en los últimos años ha colocado presión a las tecnologías que las soportan para que mantengan y mejoren su rendimiento, y pueda continuar este crecimiento que día a día presentan. El aumento tanto de dispositivos como de tráfico en la red conlleva a necesitar mejores tecnologías junto a estas necesidades.

Las redes malladas inalámbricas son una alternativa a las típicas redes basadas en una infraestructura previamente establecida. Este tipo de redes permiten conectar equipos donde no se puede establecer una infraestructura física permanente. Un ejemplo es en la ciudad de Caracas, donde se tienen urbanizaciones con hogares que cuentan con acceso a Internet, limitando con barriadas donde no se cuenta con acceso fijo a la red. Una solución a esto podría ser ampliar el rango de cobertura de un punto de acceso inalámbrico mediante redes malladas, para así compartir el mismo con las zonas cercanas más allá del rango que puede cubrir el equipo por sí solo. Los propietarios de dicha conexión a Internet podrían ceder parte del ancho de banda para ayudar a proveer de Internet a estas zonas.

Con estos puntos como base, en este trabajo se estudia el rendimiento de una red mallada inalámbrica bajo diferentes protocolos de enrutamiento, cuando se cuenta con diferentes anchos de banda para conectar con Internet y un número de nodos cada vez mayor en la red.

1.1. Planteamiento del problema

Las redes inalámbricas Ad-Hoc pueden brindar una cantidad de beneficios para aplicaciones donde no se pueden contar con infraestructura. Pero para ello se deben de tomar en cuenta varios factores que en una red inalámbrica común no se tomarían. Este es el caso del enrutamiento dentro de la red mallada.

Los protocolos de enrutamientos tradicionales no suelen funcionar bien para este tipo de escenarios. Debido a que, al contrario de una red tradicional, una red

mallada suele ser inherentemente inestable, el enrutamiento debe adaptarse rápidamente a los cambios que se producen, como equipos que entran o salen de la red, enlaces que dejan de estar disponibles, etc. Por esto, los protocolos de enrutamiento para redes Ad-Hoc es un tema de estudio actualmente.

Sin embargo, muchos de estos estudios presentan escenarios donde la comunicación es únicamente dentro de la red mallada, pero pocos estudios han tomado en cuenta casos como tener un acceso limitado hacia fuera de la malla (por ejemplo, a Internet).

Este trabajo de investigación se plantea un escenario de este tipo, donde la red mallada presenta un único punto de salida con un ancho de banda limitado. En este caso, los protocolos de enrutamiento deben enfrentarse al “embudo” formado en los nodos cercanos al punto de acceso de salida. Por lo tanto, con todo lo planteado anteriormente, este trabajo pretende ayudar a encontrar la respuesta a la siguiente interrogante:

¿Cuál protocolo de enrutamiento puede ofrecer el mejor rendimiento en una red mallada inalámbrica, en la que el acceso a Internet sea limitado?

1.2. Objetivo general

Evaluar varios protocolos de enrutamiento para redes malladas con acceso hacia y desde Internet con limitado ancho de banda y diferentes cantidades de nodos.

1.3. Objetivos específicos

Los objetivos específicos de este Trabajo Especial de Grado son:

- Seleccionar los protocolos de enrutamiento a ser estudiados.
- Definir casos de estudio y escenarios para el estudio de los protocolos seleccionados.
- Diseñar un banco de prueba simulado para el estudio de los protocolos seleccionados.
- Realizar pruebas de comportamiento y rendimiento en el entorno simulado para los protocolos seleccionados.

- Capturar aspectos cuantitativos como el rendimiento (*throughput*), fluctuaciones (*jitter*), retardo, etc., para elaborar comparaciones entre los protocolos seleccionados.
- Analizar los resultados obtenidos.

1.4. Justificación e importancia

Una de las ventajas principales de las redes malladas es poder funcionar sin una infraestructura establecida, contando únicamente con los equipos móviles que pertenecen a la red. Las ventajas que presenta este tipo de soluciones son notables, al poder interconectar equipos en ubicaciones remotas, extender el área de cobertura más allá de una conexión punto a punto, ajustarse automáticamente a nodos que se unen o se desconectan a la red, etc.

Sin embargo, las redes malladas aún se encuentran en un constante estudio, y uno de los aspectos más estudiados es la habilidad de los equipos en la red de conocer quienes participan en esta, y cómo hacerles llegar la información a estos destinos. Adicionalmente, se debe tomar en cuenta que por lo general el acceso a Internet en la red mallada es limitado a unos pocos nodos, en los cuales se pueden formar embudos y afectar de manera notable el tráfico de datos. En estos aspectos existe una gran cantidad de soluciones propuestas, cada una con puntos positivos y negativos propios aún por verificar.

Es por esto que se estudian dos de los protocolos más importantes en las redes malladas, comparándolos entre sí para analizar su comportamiento, ventajas y desventajas, tomando en cuenta un acceso limitado a Internet desde la red; para definir cuál de dichos protocolos sería el más idóneo en la situación planteada.

1.5. Distribución del documento

El presente Trabajo Especial de Grado está constituido de los siguientes capítulos:

- *Capítulo 1 - Introducción:*

Introducción y explicación del problema a atacar. Objetivos a alcanzar junto con la justificación de los mismos. Distribución del documento.

- *Capítulo 2 - Marco Teórico:*

Marco teórico, donde se definen y explican términos claves para el desarrollo del presente trabajo, como son redes inalámbricas y sus estándares, redes malladas inalámbricas, protocolos de enrutamiento para estas, y el estándar para redes malladas inalámbricas 802.11s.

- *Capítulo 3 - Metodologías, Herramientas y Métricas:*

Metodologías de desarrollo en la investigación, fases para el desarrollo, herramientas de hardware y software, descripción del simulador utilizado y métricas obtenidas de la simulación.

- *Capítulo 4 - Implementación de los Escenarios y Ejecución de las Pruebas:*

Explicación y selección de los escenarios a evaluar. Desarrollo e implementación de los escenarios seleccionados dentro del simulador. Ejecución de pruebas previas a la ejecución final. Ejecución del simulador y extracción de datos

- *Capítulo 5 - Análisis de Resultados:*

Resultados finales en formato de gráficas, comparando los diferentes resultados, realizando tanto el análisis de estos como un análisis final general.

- *Capítulo 6 - Conclusiones:*

Expresa las conclusiones luego de analizada la información final de la investigación, junto a las limitaciones presentadas y recomendaciones para posibles investigaciones futuras.

2. Marco Teórico

2.1. Redes Inalámbricas

2.1.1. Definición

El concepto de redes inalámbricas es aplicado al tipo de redes en las que el medio de comunicación utilizado no depende de un medio físico guiado (como cables metálicos o fibra óptica), sino de un medio de transmisión no guiado, comúnmente ondas electromagnéticas (aunque existen tecnologías basadas en transmisión por luz) [1].

Este concepto provee de beneficios adicionales no asociados con las redes alámbricas. Entre ellas destaca, en especial, la movilidad. Al no depender de una instalación física, los equipos que utilizan este tipo de enlaces no necesitan estar fijos y pueden ser móviles dentro del rango de transmisión de la red inalámbrica.

Adicionalmente, al no necesitar medios físicos, se reduce la cantidad de cables y conexiones requeridas en otros casos. Así, como no es necesario el uso de cables, se facilita la instalación de redes sin modificar la infraestructura física instalada.

Sin embargo, por ser un medio de difusión no guiado, las redes inalámbricas tienen como una de sus principales desventajas problemas de seguridad. Cualquier dispositivo compatible con la tecnología de transmisión utilizada, puede capturar todo lo que se esté enviando por la red siempre que se encuentre a rango de la señal del emisor.

Otro inconveniente en este tipo de redes es lo propenso que son las señales a las interferencias. Debido a esto, los países cuentan con regulaciones para el uso del espectro electromagnético, definiendo espacios para ser utilizados por diferentes tipos de dispositivos u organizaciones.

Las redes inalámbricas han cumplido un papel protagónico en la evolución moderna de la tecnología. Desde la telefonía celular hasta las redes inalámbricas locales, las tecnologías inalámbricas han sido pilares en el desarrollo de las facilidades de telecomunicaciones modernas.

2.1.2. Tipos de redes inalámbricas

Al igual que las redes cableadas, las redes inalámbricas se pueden clasificar según la distancia o zona geográfica que se logra cubrir [2], como se puede observar en la ilustración 2-1.

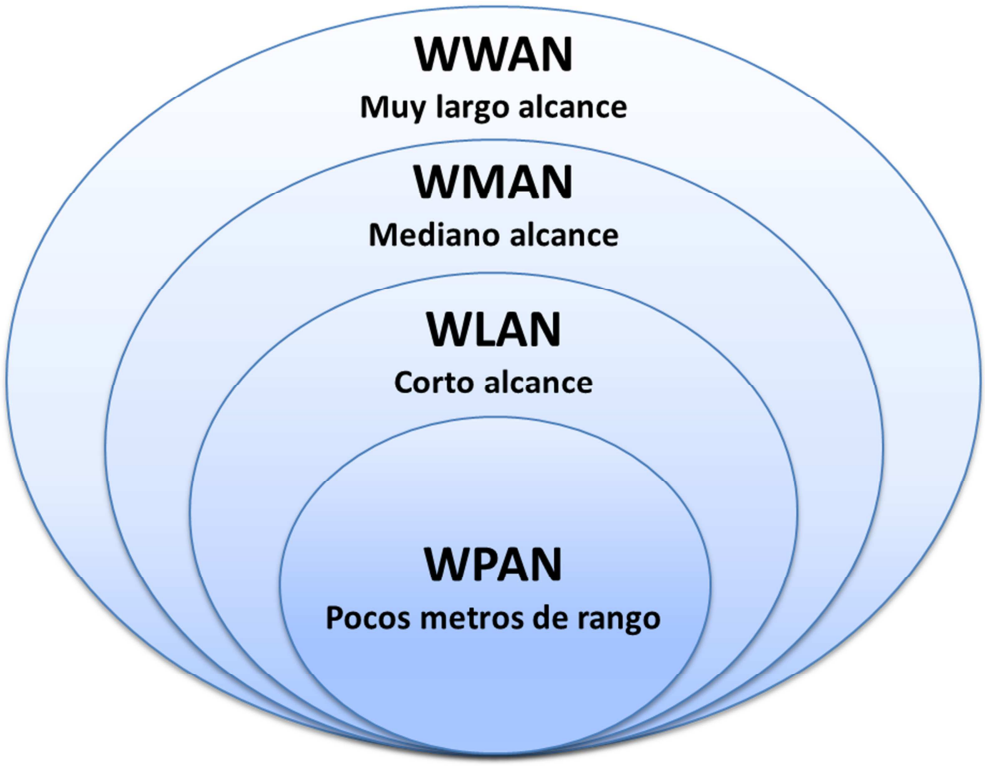

Ilustración 2-1 - Clasificación de las redes inalámbricas según distancia a cubrir.

- *Redes WWAN:*

Las Redes Inalámbricas de Área Amplia (*Wireless Wide Area Network,* WWAN) son utilizadas para muy largas distancias, que pueden ser desde regiones geográficas, países o continentes enteros. Por la distancia de transmisión, se requieren diferentes técnicas de transmisión a los otros tipos de redes. Sin embargo, su velocidad suele ser muy reducida en comparación a estas.

- *Redes WMAN:*

Las Redes Inalámbricas de Área Metropolitanas (*Wireless Metropolitan Area Network,* WMAN) se utilizan en distancias que van desde un campus hasta

una ciudad. Por lo general se utilizan para interconectar diferentes WLANs, o para ser usadas directamente como tecnología de última milla (*Last Mile*). En oportunidades, las WMAN se consideran igualmente WWAN por la vasta distancia que logran cubrir.

- *Redes WLAN:*

Las Redes Inalámbricas de Área Local (*Wireless Local Area Network,* WLAN) se caracterizan por ser redes de corto alcance, por lo general de varias decenas de metros hasta un edificio pequeño. Son las redes más conocidas, suelen ser privadas y veloces, y utilizadas para interconectar equipos finales.

Entre las redes WLAN, tenemos dos subtipos:

- Con infraestructura: Se denota con infraestructura a este tipo de WLAN ya que cuenta con un elemento encargado de controlar y administrar la interconexión entre los equipos. A este coordinador, conocido como estación base, se conectan todos los equipos inalámbricos que requieran acceso a su red. Generalmente, dicha estación base también sirve de puerta de enlace hacia Internet u otra red fija, realizando las funciones de puente y enrutamiento respectivas.

- Sin infraestructura o Ad-Hoc: Una red sin infraestructura es aquella donde no existe un elemento coordinador, y todos los nodos están en igualdad de condiciones. Los nodos se conectan entre si según su alcance y disponibilidad, y el enrutamiento entre estos nodos es un trabajo realizado en conjunto por ellos mismos. Esto implica que cada nodo debe servir también de enrutador en caso de ser necesario. Se diferencian de las redes con infraestructura en el hecho que pueden ser generadas espontáneamente, ya que los nodos pueden conectarse a la red mediante cualquier otro miembro de la misma. Es de notar que las redes Ad-Hoc suelen reducir su rendimiento a medida que crecen la cantidad de nodos en ella.

- *Redes WPAN:*

Las Redes Inalámbricas de Área Personal (*Wireless Personal Area Network,* WPAN) se caracterizan por un rango muy limitado, por lo general de pocos metros. Suelen ser aplicadas para conectividad entre equipos muy cercanos, como teléfonos móviles, equipos periféricos, accesorios electrónicos personales, etc.

2.2. Estándar IEEE 802.11

IEEE 802.11 es un estándar que define el funcionamiento tanto de la capa física (PHY) como la subcapa de control de acceso al medio (*Media Access Control*, MAC) dentro de la capa de enlace de datos en el modelo OSI para una red de tipo WLAN [3]. IEEE 802.11 recopila un conjunto de estándares y reglas utilizadas para la implementación de redes WLAN para la transmisión de datos. El estándar fue liberado en 1997, y a partir de esa fecha se han ido añadiendo múltiples agregados o enmiendas. Estas enmiendas llevan por notación “802.11” seguido de una o más letras. Los agregados se utilizan para definir nuevos detalles al estándar, correcciones de enmiendas anteriores, extensiones para ciertas funcionalidades, etc. Entre las enmiendas más conocidas se tienen las siguientes:

2.2.1. IEEE 802.11a

Liberado junto a 802.11b en 1999 [4], estas difieren principalmente en el tipo de modulación, frecuencia y velocidad. 802.11a trabaja en la frecuencia de 5GHz, utilizando una modulación del tipo OFDM (*Orthogonal Frequency Division Multiplexing*), y tiene una velocidad teórica máxima de 54 Mbits.

Tanto la frecuencia utilizada como la velocidad de esta especificación son ventajas en comparación de 802.11b, ya que ofrece no solo una tasa de transferencia mayor, sino en una frecuencia mucho más descongestionada en comparación de la 2,4 GHz (utilizada por muchos otros equipos electrónicos, al ser una frecuencia libre).

Sin embargo, el rango se reducía notablemente debido a la utilización de esta frecuencia, y su mayor facilidad de ser absorbida o desviada por objetos solidos presentes en su camino. Además, el uso de 5 GHz evita que los equipos desarrollados bajo este estándar fueran compatibles con otras especificaciones básicas de 802.11, como el 802.11 original (llamado 802.11-1997, siendo hoy obsoleto) u 802.11b/g.

2.2.2. IEEE 802.11b

Junto a 802.11a, este estándar fue liberado en 1999 [4]. 802.11b funciona en la frecuencia 2,4 GHz, siendo esta una frecuencia libre y utilizada también por otros equipos electrónicos. 802.11b (y posteriormente 802.11g) sufren ocasionalmente de interferencias externas por esta razón. 802.11b también se

diferencia de 802.11a por una velocidad teórica máxima de 11 Mbits, utilizando una modulación DSSS (*Direct Sequence Spread Spectrum*) derivada del estándar original 802.11-1997.

Este protocolo fue rápidamente aceptado y popularizado gracias al aumento de desempeño en comparación de 802.11-1997, además de la compatibilidad con el estándar anterior, lo que permitió una fácil transición hacia el nuevo estándar. Para cuando los primeros productos que utilizaban 802.11a llegaron al mercado, ya el mismo había asimilado en su mayoría a 802.11b como el estándar *de facto* en tecnología inalámbrica local [5].

2.2.3. IEEE 802.11g

Debido a las ventajas y desventajas propias de los estándares 802.11a/b (mayor velocidad sacrificando rango de transmisión en el caso del primero, y el caso opuesto en el segundo), se creó el agregado 802.11g, el cual combina la mejor distancia de transmisión de 802.11b al utilizar la frecuencia 2,4 GHz, con la tasa de transferencia de datos de 802.11a gracias a la modulación OFDM presente en esta [6]. Así se logra tener una tasa de transmisión de datos teórica de 54 Mbits/s en la popular frecuencia libre 2,4 GHz.

Al trabajar en la misma frecuencia que 802.11b, 802.11g también proveyó compatibilidad con este, al permitir operaciones utilizando modulación DSSS, pero con la desventaja de funcionar a velocidades inferiores. Muchos de los equipos se ofrecieron con capacidad de doble banda, utilizando ambas frecuencias para proveer de compatibilidad a los equipos que funcionan bajo cualquiera de los 3 estándares (802.11a/b/g).

Al igual que 802.11b, fue rápidamente adoptado incluso antes de ser ratificado por la IEEE, en gran medida por la falta de rendimiento de 802.11b (el cual ya era ampliamente aceptado), y a la compatibilidad con versiones anteriores instalados en la gran cantidad de equipos desplegados. Sin embargo, heredó algunas deficiencias de 802.11b, como la facilidad de sufrir de interferencia en la congestionada frecuencia 2,4 GHz. [5]

2.2.4. IEEE 802.11n

Este estándar, liberado en 2009, es una gran mejora al rendimiento en comparación de todos los agregados anteriores (802.11a/b/g). También incluyó un gran número de mejoras adicionales, entre las que destacan la incorporación de la

tecnología MIMO (*Multiple Input, Multiple Output*) y el uso tanto de la frecuencia 2,4 Ghz como 5 Ghz. 802.11n lo<gra con esto tener una tasa de transmisión teórica máxima de 600 Mbits (dadas ciertas circunstancias óptimas)[7].

La tecnología MIMO es uno de los principales cambios traídos por este estándar. Con esta, se agrega la capacidad de utilizar múltiples antenas para la transmisión y recepción de datos. Gracias a esto, y mediante el uso de ciertos algoritmos, se puede lograr transmitir varias cadenas de datos al mismo tiempo usando una misma frecuencia (aumento de tasa de datos), o transmitir la misma cadena de datos por diferentes antenas (aumento del rango, reducción de interferencia).

802.11n, al igual que 802.11g, fue asimilado rápidamente por el mercado. El aumento de rendimiento y rango, junto a la compatibilidad con equipos 802.11a/g, lo colocó como estándar de facto para redes de tipo WLAN.

2.2.5. IEEE 802.11ac

El objetivo principal del estándar es proveer velocidades de transmisión superiores a 1 Gbps utilizando tecnologías inalámbricas. Este estándar fue aprobado por parte de la IEEE a finales año 2013 [8], y ya existen fabricantes que han liberado equipos compatibles con 802.11ac.

802.11ac se trata de una evolución del estándar anterior 802.11n para proveer un rendimiento muy superior a este. Mientras 802.11n tiene una velocidad máxima teórica de 600 Mbps, 802.11ac eleva esta velocidad hasta los 1,3 Gbps [9], aunque se prevé que dicho rendimiento sea superior en un futuro debido a las posibilidades de expansión del estándar.

Un detalle importante con este nuevo estándar, es que únicamente opera en la banda de frecuencia de los 5 GHz, sin compatibilidad con los equipos que utilicen 2,4 GHz. Por esto 802.11ac trabajará en conjunto con 802.11n, sin reemplazarlo directamente, para cubrir clientes que utilicen cualquiera de las tecnologías.

2.2.6. Comparativa entre los principales agregados de IEEE 802.11

En la tabla 1, se presenta una breve comparación entre los agregados anteriormente descritos:

Protocolo	802.11a	802.11b	802.11g	802.11n	802.11ac
Fecha de lanzamiento	Septiembre, 1999	Septiembre, 1999	Junio, 2003	Octubre, 2009	Diciembre, 2013
Velocidad máxima teórica	54 Mbps	11 Mbps	54 Mbps	600 Mbps	1,3 Gbps
Rango aproximado (al aire libre)	120 mts.	140 mts.	140 mts.	250 mts.	N/D
Modulación	OFDM	DSSS	OFDM	OFDM	OFDM
Frecuencia	5 GHz	2,4 GHz	2,4 GHz	2,4 / 5 GHz	5 GHz
Acceso al medio	CSMA/CD	CSMA/CD	CSMA/CD	CSMA/CD	CSMA/CD
MIMO	No	No	No	Si	Si

Tabla 2-1 - Comparación entre los agregados de IEEE 802.11.

2.3. Redes Malladas Inalámbricas

Las Redes Malladas Inalámbricas (*Wireless Mesh Networks*, WMN) son un tipo de tecnología que se basa en el uso de una topología sin infraestructura [10], donde cada nodo se comunica directamente con otros nodos alcanzables, tomando una forma de malla como se puede apreciar en la ilustración 2-2. Al no tener un elemento controlador central, cada uno de los integrantes debe cumplir funciones tanto de host final como de enrutador de ser necesario. A diferencia de las redes convencionales, donde el enrutamiento es procesado por equipos específicos para esta labor, la comunicación entre los nodos se realiza por un esfuerzo general, por lo que se dice que todos los nodos se encuentran en igualdad de condiciones.

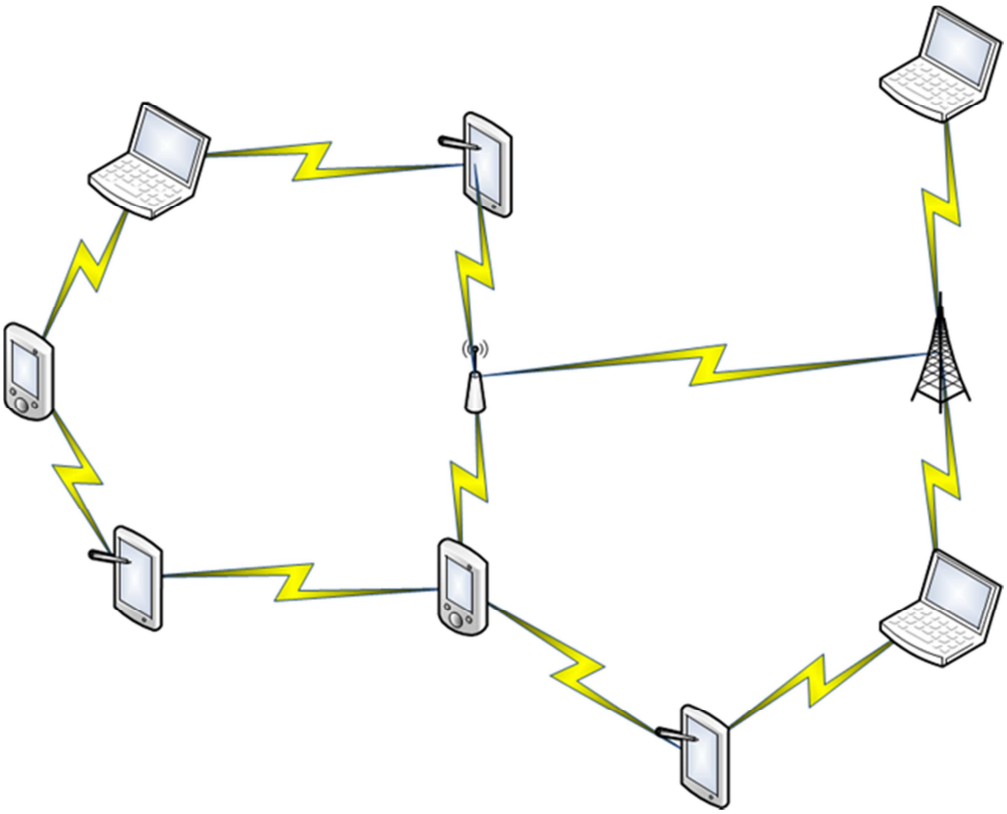

Ilustración 2-2 - Ejemplo de una red mallada.

2.3.1. Características

Algunas de las características más importantes para una red mallada son:

- Una red mallada puede ser implementada en cualquier sitio, siempre y cuando los nodos se puedan comunicar entre sí. No es necesario tener un enlace principal o *backbone*.
- Los nodos pueden conectarse a cualquier otro equipo perteneciente a la red mallada, sin importar su posición. Es suficiente con poder mantener conectividad con un nodo para comunicarse con los demás de la red.
- No existe centralización en la información de la red. Cada nodo tiene la capacidad de mantener suficiente información de la misma para poder encaminar los datos (enrutamiento proactivo), o bien tiene la capacidad de encontrar dicha información en el momento que lo necesite (enrutamiento reactivos).

- Cada nodo dentro de una red mallada puede ser origen, destino o intermediario. Por tanto, debe prestar funciones no solo de host origen/destino sino también de enrutador.
- Debido a la movilidad y la variación del estado de los enlaces, servicios que requieren priorización (*Quality of Service*, QoS) se ven fuertemente afectados por el funcionamiento de las redes Ad-Hoc.
- Los nodos son auto configurables, no necesitan un elemento externo para ajustarse a la red.
- Debido a que los nodos pueden dejar de estar disponible en cualquier momento, las redes malladas suelen tener algún tipo de recuperación de errores en los enlaces entre hosts (función llamada autocuración o *self-healing*).

2.4. Protocolos de enrutamiento para redes malladas inalámbricas

Cuando se tienen múltiples redes interconectadas, los dispositivos intermedios encargados de reenviar los paquetes deben conocer cómo y en qué dirección se encuentra el destino final. Esta función es llamada enrutamiento o encaminamiento de los paquetes. Para realizar esto, cada dispositivo intermedio debe tener el conocimiento necesario de la topología que le permita determinar cuál ruta deben seguir los paquetes a ser reenviados [11].

En casos donde se tiene un conjunto de redes sencillas y/o mayormente sin cambios a lo largo del tiempo, se pueden utilizar enrutamiento estático. Sin embargo, en casos más complejos donde, por ejemplo, se tiene una gran cantidad de redes o existen muchos cambios en la topología lo ideal es tener un sistema que automatice esta tarea. Para eso se tienen los protocolos de enrutamiento, los cuales definen procesos para poder completar las rutas a seguir utilizando como fuente tanto la información de la topología como un algoritmo para la toma de decisiones de enrutamiento.

Una de las formas básicas para clasificar los protocolos de enrutamiento es dependiendo la información de la ruta es solicitada al momento de necesitarla, o si mantiene la información permanentemente.

A continuación se explican los tipos de protocolos proactivos, reactivos e híbridos:

2.4.1. Protocolos de enrutamiento proactivos

Este tipo de protocolos está basado en el conocimiento previo de la red, para que al momento de necesitar una ruta tenera de antemano. Se calcula previamente, mediante un algoritmo y la información de la red, cuál es la mejor ruta a tomar. Por lo general, este tipo de protocolos utilizan algoritmos de tipo Dijsktra o Bellman-Ford para conseguir dicha ruta.

Los protocolos con este comportamiento deben actualizar sus tablas bien sea periódicamente o cada vez que hay un cambio en la red. Esto significa que requieren mayor uso de capacidad de procesamiento por parte del host, y un mayor consumo del ancho de banda por las actualizaciones emitidas. Estos algoritmos reducen considerablemente el tiempo para encontrar una ruta, en comparación de los protocolos reactivos.

Entre este tipo protocolos se tienen a *Highly Dynamic Destination-Sequenced Distance Vector* (DSDV) y *Optimized Link State Routing Protocol* (OLSR).

2.4.2. Protocolos de enrutamiento reactivos

Los protocolos reactivos crean entradas en las tablas de rutas solo cuando estas son requeridas. Al contrario de los protocolos proactivos, no mantienen una visión completa de la red, y la información requerida es destino, próximo salto y distancia de los destinos necesitados.

Ya que solo almacenan rutas necesarias o recientes, los hosts no requieren tanto poder de cómputo para procesar los requerimientos de enrutamiento. Sin embargo, al tener que buscar las rutas recurrentemente, suele existir un retraso al principio de cada transmisión mientras se resuelve el enrutamiento.

Entre este tipo protocolos se tienen a *Dynamic Source Routing* (DSR) y *Ad-Hoc On-demand Distance Vector* (AODV) como ejemplos.

2.4.3. Protocolos de enrutamiento híbridos

Los protocolos de enrutamiento híbridos combinan tanto el comportamiento de los protocolos reactivos con los protocolos de tipo proactivo. Así se logra unir las fortalezas de ambos tipos de enrutamiento.

Típicamente, este tipo de protocolos se basan en protocolos de vector-distancia, y agregan múltiples de las funciones que suelen utilizarse en los protocolos de estado de enlace.

Como ejemplo, se tienen a *Hybrid Wireless Mesh Protocol* (HWMP) y *Zone Routing Protocol* (ZRP).

2.4.4. Ejemplos de protocolos de enrutamiento

A continuación se detallan los protocolos de enrutamiento Optimized Link State Routing Protocol (OLSR), Ad-Hoc On-demand Distance Vector (AODV) y Hybrid Wireless Mesh Protocol (HWMP):

- *Optimized Link State Routing Protocol (OLSR):*

Siendo un protocolo de enrutamiento proactivo [12], OLSR se basa en el conocimiento previo de la topología de red para definir el direccionamiento de los paquetes. Sin embargo, su comportamiento está diseñado para reducir en gran medida el sobrepeso del tráfico de control que comúnmente tienen los protocolos proactivos.

Para esto, OLSR no utiliza un broadcast común donde todos los nodos retransmiten lo recibido, sino que realiza una difusión (o *broadcasting*) selectiva, en donde solo determinados vecinos repiten la información enviada por un host. Este tipo de comportamiento tiene como nombre “inundación usando retransmisores multipunto” (o “*flooding using multipoint relays*”). Se puede observar la diferencia de cuantas veces la información es recibida en la ilustración 2-3, donde se comparan ambos tipos de inundación.

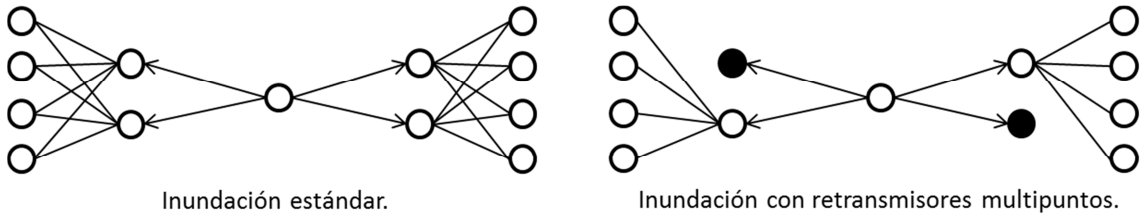

Ilustración 2-3 - Difusión (*broadcast*) estándar vs. Retransmisores multipuntos.

OLSR define tres funciones principales: detección de vecinos, un mecanismo para transmisión de datos de control, y otro mecanismo para definir la información de la topología de red a transmitir.

En OLSR, dos nodos adyacentes son vecinos solo sí existe un enlace bidireccional entre ellos. Cuando existe esta comunicación en ambos sentidos, se dice que hay un enlace simétrico, mientras que si este es unidireccional se le llama enlace asimétrico.

Adicionalmente, también existen los vecinos a dos saltos (*Two-hop Neighbors*). Estos existen, como se observa en la ilustración 2-4, cuando un nodo A tiene comunicación con otro nodo C, utilizando como intermediario un tercer nodo B. Dos requisitos para que estos sean llamados de esta manera son: todos los enlaces (tanto entre A y B, como B y C) deben ser simétricos, y los nodos A y C no pueden ser vecinos directos. Este ejemplo se puede observar gráficamente en la Ilustración 5.

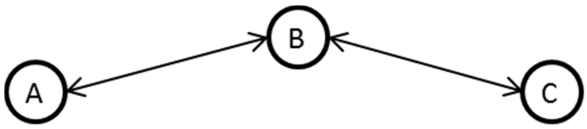

Ilustración 2-4 - Ejemplo de vecinos a dos saltos de distancia.

Para la detección de vecinos, OLSR emite paquetes HELLO periódicamente. Estos paquetes (que incluye tanto la dirección del nodo emisor como los nodos conocidos por este, y la simetría de estos) se utilizan no solo para la detección de los vecinos directos, sino para conocer los vecinos a dos saltos próximos al nodo.

Además, dichos mensajes HELLO son usados para detectar la calidad del enlace, en términos de simétrico o asimétrico, entre el nodo emisor y sus vecinos inmediatos. Si un nodo recibe un mensaje HELLO en el que está incluida su propia dirección (indicativo que el vecino recibe sus mensajes), el mismo considera el estado del enlace con este como simétrico.

En función de reducir la cantidad de mensajes repetidos en la red, OLSR selecciona vecinos específicos, los cuales serán encargados de retransmitir los mensajes de control emitidos por el primero. A estos vecinos seleccionados se les llama retransmisores multipunto (MPR o *MultiPoint Relay*, por sus siglas en inglés). Cada nodo es encargado de elegir sus propios MPR, utilizando para esto la información de los vecinos próximos y a dos saltos. Al elegir los MPR, es importante que estos cubran en totalidad todos los vecinos a dos saltos de distancia del nodo emisor, como se puede observar en la ilustración 2-5. El emisor notifica a los nodos seleccionados en los paquetes HELLO, usando *piggyback* (anexando la información al final del paquete) a dichos mensajes.

Al tener tanto una técnica para conocer la topología parcial de la red en cada nodo, como un método para distribuir eficientemente mensajes por la red, solo se necesita conocer cómo y qué información enviar por esta, para así lograr que todos los nodos tengan información suficiente de la topología de red y logren realizar un enrutamiento efectivo. En OLSR, los nodos emiten un mensaje de control de topología (*TC-Message*), el cual se difunde a todos los nodos de la red utilizando el método de los retransmisores multipuntos.

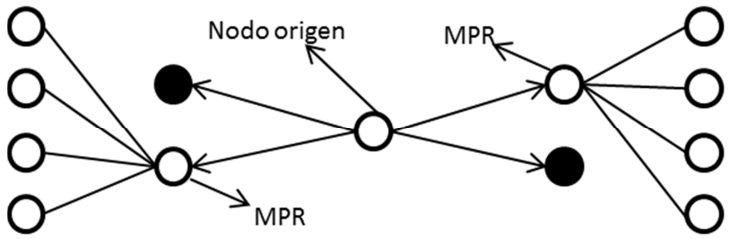

Ilustración 2-5 - Selección de los *Multipoint Relays* por un nodo.

Los TC-Message contienen tanto la dirección del nodo emisor, como todos los MPR que tiene este elegido. Tomando en cuenta que todos los nodos emitirán tanto su dirección como una lista de MPR (con lo cual está anunciando parte de sus vecinos simétricos directos), y juntando estas topologías parciales ya conocidas, se puede tener una visión general de la topología de la red. Luego, mediante algoritmos para calcular el camino más corto, se puede enrutar la información a través de la red.

OSLR representa un avance en eficiencia en comparación a otros protocolos de enrutamiento más básicos, al ofrecer una reducción del tráfico de control necesario y disminuir el peso de los broadcast en la red. Sin embargo, no toma en cuenta el rendimiento de los enlaces, por lo que utilizar únicamente el número de saltos no lleva necesariamente a la ruta óptima.

- *Ad-Hoc On-demand Distance Vector (AODV):*

AODV es un protocolo de enrutamiento reactivo salto a salto [13], almacenando temporalmente el próximo salto de la ruta a seguir en cada nodo.

Cuando un nodo necesita enviar información a otro, se inicia un descubrimiento de ruta (Path Discovery). El nodo origen envía a sus vecinos un requerimiento de ruta (Route REQuest, RREQ). Los vecinos que reciben este paquete (y no son el destino) guardan una entrada de ruta inversa hacia donde

recibieron el RREQ. Estos reenvían a sus vecinos cercanos el RREQ hasta que se alcanza el destino. Este proceso se puede observar en la ilustración 2-6.

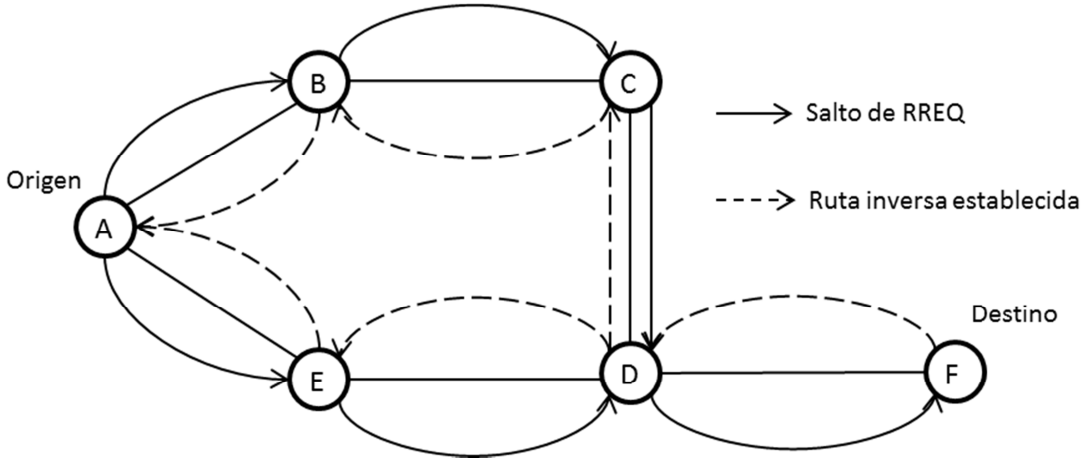

Ilustración 2-6 - Transmisión del mensaje RREQ en AODV.

Cuando el destino recibe el RREQ, este responde vía *unicast* al origen un RREP (*Route REPly*) estableciendo la ruta como se observa en la ilustración 2-7. Esta ruta se mantiene activa mientras es utilizada, pero si una entrada no se usa por cierto tiempo es eliminada.

Los RREQ también pueden ser respondidos por nodos intermedios, siempre y cuando estos ya tengan alguna ruta activa hacia el destino. En tal caso, este nodo ya informado es el encargado de enviar un RREP al origen, optimizando así el ciclo de descubrimiento de ruta.

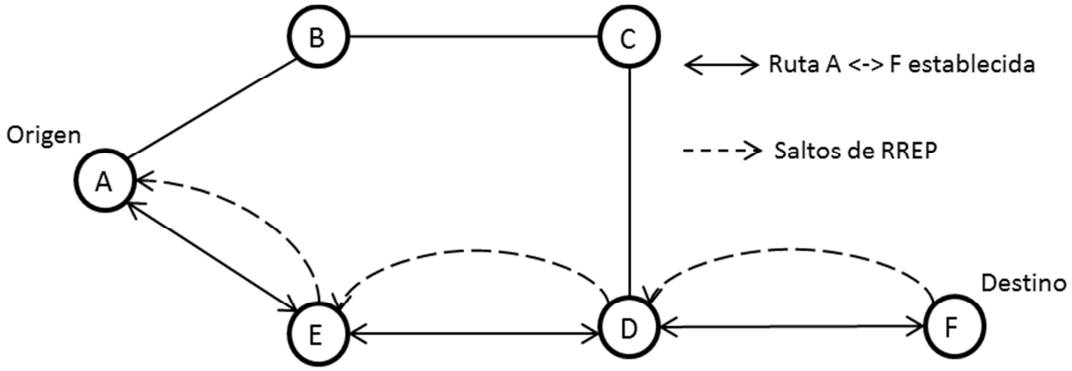

Ilustración 2-7 - Mensaje de respuesta RREP del destino en AODV.

AODV también cuenta con un mecanismo de actualización e invalidación de rutas en caso de falla de un nodo o enlace. Este mecanismo es ejecutado por mensajes llamados error de ruta. Cuando un nodo invalida una ruta, este crea un paquete de error en ruta (*Route ERRor*, RERR) anunciando el cambio, y lo envía

hacia los nodos cercanos. Al recibir un RERR, cada nodo verifica si tiene alguna ruta con el destino invalidado, y retransmite el mensaje a otros nodos.

Sin embargo, este protocolo presenta una desventaja al tener una gran latencia debido al tiempo y sobrecarga en la red al momento de necesitar una ruta. Adicionalmente, la sobrecarga de mensajes de control en dichos momentos requiere una mayor cantidad de ancho de banda.

- *Hybrid Wireless Mesh Protocol (HWMP):*

Este protocolo híbrido está basado en AODV, modificado para tomar también aspectos de un protocolo de estado de enlace [14]. Para esto, incluye dos modos para construir las rutas, tanto de una manera reactiva como proactiva, dándole así la capacidad de funcionar de una u otra forma.

Los dos modos para la detección de rutas en HWMP son:

- Modo Bajo Demanda (ilustración 2-8): en esta modalidad, el protocolo trabaja de manera reactiva, y cada nodo es capaz de descubrir una ruta por sí mismo. Permite a las estaciones comunicarse entre sí utilizando rutas punto a punto. No depende de ningún punto central de control para su funcionamiento.

- Modo Árbol (ilustración 2-9): para esta función, se requiere tener una estación central trabajando como “raíz de la malla” (*root mesh*). A diferencia del modo bajo demanda, en este modo la raíz establece rutas con cada una de las estaciones dentro de la malla, centralizando así la información. Al necesitar una ruta que no sea directamente alcanzable, las estaciones utilizarán a la raíz como “enrutador” hacia el destino final. Al establecer por adelantado las rutas a cada estación dentro de la red mallada, este toma la funcionalidad de un protocolo proactivo.

HWMP utiliza 3 tipos de mensajes básicos para establecer rutas, llamados *Path REQuest* (PREQ), *Path REPlies* (PREP) y *Path ERRor* (PERR). El uso de los dos primeros es muy similar a los mensajes *RouteRequest* y *RouteReply* en AODV. Adicionalmente a estos tres elementos, también se establece un cuarto mensaje llamado Anuncio de Raíz *(Root ANNouncement*, RANN), utilizado en el modo árbol para anunciar cuál estación está cumpliendo las funciones de raíz de la malla.

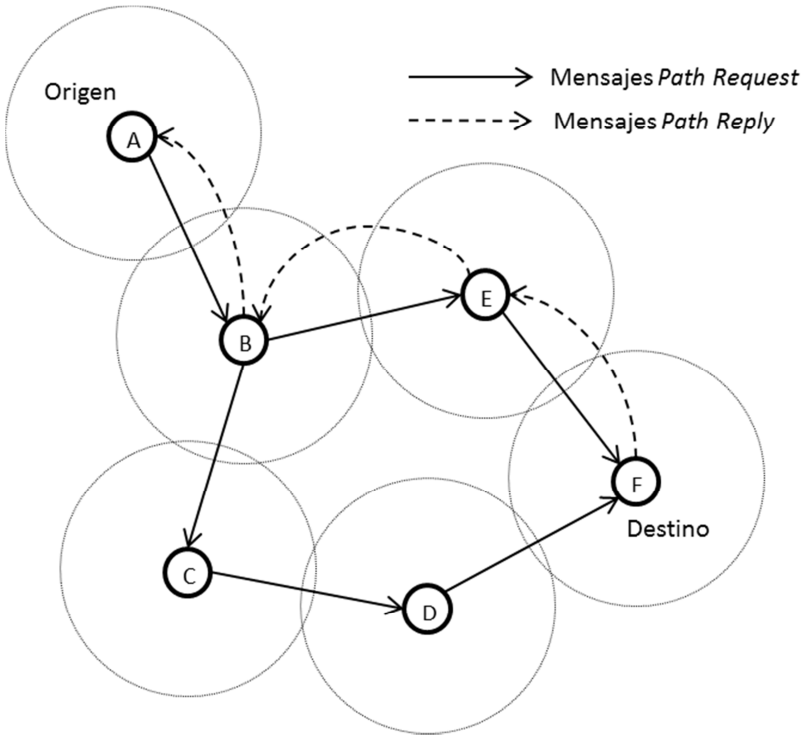

Ilustración 2-8 - Modo bajo demanda en HWMP.

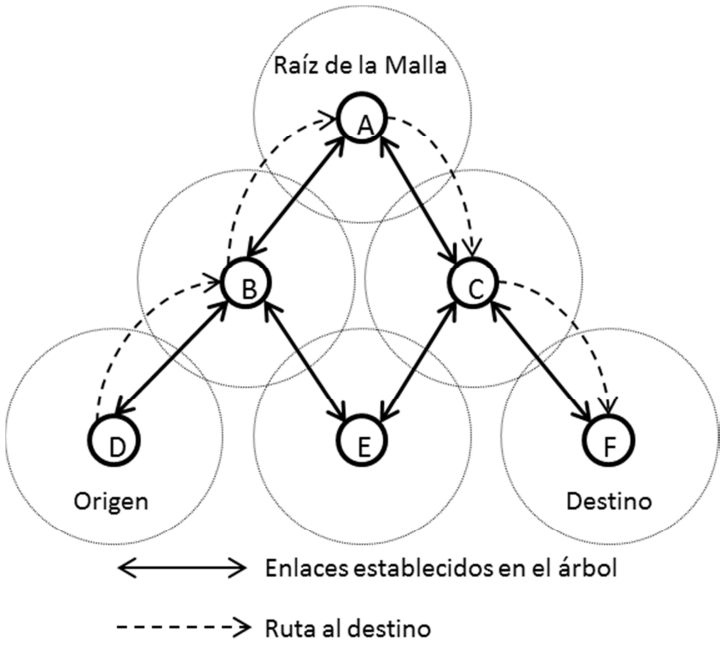

Ilustración 2-9 - Modo árbol en HWMP.

Al requerir una ruta hacia un destino, un nodo envía un PREQ a todos los nodos a su alcance. Al recibir la solicitud, dichos nodos pueden responderla utilizando un mensaje PREP si ellos son el destino o bien cuenta con una ruta

hacia el mismo. En caso contrario retransmiten la solicitud hacia otros nodos a su alcance hasta que todos los nodos procesen la solicitud. Sin embargo, es posible que un nodo pueda no tener información hacia el destino buscado, por lo que responde con un mensaje PERR.

En cada salto y antes de realizar el reenvío del PREQ, el nodo revisa el número de secuencia que contiene el mensaje. Si el número de secuencia es nuevo, o si es repetido pero tiene una mejor métrica que los recibidos anteriormente, el nodo reduce el Tiempo De Vida (TTL*, Time To Live*), modifica el contador de saltos (*Hop Count*) sumándole 1, y agrega a la métrica su propia medición, y procede a reenviarlo a sus vecinos más cercanos.

Debido a que en ciertos momentos es preferible una ruta con mayor número de saltos pero con mejor tasa de transmisión que uno más cercano y pero lento, HWMP utiliza una métrica que toma en cuenta no solo el número de saltos hacia un destino, sino la velocidad del enlace y la tasa de errores del mismo. Esta métrica es llamada *Airtime Link Metric*. En esta, se calcula la velocidad de transmisión de una muestra de datos, velocidad a la cual se le adiciona la tasa de errores. La métrica está dada por la siguiente formula: [15]

$$M = [Oma + Op + (Bt / R)] / (1 - Ept)$$

Donde M es la métrica total del enlace; Oma es la sobrecarga por acceso al medio; Op es la sobrecarga del protocolo; Bt es la cantidad de bits en el mensaje de prueba; R es la tasa de envío en Mbps; y Ept es la tasa de error en la prueba. Ept toma como un valor menor a 1, y mayor o igual a 0. Entre menor sea el valor resultante, mejor es la calidad del enlace. En cada salto de un mensaje, este valor se calcula para el último salto y se agrega al que contiene el paquete recibido.

HWMP fue pensado especialmente para redes malladas basadas en tecnología 802.11, con la inestabilidad presente en está. En caso de que una ruta falle, el nodo que detecta dicha falla envía un PERR hacia el origen para que este proceda buscar una nueva ruta o utilizar otra previamente almacenada.

Un punto interesante en el modo bajo demanda, es que también puede comportarse como un protocolo proactivo. Los nodos pueden descubrir la mejor ruta hacia todos los otros nodos enviando un PREQ con dirección destino MAC *broadcast*. Todo nodo que reciba dicho PREQ responderá con un PREP, informando así al nodo emisor de todas las rutas posibles a cada nodo. Este tipo de búsqueda de rutas suele causar bastante tráfico, por lo que es preferible utilizar el modo árbol para un modo proactivo.

HWMP junta elementos necesarios en las redes inalámbricas como detección de calidad de los enlaces, diferentes modos para diferentes situaciones, y recuperación de errores, dando así un protocolo bastante apto para las situaciones móviles de las redes malladas.

2.5. IEEE 802.11s

Con la rápida adopción de la tecnología 802.11, surgieron diferentes necesidades adicionales que cubrir con esta. Una de estas necesidades fue la de proveer acceso inalámbrico en locaciones donde colocar un punto de acceso inalámbrico resultaba dificultoso. Al tener un cable Ethernet una distancia máxima de 100 metros en condiciones óptimas, o por restricciones de la ubicación, conectar un punto de acceso en sitios difíciles o abiertos tiende a ser un trabajo complejo; por ejemplo tender un cableado para proveer de conectividad a varios puntos de acceso que cubre un estacionamiento, o en el techo de un gran almacén donde el acceso es limitado, suele ser una ardua tarea.

Pero una de las ventajas de las tecnologías inalámbricas es exactamente prescindir de cableado físico para la interconectividad de equipos. Múltiples fabricantes proveen mecanismos para interconectar varios puntos de accesos utilizando conectividad inalámbrica. Pero este tipo de procesos no se encuentran estandarizados, por lo que la implementación de estos sistemas resulta también difícil, y la compatibilidad entre fabricantes es muy reducida.

Por estas razones, la IEEE designó el grupo de trabajo 802.11s [16], encargado de elaborar un estándar para redes malladas inalámbricas, facilitando así el despliegue de puntos accesos y equipos sin el uso de cables, conectadas de forma Ad-Hoc.

En 802.11s, la nube mallada donde participan los equipos se le llama Conjunto de Servicios Básicos Mallados, o *Mesh Basic Service Set* (MBSS). En la ilustración 2-10 se puede observar la diferencia entre MBSS y una BSS estándar. Cada equipo con capacidad de comunicarse utilizando 802.11 es llamado Estación (*Station* o STA), y cuando una de estas estaciones implementa 802.11s es llamada estación mallada, o *mesh station* (*Mesh STA*).

Adicionalmente, una estación mallada puede contar con uno o más roles dentro de la nube MBSS:

- *Mesh Access Point*:

Es una estación mallada que provee acceso a otros equipos STA que no participan en el MBSS. Se le llama también punto de acceso mallado, o MAP.

- *Mesh Gate*:

Una estación mallada que se encuentra en el borde del MBSS, dando acceso desde y hacia a este, se le llama puerta mallada o MG.

- *Mesh Portal*:

Cuando un equipo cuenta con acceso a otro tipo de red diferente a 802.11 y provee por dicho medio acceso a Internet, se le llama portal mallado o MP.

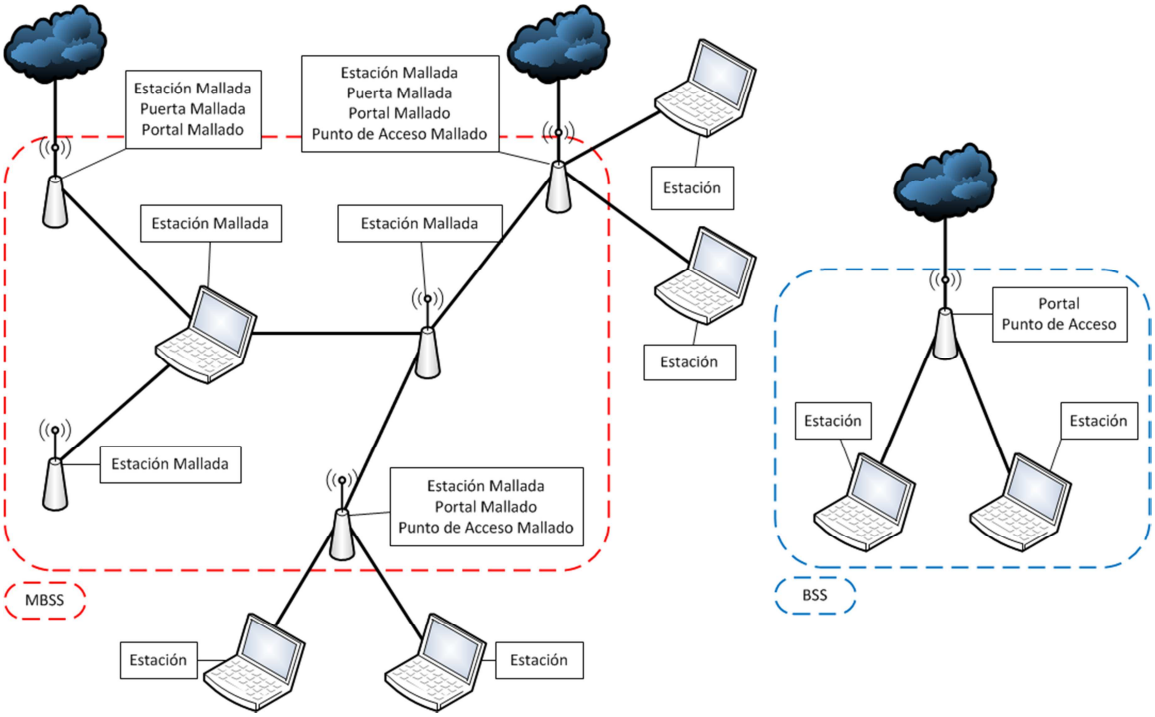

Ilustración 2-10 - Distintos roles disponibles en 802.11s vs. BSS estándar.

Es de notar que estos roles pueden combinarse en una misma estación mallada, pudiendo tener un equipo que cumple con funciones de punto de acceso y portal a la vez, o incluso todas los roles juntos.

El estándar 802.11s define igualmente otros aspectos de una red mallada inalámbrica, como cual protocolo de enrutamiento usar, la métrica de dicho protocolo, seguridad entre los nodos, ahorro de energía, etc.

En cuanto a los dos primeros temas (protocolo de enrutamiento y métrica), 802.11s señala el uso por defecto de HWMP con su propia métrica (*Airtime Link Metric*), pero soportando otros protocolos y métricas que los fabricantes quieran agregar. Esto da una base firme para iniciar el funcionamiento el estándar, mientras se permite la expansión o utilización de soluciones alternativas para casos específicos o mejoras futuras.

802.11s brinda ciertos beneficios en comparación de las instalaciones básicas [17], como por ejemplo:

- Se amplía el rango de cobertura, al poder utilizar otros equipos intermedios para poder transmitir hacia un destino.
- Se facilita el diseño e instalación de la red, ya que no se requiere que cada punto de acceso cuente con conexión cableada, reduciendo las consideraciones a tener en cuenta a la hora de diseñar la ubicación de los mismos.
- Los nodos pueden tener múltiples caminos para llegar a un destino. Así, en caso de falla en un enlace, se puede tomar una ruta distinta para entregar la información. Esta capacidad de recuperarse de errores se le llama *self-healing*.
- Los nuevos nodos que se unan se configuran automáticamente, por lo que resulta más fácil la expansión de la red.

Sin embargo, 802.11s también incluye desventajas que deben ser tomadas en cuenta:

- El aumento de equipos en la red puede aumentar tanto el tráfico de la red como el número de saltos, lo que con lleva a una reducción del rendimiento en el MBSS. Por esto, las redes 802.11s suelen ser limitadas en cuanto a lo escalables que pueden lograr ser.
- Las redes 802.11s deben tratar de diseñarse por completo desde el principio, precisamente por el punto anterior.
- La seguridad es un punto crítico en las redes inalámbricas, y por tanto también es un tema del que las redes 802.11s pueden sufrir. Sin embargo la IEEE tiene en 802.11s mecanismos de autenticación y seguridad para evitar este tipo de problemas.

802.11s representa un gran avance para las redes Ad-Hoc. Al contar ahora con una estandarización de cómo debe funcionar una red mallada, la interoperabilidad de los equipos se deberá ver positivamente afectada para futuras implementaciones.

3. Metodologías, Herramientas y Métricas

Con el fin de estudiar los protocolos de red seleccionados, se necesita crear un ambiente de prueba donde se puedan ejecutar estos protocolos de manera controlada. Para esto, se ha elegido utilizar una simulación como banco de pruebas (o *testbed*), con el que se podrán configurar los parámetros y situaciones necesarias en la búsqueda de resultados para ser estudiados, sin requerir de tener todos los equipos físicamente disponibles.

3.1. Fases para el desarrollo

Para llevar a cabo el desarrollo basado en simulaciones, es necesario seguir una serie de pasos que lleven el estudio hasta la obtención de resultados. Los siguientes puntos nombran cada uno de dichos pasos:

- *Escogencia y descripción de los escenarios:*

Se dice que en un escenario se colocan los elementos necesarios para representar un ambiente donde se llevará a cabo algún acto. En el caso de las simulaciones, luego de definido los parámetros esenciales que se mantendrán fijos durante las ejecuciones, se nombran los parámetros a variar para obtener distintos casos. Cada una de estas variaciones se toma como un escenario de prueba a ser estudiado.

- *Selección de métricas de importancia a utilizar:*

Luego de elegidos los escenarios, el siguiente paso es decidir que métricas son represantivas para el análisis del presente proyecto. Entre muchas métricas disponibles, se deben seleccionar aquellas que representen los cambios más importantes, de donde se puedan identificar ventajas o desventajas de cada uno de los protocolos a estudiar.

- *Implementación de los escenarios:*

La implementación de los escenarios se realizará en un simulador de redes, donde se configurarán los escenarios utilizando la sintaxis que provea este. Además, en esta fase se realizaran las pruebas necesarias previas a las ejecuciones finales para verificar el buen funcionamiento de los escenarios.

- *Ejecución de pruebas y generación de resultados:*

Una vez diseñados los escenarios y configurados, se procede a ejecutar las simulaciones con los parámetros finales. Estas simulaciones generarán resultados que serán verificados y ordenados, para así tener información suficiente que permita ver el comportamiento de los protocolos en los ambientes seleccionados.

- *Análisis de resultados:*

Una vez generados todos los resultados, se deben interpretar el significado de estos. Se compararan los resultados entre ellos, determinando luego del análisis la información crítica para llegar a la conclusión del presente trabajo.

3.2. Herramientas a utilizar en el desarrollo

En el desarrollo del presente Trabajo Especial de Grado se utilizaron las siguientes herramientas de trabajo:

3.2.1. Hardware y sistemas operativos

La parte práctica de este trabajo se ejecutará en un entorno de equipos virtuales. Se utilizarán tres máquinas virtuales independientes en la ejecución de las pruebas en paralelo, para así reducir el tiempo total de ejecución en las simulaciones. Adicionalmente se usa un ambiente de trabajo no virtualizado en un equipo portátil para el desarrollo y pruebas de las simulaciones antes de su ejecución.

La virtualización será implementada con la herramienta Microsoft HyperV, instalada en un sistema operativo Microsoft Windows Server 2012 Standard. Esto es alojado en un equipo servidor HP ProLiant DL360 G7. Las instancias virtuales se basan en el sistema operativo Linux, distribución Ubuntu 13.10. El ambiente de desarrollo se basa igualmente en el sistema operativo Linux, distribución Ubuntu 13.10.

A continuación en la tabla 3-1, la información de los equipos físicos a utilizar:

Equipos Físicos			
Cantidad	Marca	Modelo	Características
1	HP	ProLiant 360 G7	Procesador: Intel Xeon E5640 2.66 GHz Quad-Core Memoria: 8 GB DDR3 Almacenamiento: 4x300 GB HDD en RAID5 Sistema Operativo Base: Microsoft Windows Server 2012 Standard
1	Lenovo	ThinkPad L420	Procesador: Intel Core i3-2310M 2.10 GHz Dual-Core Memoria: 2 GB DDR3 Almacenamiento: 1x500 GB HDD Sistema Operativo Base: Linux Ubuntu 13.10

Tabla 3-1 - Equipos físicos a utilizar.

En la tabla 3-2 se observan la configuración de las instancias virtuales:

Instancias Virtuales	
Cantidad	Características
3	Memoria: 2 GB asignados base + 1 GB agregado dinámicamente Almacenamiento: 100 GB asignados dinámicamente Sistema Operativo Base: Linux Ubuntu 13.10

Tabla 3-2 - Instancias virtuales de simulación.

En la ilustración 3-1 se puede observar el resultado de las tres instancias virtuales completamente instaladas y en ejecución:

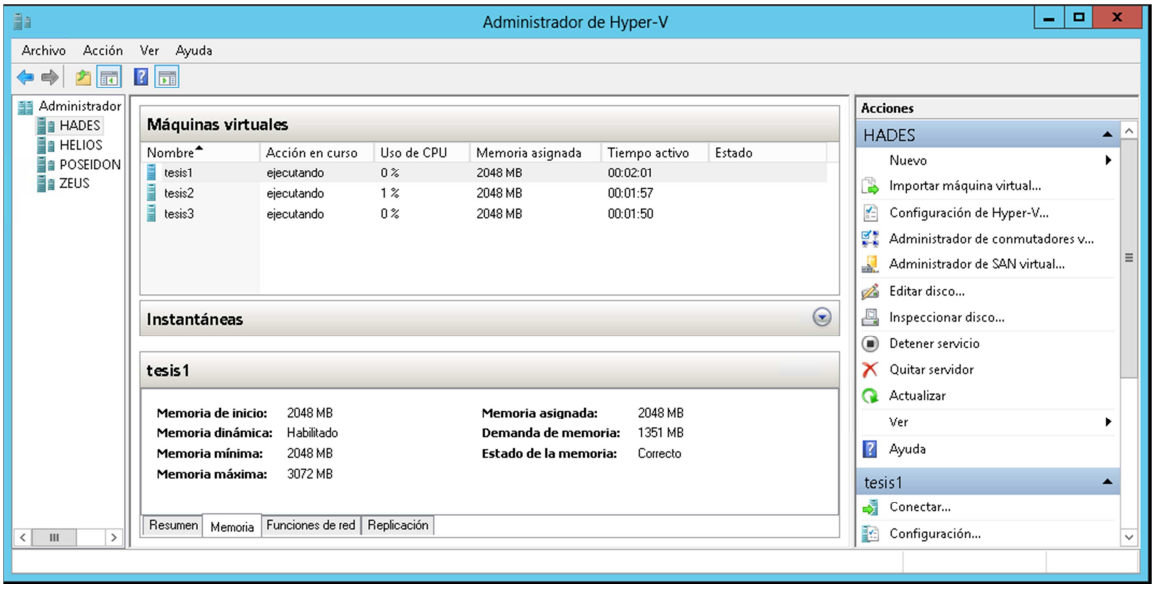

Ilustración 3-1 - Instancias virtuales en ejecución.

3.2.2. Simulador

La base para realizar la implementación es la utilización de un simulador de redes. En este caso se tomó *Network Simulator 3*, en su versión estable ns-3.19.

Network Simulator 3 (NS-3) [18] es un simulador de redes informáticas de código abierto basado en eventos discretos. El objetivo principal de esta herramienta es el desarrollo de un simulador completo y libre para promover investigaciones en el campo de redes y validación de las mismas, evitando usar sistemas reales que suelen ser costos y complejos. Hoy en día su versión más actual es Network Simulator 3, o NS-3.

- *Historia*

Network Simulator comenzó como una variante de un antiguo simulador de redes llamado REAL en el año 1989. Fue desarrollado por el Grupo de Investigación de Redes (*Network Research Group*, NSG) en el Laboratorio Nacional de Lawrence Berkely, por lo que se conoció inicialmente por LNBL Network Simulator (luego abreviado a NS-1). En 1995 el desarrollo de NS-1 era apoyado por diferentes entes, entre los que destacan DARPA (Agencia de Investigación de Proyectos Avanzados de Defensa, por sus siglas en inglés), Xerox, *University of California / Berkely* (UCB), y *University Southern California / Information Sciences Institute*. NS-1 fue basado en C++, con capacidad de describir escenarios de simulación en TCL (*Tool Command Language*).

Ya en 1995 se comenzaría el desarrollo de NS-2, el cual tendría compatibilidad con NS-1, e igualmente programado en C++ y OTCL (una versión orientada a objetos de TCL original). Fue desarrollado por Lawrence Berkely Laboratory con el apoyo de DARPA. El año siguiente, NS-2 fue extendido y distribuido por el proyecto VINT (*Virtual InterNetwork Testbed* – Banco de Pruebas de InterRedes Virtual).

La historia de NS-3 comenzó en 2004, cuando dos equipos separados comenzaron a explorar posibles reemplazos de NS-2. Un equipo financiado por la Fundación de Ciencia Nacional de Estados Unidos (*U.S. Science Foundation*, o USSF por su siglas en inglés) comenzaron a trabajar en NS-3, mientras otro grupo perteneciente a Instituto Nacional de Investigación en Informática y Automática (*Institut National de Recherche en Informatique et en Automatique*, INRIA por sus siglas en francés) trabajaba independientemente en el mismo objetivo, pero con enfoque principal en modelar el estándar IEEE 802.11. El resultado de este último

equipo fue conocido inicialmente como *Yet Another Network Simulator* (Otro Simulador Más de Redes, o YANS por sus siglas en inglés).

A inicios de 2005, ambos equipos unieron sus esfuerzos en un único proyecto. Desde el inicio se eliminó la compatibilidad con versiones anteriores, debido al trabajo extra que traería mantener esta funcionalidad. Así, el desarrollo de NS-3 (escrito en C++) comenzó en 2006, teniendo la primera versión pública a mediados de 2008.

Actualmente, NS-3 se encuentra en desarrollo constante tanto por el equipo de desarrollo original, como por la comunidad (al ser desarrollado bajo las premisas de Código Abierto). NS-3 ya implementa múltiples protocolos de enrutamiento inalámbrico, como AODV, DSDV, DSR, HMWP, OLRS, etc. Mientras tanto, hoy en día en NS-2 solo se realizan tareas de mantenimiento, y NS-1 no está siendo atendido de ninguna manera por encontrarse ya completamente obsoleto.

- *Arquitectura general de NS-3*

Network Simulator 3 está basado en modelos orientados a objetos, por lo que existen múltiples abstracciones y generalizaciones previas para la implementación de un escenario a simular [19]. Para comenzar, todo el manejo del flujo de datos está basado en la pila TCP/IP. Por tanto, cada dato generado pasa a través de las capas como si se tratara de una implementación real, y cada capa de la pila es a su vez independiente de las demás.

En un diseño general de redes se suele hacer distinciones a los equipos según su función, ubicación en la red, etc. Sin embargo, para NS-3 todos los dispositivos o funciones parten de una unidad básica llamada “Objeto” (*Object*).

Así, cualquier equipo en la red es derivado como un objeto llamado “Nodo” (*Node*), independientemente de su función o ubicación dentro del escenario. En la ilustración 3-2, se puede observar a la izquierda un diagrama de red donde se distingue cada equipo según su función, y a su derecha el mismo diagrama donde se abstrae de la función, y cada equipo es tomado simplemente como un equipo en la red, o un nodo.

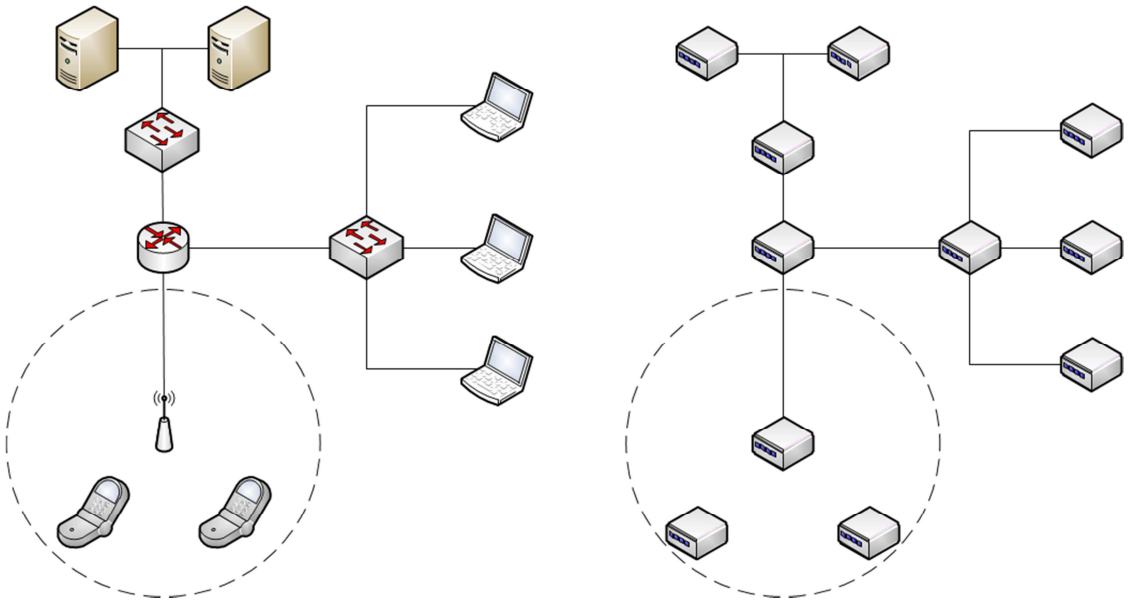

Ilustración 3-2 - Nodos definidos por contexto (izq.) vs. Genéricos (der.).

A estos nodos se les “instala” otros objetos especializados que permiten modelar el funcionamiento de estos como sea requerido. Cada objeto puede representar tarjetas interfaces de red, protocolos, aplicaciones, etc. Igualmente, a estos objetos adicionales se le podrían instalar otras funciones específicas a lo que representan.

Algunos de los objetos básicos son:

- Nodo (*Node*): Representa un equipo en la red. Puede ser un host final o cualquier equipo intermedio, ya que no distingue en funciones o ubicación en la red. Puede ser visto como un equipo o computador vacío, al que se le colocarán o instalaran otras funciones y tarjetas.

- Canal (*Channel*): Crea un medio de comunicación física por donde viajara la información. En NS-3 hay tres subclases básicas derivadas de canal: CsmaChannel (basado en los medios que utilizan Acceso Múltiple con Escucha de Portadora (CSMA o *Carrier Sense Multiple Access*), PointToPointChannel (para enlaces en el que medio es de tipo punto a punto), y WifiChannel (usado en los enlaces inalámbricos, donde el medio de transmisión es el aire).

- Dispositivo de red (*Net Device*): En un equipo, para conectarse a algún tipo de red se necesitaría agregar una tarjeta con interfaz de red (o *Network Interface Card*, NIC por sus siglas en inglés). Adicionalmente, se necesitarían agregar

controladores específicos para que esta tarjeta pueda ser utilizada. En NS-3 los objetos NetDevice incluyen ambos componentes. Estos dispositivos de red son instalados en los nodos para que puedan comunicarse con otros utilizando un canal de comunicación.

- Interfaz (*Interface*): Para utilizar direcciones IPv4/IPv6 se utilizan los objetos interfaces, los cuales se encargan de asignar una o varias direcciones IP a los dispositivos de red en los nodos, definir protocolos de enrutamiento, y otras tareas tanto en el uso de IP o a los dispositivos de red.

- Aplicación (*Application*): Representan las aplicaciones encargadas de generar y recibir información a través de la red.

En la siguiente ilustración (3-3) se ejemplifica como a los nodos base se le instalan otras funciones, en este caso el *protocolo IPv4*, una *aplicación* encargada de crear el tráfico de red y recibirlo para ser procesado, y un *dispositivo de red* inalámbrico. Adicionalmente se tiene un objeto *canal*, el cual controla como se mueven las ondas de radio a través del aire. Este canal se asocia al *dispositivo de red*, al igual que la *interfaz IPv4*, en la que se asigna la dirección IPv4 para el dispositivo de red al que está instalado.

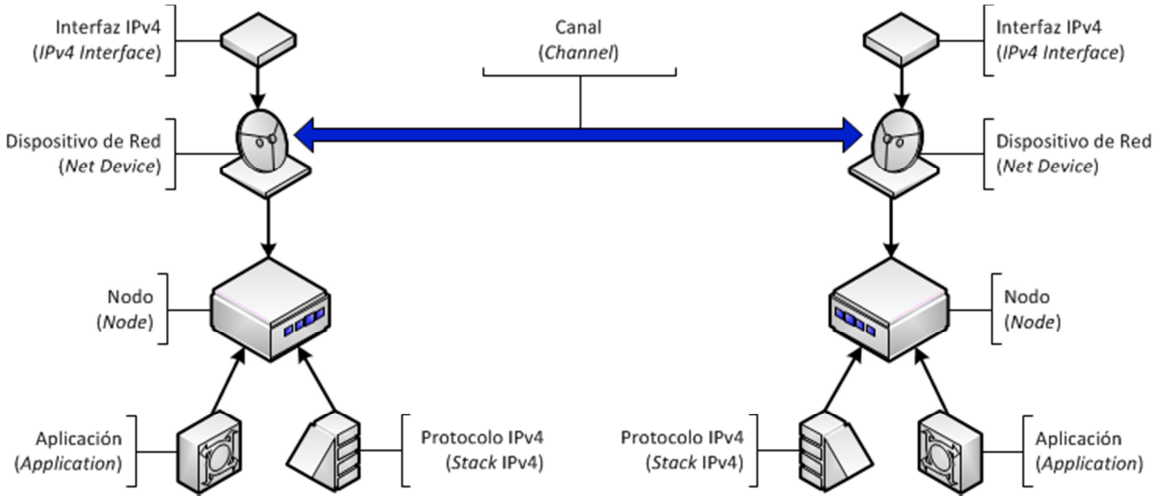

Ilustración 3-3 - Dos nodos con los objetos necesarios para comunicarse.

Esta modularidad le brinda una gran flexibilidad al simulador no solo para la creación de escenarios, sino para el desarrollo de nuevos módulos y funciones adicionales a las originales las cuales se pueden agregar en la medida de lo requerido.

- *Instalación*

Para la instalación del simulador, se usará como base una instalación en limpio de Linux Ubuntu versión 13.10. Luego de descargar e instalar todas las dependencias y paquetes necesarios previos a la instalación, se seguirá como guía las instrucciones en línea del simulador en la página web http://www.nsnam.org/wiki/Installation.

Una vez instaladas las dependencias previas, se descarga e instala el simulador como tal utilizando primeramente la herramienta *Bake*, el cual automatiza la descarga y descompresión de los archivos necesarios. Igualmente, siguiendo las instrucciones provistas por el manual de instalación en línea. Luego de descargado, se compila y configura el simulador utilizando la herramienta *Waf*, con la que adicionalmente se ejecutarán los escenarios a simular. Al finalizar, el simulador se encontrará listo para ser utilizado.

Network Simulator ha sido uno de los simuladores de redes más importantes que existen, gracias a su flexibilidad, apoyo de la comunidad, gran documentación, y constantes actualizaciones por parte del grupo de desarrolladores.

4. Implementación de los Escenarios y Ejecución de las Pruebas

En este capítulo, se describe el diseño, implementación y ejecución de las pruebas y sus distintos escenarios.

4.1. Escenario base

El escenario base consta de tres tipos de nodos:

- *Nodo Internet:*

Este nodo actúa como un equipo en Internet, con los cuales los clientes mallados se comunicarán para simular acceso fuera de la red mallada a través de un enlace compartido.

- *Punto de acceso mallado:*

Es un equipo que se encuentra situado entre Internet y la zona inalámbrica. Simula un punto de acceso con conectividad a Internet que comparte dicho acceso a una red mallada inalámbrica.

- *Estaciones clientes mallados:*

Son todos los equipos móviles, que presentan conectividad inalámbrica y comportamiento de una red Ad-Hoc, simulando así a los clientes finales.

Debido al uso de dos tecnologías de comunicación, se destacan las siguientes zonas:

- *Zona de Internet:*

En esta se encuentra el enlace punto a punto que conecta al punto de acceso mallado a Internet. Este enlace puede variar su velocidad, para así simular varios casos de conectividad con Internet con distintos anchos de bandas provistos por algún ISP. Incluye un enlace entre Internet y el punto de acceso mallado.

- *Zona mallada:*

Aquí se encuentran los equipos con conectividad inalámbrica, siendo en este caso el punto de acceso y los clientes finales mallados.

En la ilustración 4-1, se pueden apreciar los distintos elementos anteriormente mencionados más claramente:

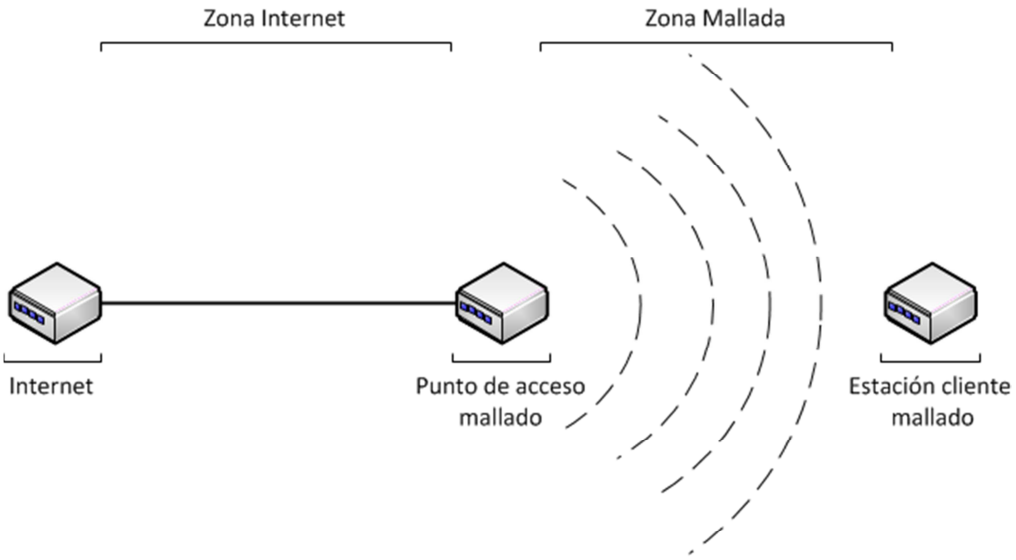

Ilustración 4-1 - Elementos básicos de los escenarios.

El punto de acceso se encuentra centrado en la malla verticalmente, a una distancia fija horizontal de la misma. Para mallas con número vertical de nodos impar, el punto de acceso siempre quedará a la misma altura que el nodo central, como se puede observar en la ilustración 4-2 en una malla de 3x3.

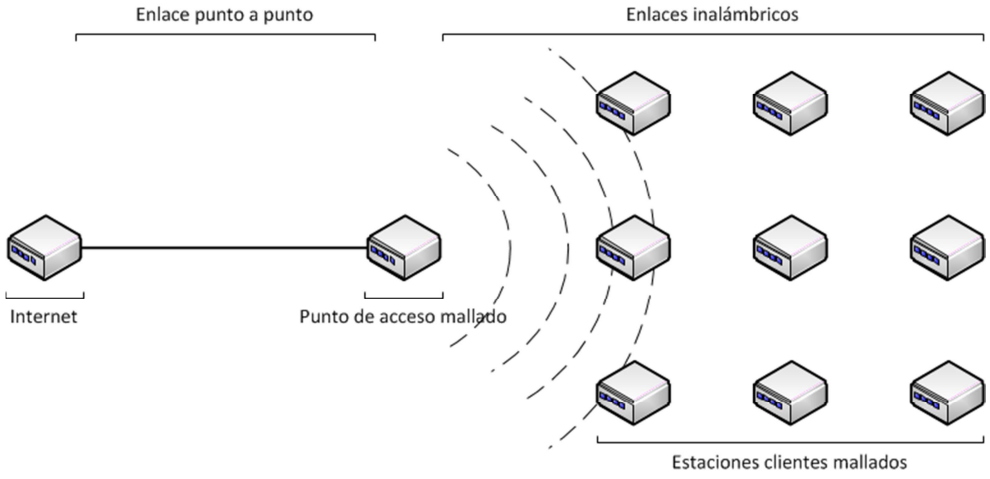

Ilustración 4-2 - Ejemplo de un escenario con altura impar.

Para las mallas donde el número de nodos vertical es par, el punto de acceso mallado se encuentra a la misma altura que el nodo situado anterior a la mitad de la altura total. Esto es para no alterar la cantidad de nodos que alcanzan directamente al punto de acceso en relación a las mallas con nodos impares, posiblemente modificando los resultados. En la ilustración 4-3 se detalla como en una red mallada de 4 nodos de altura el punto de acceso se encuentra a la misma altura del segundo nodo (el nodo ubicado antes de la mitad de la altura en distancia).

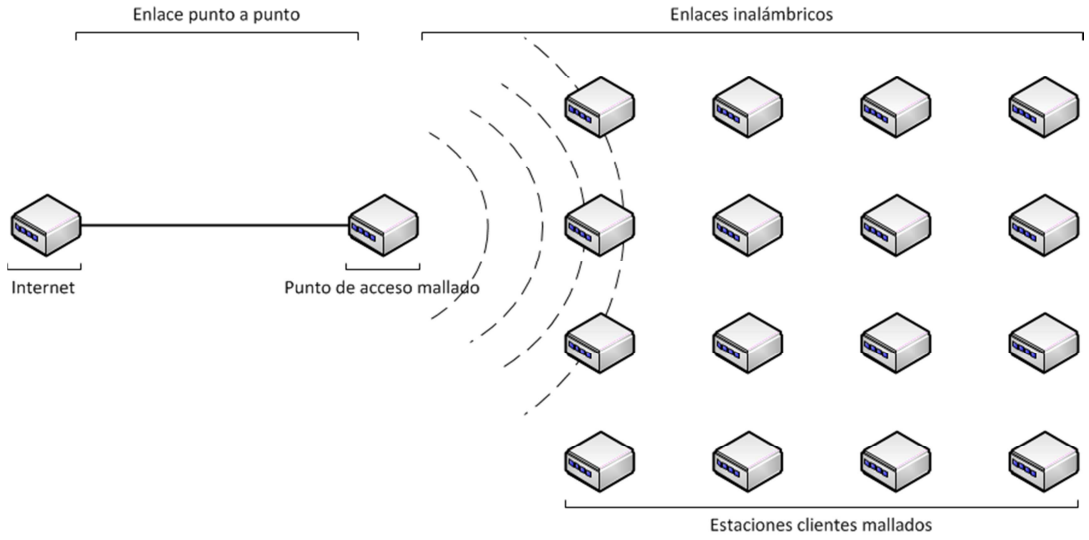

Ilustración 4-3 - Ejemplo de un escenario con altura par.

4.2. Escogencia y descripción de los escenarios

Luego de tener un escenario base, es necesario establecer que parámetros variar para la obtener distintos casos a analizar, y así realizar luego las comparaciones pertinentes.

Los elementos a variar fueron los siguientes:

- *Protocolos de enrutamientos mallados inalámbricos:*

Este es el principal elemento a variar, ya que es necesario revisar los distintos protocolos de enrutamiento para analizar cómo afectan el comportamiento en la red.

Se escogieron los protocolos Optimized Link State Routing Protocol (OLSR) y Hybrid Wireless Mesh Protocol (HWMP) tanto en su modo reactivo como proactivo. Fueron tomados para comparar un protocolo básico, en el que el enrutamiento dentro de la red mallada se realiza (siguiendo el modelo TCP/IP) en la capa de red (IP), como es el caso de OLSR, y otro en la capa de enlace (MAC) en HWMP. Como razón adicional, se tomó el protocolo HWMP debido a que es el protocolo predefinido a utilizar en el estándar IEEE 802.11s, para así tener una referencia del funcionamiento de este estándar en comparación de uno base.

- *Número de estaciones clientes mallados:*

El número de clientes es un factor básico para alterar el funcionamiento de la red. Para mantener la forma de la malla constante, se eligieron los tamaños para la malla de 3x3 (9 nodos), 4x4 (16 nodos), 5x5 (25 nodos) y 6x6 (36 nodos).

- *Flujos de carga o descarga de Internet:*

Por lo general, un usuario doméstico suele utilizar una mayor parte del tiempo de navegación en descargas (descargas de páginas, archivos, imágenes, etc.), mientras que el tráfico de subida tiene un patrón diferente. Para este trabajo, se decidió realizar pruebas con tráfico basado en solo descargas, y las mismas pruebas con tráfico de subida en búsqueda de algún cambio en el comportamiento de la red.

- *Ancho de banda en el enlace a Internet:*

Tomando en cuenta que uno de los factores principales que afecta el rendimiento de una red es el ancho de banda del proveedor del servicio de Internet, se decidió limitar el mismo a dos de las velocidades más comunes ofrecidas por los ISP para hogar [22][23], las cuales son 1 Mbps y 2 Mbps, velocidades básicas para un servicio de Internet para casa.

A continuación en la tabla 4-1 se describen los escenarios generados como resultados de los elementos anteriormente mencionados, en los cuales se ejecutaran las pruebas con los diferentes protocolos de enrutamiento para determinar el rendimiento de la red. Cada prueba en cada escenario se repetirá 30 veces, ejecutándose con cada uno de los protocolos para tener unos resultados muestralmente aceptables.

Escenario	Número de nodos mallados	Dirección de flujo	Ancho de banda
1	3x3 (9 + nodo de acceso)	Descarga	1 Mbps
2	3x3 (9 + nodo de acceso)	Carga	1 Mbps
3	3x3 (9 + nodo de acceso)	Descarga	2 Mbps
4	3x3 (9 + nodo de acceso)	Carga	2 Mbps
5	4x4 (16 + nodo de acceso)	Descarga	1 Mbps
6	4x4 (16 + nodo de acceso)	Carga	1 Mbps
7	4x4 (16 + nodo de acceso)	Descarga	2 Mbps
8	4x4 (16 + nodo de acceso)	Carga	2 Mbps
9	5x5 (25 + nodo de acceso)	Descarga	1 Mbps
10	5x5 (25 + nodo de acceso)	Carga	1 Mbps
11	5x5 (25 + nodo de acceso)	Descarga	2 Mbps
12	5x5 (25 + nodo de acceso)	Carga	2 Mbps
13	6x6 (36 + nodo de acceso)	Descarga	1 Mbps
14	6x6 (36 + nodo de acceso)	Carga	1 Mbps
15	6x6 (36 + nodo de acceso)	Descarga	2 Mbps
16	6x6 (36 + nodo de acceso)	Carga	2 Mbps

Tabla 4-1 - Pruebas a ejecutar para cada protocolo de enrutamiento.

4.3. Métricas utilizadas en el desarrollo

Las medidas utilizadas para estimar el desempeño de la red son las siguientes:

4.3.1. Rendimiento de tráfico (throughput)

El rendimiento, o throughput, es definido como la cantidad de datos que son enviados entre dos puntos en un tiempo determinado, y llegan correctamente a su destino [20]. En redes, es usual utilizar bits transmitidos en un segundo (bps) o bytes transmitidos por segundo (Bps), o alguna unidad múltiplo en cuanto a la cantidad de datos (kilobits por segundo, megabytes por segundo, etc.).

4.3.2. Pérdida de paquetes

En una transmisión, los datos pueden no llegar siempre a su destino. Estos pueden corromperse durante su envío, ser descartados en algún equipo intermedio, etc. La pérdida de paquetes representa la relación entre los paquetes enviados y los paquetes recibidos correctamente. En este trabajo, la pérdida de paquetes se representa en términos de porcentaje, relacionando cuantos paquetes de los enviados no llegaron al destino sea cual fuere la razón.

4.3.3. Retraso extremo a extremo

Las transmisiones de datos siempre tienen cierto retraso inherentes al como viaja la información a través del medio, tiempo de procesamiento en los equipos, etc. También pueden existir otros eventos que agreguen algún retraso adicional a la llegada del paquete, como retraso en cola, retrasos por paquetes perdidos y reenviados, etc. En el trabajo presente, el retraso extremo a extremo es el tiempo (en milisegundos) que toma un paquete en salir del dispositivo de red desde origen hasta que es recibido por el destino.

4.3.4. Fluctuación (jitter)

Cada paquete enviado puede presentar diferentes retrasos en su transmisión [21], bien sea por congestión de la red, falla o interrupción del envío, uso de una ruta diferente por donde transitar, etc. A dicha variabilidad en los retrasos se le llama Jitter, y es representado en este trabajo en milisegundos.

4.3.5. Retraso inicial

Al inicio de un envío de una serie de paquetes, pueden existir mecanismos adicionales antes de la transmisión de datos como tal. Por ejemplo, puede ser necesario negociar la conexión, o la búsqueda de una ruta para llegar al destino desde el origen. Esto puede proporcionar un retraso inicial adicional al retraso estándar antes de comenzar a transmitir información. En este Trabajo Especial de Grado, el retraso inicial de un flujo de datos específico es la diferencia entre el tiempo cuando es enviado el primer paquete con el tiempo que es recibido el mismo, y medido en milisegundos.

4.4. Definición de parámetros adicionales

Adicional a los escenarios definidos anteriormente, existe una cantidad de parámetros inherentes a la simulación que son necesarios establecer. Estos son:

- *Estándar 802.11:*

Se decidió utilizar el estándar IEEE 802.11g a 6 Mbps como base para las comunicaciones inalámbricas.

- *Distancia de la red mallada al punto de acceso:*

Para tomar la distancia entre el nodo de acceso y los nodos clientes, se basó en la información anteriormente documentada en un Trabajo Especial de Grado [24], donde se señalan distintas barriadas que limitan con edificios residenciales. Luego de medir la distancia entre diferentes edificios y las barriadas (ver anexo A con las ilustraciones de las barriadas medidas), se tomó 50 metros como el valor promedio redondeado entre la separación entre la red mallada y el nodo de acceso mallado.

- *Tiempo de simulación:*

Se tomó un tiempo simulado por ejecución de 120 segundos.

- *Rango de alcance para los dispositivos inalámbricos:*

Si el punto de acceso mallado lograra cubrir todos los nodos clientes, estos se conectarían directamente a él, haciendo del protocolo de enrutamiento superfluo en este escenario. Por tanto, se eligió una distancia de 65 metros como rango máximo, para obligar a la red mallada (ubicada a 50 metros de distancia) establecer comunicación solo con el nodo más cercano, sin cubrir en su totalidad a los clientes. En la ilustración 4-4 se pueden observar las separaciones y rango del punto de acceso mallado con la malla como tal.

Para los nodos mallados, la distancia entre ellos es de 30 metros, por lo que un nodo mallado puede establecer comunicación con cualquier otro nodo alrededor. En la ilustración 4-5 se puede observar las separaciones y rango de una estación cliente mallado.

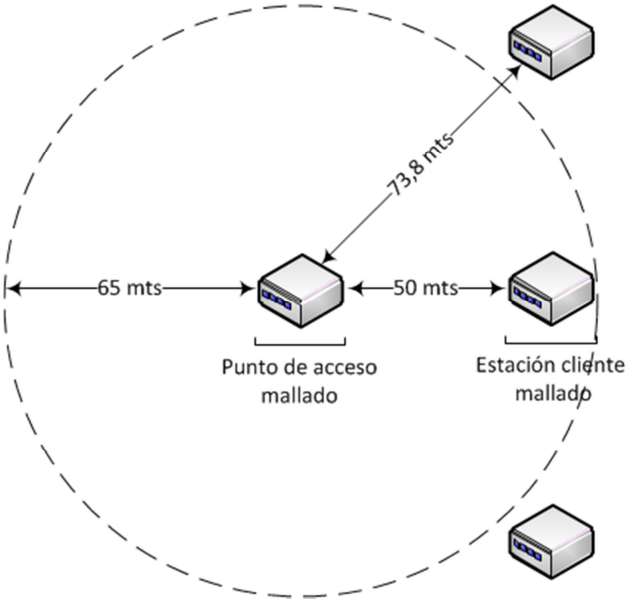

Ilustración 4-4 - Distancias a un punto de acceso mallado.

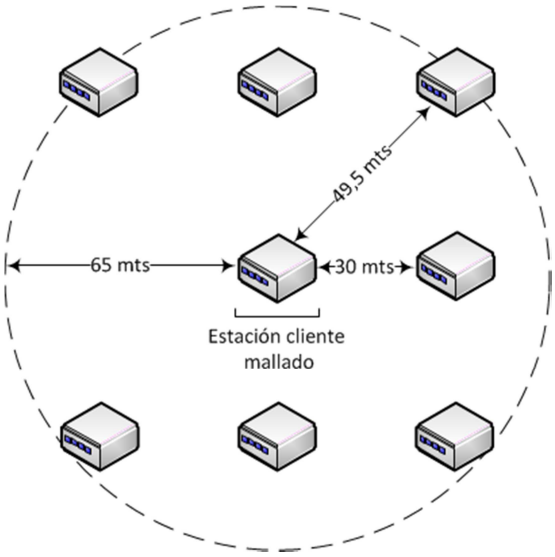

Ilustración 4-5 - Distancias a una estación cliente mallado.

- *Aplicaciones generadoras de tráfico:*

Las aplicaciones son necesarias para crear carga suficiente en la red, y así simular el funcionamiento de la misma. En el caso del trabajo presente, se creó una aplicación para cada camino de datos que se necesita establecer. En un camino de datos se indica que nodo es el origen y que nodo el destino,

activándose una aplicación generadora de tráfico con ambos. Para los casos de carga, en cada nodo mallado se configuró una aplicación generadora de tráfico con destino el nodo de Internet (ilustración 4-6). En los casos de descarga, al nodo que representa Internet se le instaló una aplicación por cada nodo mallado, cada uno apuntando a un único nodo destino (ilustración 4-7).

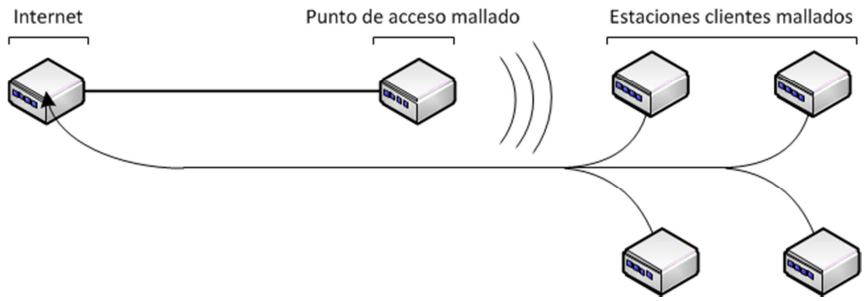

Ilustración 4-6 - Flujos de carga desde cada cliente mallado.

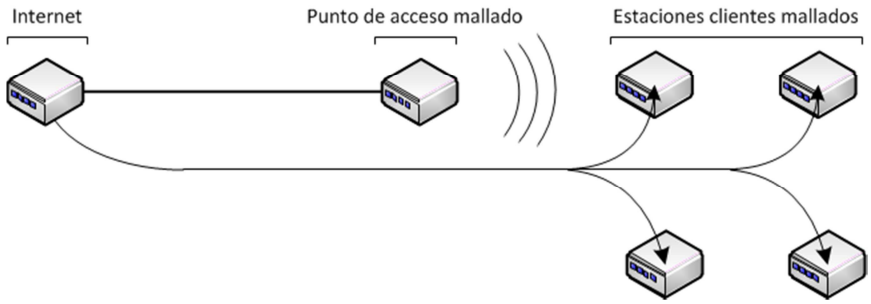

Ilustración 4-7 - Flujos de descarga desde Internet a cada cliente mallado.

Las aplicaciones generan datos continuamente hasta el final a una tasa de 256 kbps. Cada paquete enviado lleva un tamaño de 128 bits. Con esto se genera una mayor cantidad de paquetes, que pueden ser tratados independientemente en las transmisiones.

4.5. Implementación de los escenarios

Para implementar un escenario en el simulador Network Simulator 3, se ingresan todos los datos requeridos en un código basado en C++. Este código es

llamado guion (o *script*), el cual luego el simulador utiliza como guía para ejecutar la simulación como tal.

En el caso del presente trabajo, se tienen dos scripts que, aunque similares, en uno se utiliza el protocolo OLSR, mientras en el segundo se utiliza HWMP como protocolo de enrutamiento más el estándar IEEE 802.11s.

Al comienzo de cada script, se crean todos los nodos a utilizar y se colocan en un contenedor común. Luego, se crean contenedores adicionales donde se guardaran algunos nodos según su clasificación, para el uso futuro en el código. En la ilustración 4-8 se observa el segmento del script donde se realizan estas creaciones.

```
125 // Node container creation (all node containers starts with "nc_")
126   NodeContainer nc_all; // Contains every node (starting with internet node, access nodes and then mesh nodes
127   NodeContainer nc_mesh; // Contains only mesh nodes (x*y nodes)
128   NodeContainer nc_wireless; // Contains access and mesh nodes
129   NodeContainer nc_internet; // Contains internet nodes
130   NodeContainer nc_destiny; // Contains nodes with sink for apps (internet and mesh nodes)
131   nc_mesh.Create (m_xNodes * m_yNodes);
132   nc_all.Create (2);
133   nc_all.Add (nc_mesh);
134   nc_wireless.Add (nc_all.Get (1));
135   nc_wireless.Add (nc_mesh);
136   nc_internet.Add (nc_all.Get (0));
137   nc_internet.Add (nc_all.Get (1));
138   nc_destiny.Add (nc_all.Get (0));
139   nc_destiny.Add (nc_mesh);
```

Ilustración 4-8 - Creación de los nodos.

Más adelante, en la ilustración 4-9, se crea el enlace punto a punto, que conectará el nodo Internet con el punto de acceso mallado. Es de notar que este enlace tiene un retraso fijo de 1 milisegundo, para no causar diferentes resultados en los flujos de datos.

```
141 // Create P2P links between n0 <-> n1 (all devices starts with "de_")
142   PointToPointHelper p2pinternet;
143   p2pinternet.SetDeviceAttribute ("DataRate", StringValue (m_txInternetRate));
144   p2pinternet.SetChannelAttribute ("Delay", StringValue ("1ms"));
145   NetDeviceContainer de_internet;
146   de_internet = p2pinternet.Install (nc_internet);
```

Ilustración 4-9 - Creación del enlace punto a punto.

El siguiente paso es definir y crear el canal inalámbrico, la capa física a utilizar y el estándar en la capa de enlace. En la ilustración 4-10 se tiene la definición para configurar todo esto en preparación para el protocolo OLSR, en el que se utiliza el ayudante *WifiHelper*.

```
144 // Set channel, phy and mac layers
145   WifiHelper wifi;
146   YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
147   YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
148   NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
149   wifi.SetStandard (WIFI_PHY_STANDARD_80211g);
150   wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
151                                 "DataMode",StringValue ("ErpOfdmRate6Mbps"),
152                                 "ControlMode",StringValue ("ErpOfdmRate6Mbps")); //Valid ErpOfdmRate: 6 9 12 18 24 36 48 54
153   wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
154
155   wifiChannel.AddPropagationLoss ("ns3::RangePropagationLossModel","MaxRange", DoubleValue (65));
156   wifiMac.SetType ("ns3::AdhocWifiMac");
157   wifiPhy.SetChannel (wifiChannel.Create ());
158   NetDeviceContainer de_wireless = wifi.Install (wifiPhy, wifiMac, nc_wireless);
```

Ilustración 4-10 - Configuración inalámbrica para OLSR.

Para el caso de usar el estandar 802.11s y HWMP, el código varia ligeramente ya que es necesario usar otro ayudante, en este caso *MeshHelper*. Se tiene un condicional para elegir entre el modo reactivo en HWMP y el proactivo. El código a usar se encuentra en la ilustración 4-11.

```
148 // Set channel, phy and mac layers
149   YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
150   YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
151   wifiChannel.AddPropagationLoss ("ns3::RangePropagationLossModel","MaxRange", DoubleValue (65));
152   wifiPhy.SetChannel (wifiChannel.Create ());
153   MeshHelper mesh;
154   mesh = MeshHelper::Default ();
155   mesh.SetStandard (WIFI_PHY_STANDARD_80211g);
156   mesh.SetMacType ("RandomStart", TimeValue (Seconds(0.1)));
157   mesh.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
158                                 "DataMode", StringValue ("ErpOfdmRate6Mbps"),
159                                 "ControlMode",StringValue ("ErpOfdmRate6Mbps")); //Valid ErpOfdmRate: 6 9 12 18 24 36 48 54
160   if (m_root)
161   { // If proactive mode is enable, access node will be root 00:00:00:00:00:03
162     printf ("Proactive mode set!\n");
163     mesh.SetStackInstaller (m_stack, "Root", Mac48AddressValue (Mac48Address ("00:00:00:00:00:03")));
164   }
165   else
166   {
167     printf ("Reactive mode set!\n");
168     mesh.SetStackInstaller (m_stack);
169   }
170   mesh.SetSpreadInterfaceChannels (MeshHelper::ZERO_CHANNEL);
171   mesh.SetMacType ("RandomStart", TimeValue (Seconds (2)));
172   mesh.SetNumberOfInterfaces (1);
173   NetDeviceContainer de_wireless = mesh.Install (wifiPhy, nc_wireless);
```

Ilustración 4-11 - Configuración inalámbrica para 802.11s y HWMP.

En estos últimos dos algoritmos, aunque se aplican de diferentes maneras, se observan puntos en común. Uno de ellos es la definición del rango máximo a 65 metros de distancia para los equipos inalámbricos. Igualmente, además de definir el estándar WiFi a 802.11g, se fija la velocidad interna a trabajar en WiFi a 6 Mbps. Al final, se crean los dispositivos inalámbricos y se asignan a los nodos correspondientes.

Luego, se define como protocolo con el que trabajar a IPv4. Al mismo tiempo se configura como protocolo de enrutamiento en capa 3 a OLSR

(ilustración 4-12), mientras que para HWMP no es necesario por trabajar en capa 2 (ilustración 4-13).

```
212  // Set Internet stack
213    InternetStackHelper internetStack;
214    internetStack.Install (nc_all);
```

Ilustración 4-12 - Configuración de la pila IPv4 para 802.11s.

```
197  // Set Internet stack and OLSR protocol
198    OlsrHelper olsr;
199    InternetStackHelper internetStack;
200    internetStack.SetRoutingHelper (olsr);
201    internetStack.Install (nc_all);
```

Ilustración 4-13 - Configuración de la pila IPv4 para OLSR.

Es necesario definir las direcciones IP en todos los nodos, y la ilustración 4-14 se muestra el algoritmo encargado de realizar dicha asignación.

```
216  // Config IPs for interfaces (all interfaces starts with "if_")
217    Ipv4AddressHelper addrMesh;
218    addrMesh.SetBase ("10.0.0.0", "255.255.255.0");
219    Ipv4InterfaceContainer if_mesh = addrMesh.Assign (de_wireless);
220
221    Ipv4AddressHelper addrInternet;
222    addrInternet.SetBase ("1.1.1.0", "255.255.255.0");
223    Ipv4InterfaceContainer if_internet = addrInternet.Assign (de_internet);
```

Ilustración 4-14 - Asignación de la direcciones IPv4.

Sin embargo, siendo HWMP un protocolo de enrutamiento en capa 2, aún es necesario tener en la tabla de enrutamiento de capa 3 como accesar a otras redes aparte de la propia. Para el caso del presente trabajo, se utilizaron rutas estáticas en los nodos. Al no requerir procesamiento adicional en busqueda de rutas (si se encuentra el nodo en la misma red se entrega mediante entrega directa o HWMP, sino se entrega a puerta de enlace predeterminada igualmente por entrega directa o HWMP) no agrega retrasos adicionales a la red. En el algoritmo de la ilustración 4-15 se establece al nodo intermedio (el punto de acceso mallado) como puerta de enlace predeterminada tanto para el nodo Internet, como para los clientes mallados .

```
225  // Set default L3 routes
226    Ipv4StaticRoutingHelper ipv4RoutingHelper;
227    Ptr<Ipv4> ipv4;
228    Ptr<Ipv4StaticRouting> staticRouting;
229    // Default gateway for node 0 (send all to node 1)
230    ipv4 = nc_all.Get (0)->GetObject<Ipv4> ();
231    staticRouting = ipv4RoutingHelper.GetStaticRouting (ipv4);
232    staticRouting->SetDefaultRoute (Ipv4Address ("1.1.1.2"), 1, 1);
233
234    // Default gateway for mesh nodes (send all to access point)
235    for (tmp_x = 2; tmp_x < m_xNodes * m_yNodes + 2; tmp_x++)
236    {
237      ipv4 = nc_all.Get (tmp_x)->GetObject<Ipv4> ();
238      staticRouting = ipv4RoutingHelper.GetStaticRouting (ipv4);
239      staticRouting->SetDefaultRoute (Ipv4Address ("10.0.0.1"), 1, 1);
240    }
```

Ilustración 4-15 - Creación de las rutas estáticas para HWMP.

Otro punto básico es la creación de aplicaciones que sean capaces de recibir información y registrar los datos para su análisis. En NS-3 las aplicaciones utilizadas para esta recolección de datos son llamadas *sinks*. En el algoritmo de la ilustración 4-16 se crean e instalan dichas aplicaciones a todos los nodos, con excepción del punto de acceso mallado, el cual no genera ni recibe ningún tráfico de importancia.

```
212  // Install applications
213  // Set sinks for receiving data
214    PacketSinkHelper sinkTcp ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), 9));
215    ApplicationContainer ac_sinkTcp = sinkTcp.Install (nc_destiny);
216    ac_sinkTcp.Start (Seconds (0));
217    ac_sinkTcp.Stop (Seconds (m_totalTime));
```

Ilustración 4-16 - Creación de aplicaciones *sinks*.

Luego, se crean las aplicaciones que generarán carga en la red. Es de notar en el script de la ilustración 4-17 que puede ser ejecutado solo generando tráfico de carga a Internet, tráfico de descarga, o ambos inclusive. Es importante destacar que las aplicaciones comienzan obligatoriamente en segundo 15 de la simulación, para darle tiempo a los protocolos de estabilizarse.

```
255   if (m_flow == "upload" || m_flow == "both")
256   {
257     ApplicationContainer ac_onoffMesh [m_xNodes * m_yNodes]; // Creates 1 OnOff App in each mesh node TO internet
258     OnOffHelper onoffMesh ("ns3::TcpSocketFactory", Address (InetSocketAddress (if_internet.GetAddress (0), 9)));
259     for (tmp_x = 0; tmp_x < m_xNodes * m_yNodes; tmp_x++)
260     {
261       ac_onoffMesh [tmp_x] = onoffMesh.Install (nc_mesh.Get (tmp_x));
262       ac_onoffMesh [tmp_x].Start (Seconds (15 + rand () % 10));
263       ac_onoffMesh [tmp_x].Stop (Seconds (m_totalTime - 1 - rand () % 10));
264     }
265   }
266
267   if (m_flow == "download" || m_flow == "both")
268   {
269     ApplicationContainer ac_onoffInternet [m_xNodes * m_yNodes]; // Creates 1 OnOff App for each mesh node FROM internet
270     for (tmp_x = 0; tmp_x < m_xNodes * m_yNodes; tmp_x++)
271     {
272       OnOffHelper onoffInternet ("ns3::TcpSocketFactory", Address (InetSocketAddress (if_mesh.GetAddress (tmp_x + 1), 9)));
273       ac_onoffInternet [tmp_x] = onoffInternet.Install (nc_all.Get (0));
274       ac_onoffInternet [tmp_x].Start (Seconds (15 + rand () % 10));
275       ac_onoffInternet [tmp_x].Stop (Seconds (m_totalTime - 1 - rand () % 10));
276     }
277   }
```

Ilustración 4-17 - Creación de aplicaciones generadoras de tráfico.

Existen dos aplicaciones adicionales creadas con fines de monitoreo. Estas aplicaciones son comandos *ping* entre el nodo de Internet y el último cliente mallado en ambas vias, como se ve en la ilustración 4-18. Estas son utilizadas para observar el retraso en vivo (durante la ejecución del simulación) de la red de un extremo a otro, y comprobar a su vez que existe comunicación entre los nodos más alejados.

```
279   /////PING FOR TESTS
280     V4PingHelper ping1 (if_internet.GetAddress (0));
281     ping1.SetAttribute ("Verbose", BooleanValue (true));
282     ping1.SetAttribute ("Interval", TimeValue (Seconds (1)));
283     ApplicationContainer appPingInternet1 = ping1.Install (nc_all.Get (m_xNodes * m_yNodes + 1));
284     appPingInternet1.Start (Seconds (0));
285     appPingInternet1.Stop (Seconds (m_totalTime - 1));
286
287     V4PingHelper ping2 (if_mesh.GetAddress (m_xNodes * m_yNodes));
288     ping2.SetAttribute ("Verbose", BooleanValue (true));
289     ping2.SetAttribute ("Interval", TimeValue (Seconds (1)));
290     ApplicationContainer appPingInternet2 = ping2.Install (nc_all.Get (0));
291     appPingInternet2.Start (Seconds (0));
292     appPingInternet2.Stop (Seconds (m_totalTime - 1));
293   /////END PING FOR TESTS
```

Ilustración 4-18 - Creación de aplicaciones *ping*.

Para la recolección de los datos provenientes de las aplicaciones *sink*, es necesario utilizar el ayudante FlowMonitor (ilustración 4-19), el cual permitirá más adelante tener acceso a todos los flujos de datos registrados y exportarlos a un formato más manejable.

```
316  // Flow Monitor
317    FlowMonitorHelper flowmon;
318    Ptr<FlowMonitor> monitor = flowmon.Install (nc_destiny);
319    monitor->Start (0);
320    monitor->Stop (m_totalTime);
```

Ilustración 4-19 - Creación del monitor de tráfico *FlowMonitor*.

En este segmento de código (ilustración 4-20) se toman los datos obtenidos por el ayudante FlowMonitor y los exporta a un archivo con extensión .CSV (*Comma Separated Values*, Valores Separados por Coma). Con este formato, se guardan los datos conteniendo las estadísticas de cada flujo de datos. Cada flujo de datos está definido por una quíntupla, que contiene direccion IP orígen, IP destino, puerto orígen, puerto destino y tipo de protocolo. Cada flujo contiene como estadísticas el tiempo donde se emitió el primer y último paquete del orígen, el tiempo donde se recibió en el destino el primer y último paquete, bytes transmitidos, bytes recibidos, paquetes transmitidos, paquetes recibidos, la suma total de los retrasos de todos los paquetes en el flujo, y la suma total de todos los jitters en todos los paquetes del flujo.

```
322  //Print per flow statistics
323    monitor->CheckForLostPackets ();
324    Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
325    std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
326    for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)
327    {
328      Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
329      of << t.sourceAddress << "\t" << t.destinationAddress << "\t";
330      if (t.protocol == 6)
331      {
332        of << "TCP\t";
333      }
334      else
335      {
336        if (t.protocol == 17)
337        {
338          of << "UDP\t";
339        }
340        else
341        {
342          of << "N/A\t";
343        }
344      }
345      of << t.sourcePort << "-" << t.destinationPort << "\t";
346      of << i->second.timeFirstTxPacket.GetSeconds() << "\t" << i->second.timeLastTxPacket.GetSeconds() << "\t";
347      of << i->second.timeFirstRxPacket.GetSeconds() << "\t" << i->second.timeLastRxPacket.GetSeconds() << "\t";
348      of << i->second.txBytes << "\t" << i->second.rxBytes << "\t";
349      of << i->second.txPackets << "\t" << i->second.rxPackets << "\t";
350      of << i->second.delaySum << "\t" << i->second.jitterSum << "\n";
351    }
352    of.close ();
```

Ilustración 4-20 - Extracción de las estadísticas de los flujos de datos.

El resultado de los datos luego de extraidos y colocados en una hoja de cálculo es similar a lo que se observa en la tabla 4-2:

Origen	Destino	Tiempo Primer Tx (seg)	Tiempo Último Tx (seg)	Tiempo Primer Rx (seg)	Tiempo Último Rx (seg)	Total Tx (bytes)	Total Rx (bytes)	Total Tx (Paquetes)	Total Rx (Paquetes)	Retraso Total	Delay Total
1.1.1.1	10.0.0.4	15,00	87,88	15,01	88,29	6.536	5.960	16	15	6.237.737.915	2.449.329.902
1.1.1.1	10.0.0.11	15,00	95,15	15,02	95,61	43.280	37.520	77	67	25.759.426.491	3.951.709.953
1.1.1.1	10.0.0.25	15,00	119,44	15,01	119,80	157.032	148.392	323	308	78.953.044.122	6.021.831.051
1.1.1.1	10.0.0.10	15,00	114,34	15,03	107,95	217.808	196.496	380	343	172.394.878.022	8.206.431.683
1.1.1.1	10.0.0.21	15,00	119,81	17,02	119,81	244.344	235.048	427	409	174.329.726.541	10.245.615.348
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.

Tabla 4-2 - Datos básicos extraídos de los flujos de datos.

Adicionalmente a las columnas originales, se agregaron a cada flujo (como están en la tabla 4-3) los campos retraso y jitter promedio por paquete, throughput total del flujo, porcentaje de paquetes perdidos y retraso inicial.

Delay Prom. Por Paquete (ms)	Jitter Prom. Por Paquete (ms)	Throughput Prom. Por Flujo (kbps)	% Paquetes Perdidos	Retraso Inicial (ms)
415,85	163,29	0,64	6,25	8,20
384,47	58,98	3,64	12,99	18,30
256,34	19,55	11,06	4,64	12,70
502,61	23,93	16,52	9,74	34,50
426,23	25,05	17,52	4,22	2.019,40
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Tabla 4-3 - Datos adicionales generados a partir de los básicos.

Para configurar el tiempo de duración del simulador se utiliza el código de la ilustración 4-21, mientras que con el código de la ilustración 4-22 se ejecuta el script. Para finalizar, se limpia el estado del simulador y cierra el código en la ilustración 4-23.

```
296  // Set simulation time
297    Simulator::Stop (Seconds (m_totalTime));
```

Ilustración 4-21 - Configuración del tiempo total de simulación.

```
326   // Run the simulation
327     Simulator::Run ();
```

Ilustración 4-22 - Ejecución de la simulación.

```
384     Simulator::Destroy ();
385     return 0;
```

Ilustración 4-23 - Finalización de la simulación.

4.6. Ejecución de pruebas

Al ser un ambiente simulado, una vez configurado el simulador en el ambiente de trabajo no es requerido ningún requerimiento adicional al de ejecutar la prueba directamente. Para esta ejecución se usa la herramienta *Waf*, la cual configura, compila y ejecuta el código del simulador y el *script*.

En la ilustración 4-24, se ejecuta *Waf* únicamente para compilar todo el código del simulador, con el que se comprueba cualquier falla que se pueda generar en tiempo de compilación.

```
tesis@tesis:~/source/ns-3.19$ ./waf
Waf: Entering directory `/home/tesis/source/ns-3.19/build'
  Waf: Leaving directory `/home/tesis/source/ns-3.19/build'
'build' finished successfully (1.328s)
.
.
.
tesis@tesis:~/source/ns-3.19$ _
```

Ilustración 4-24 - Compilación del simulador.

Para correr una simulación, se utiliza el parámetro “--run [archivo]” para indicar la aplicación previamente compilada a ejecutar. En la siguiente ilustración 4-25, se observa la ejecución de la aplicación “basev11-olsr”, él cual es el *script* más reciente realizado para OLSR al momento de ser escrito el presente documento.

```
tesis@tesis:~/source/ns-3.19$ ./waf --run basev11-olsr
Waf: Entering directory `/home/tesis/source/ns-3.20/build'
Waf: Leaving directory `/home/tesis/source/ns-3.20/build'
'build' finished successfully (1.543s)
Initial random seed: 786
PING  1.1.1.1 56(84) bytes of data.
PING  10.0.0.10 56(84) bytes of data.
recv 84 bytes
recv 84 bytes
64 bytes from 10.0.0.10: icmp_seq=11 ttl=61 time=37 ms
recv 84 bytes
64 bytes from 10.0.0.10: icmp_seq=12 ttl=61 time=5 ms
recv 84 bytes
64 bytes from 1.1.1.1: icmp_seq=12 ttl=61 time=5 ms
.
.
.
recv 84 bytes
64 bytes from 10.0.0.10: icmp_seq=28 ttl=61 time=8 ms
--- 1.1.1.1 ping statistics ---
29 packets transmitted, 17 received, 41% packet loss, time 29000ms
rtt min/avg/max/mdev = 4/116.5/1055/266.9 ms
--- 10.0.0.10 ping statistics ---
29 packets transmitted, 17 received, 41% packet loss, time 29000ms
rtt min/avg/max/mdev = 5/58/402/103.3 ms
tesis@tesis:~/source/ns-3.19$ _
```

Ilustración 4-25 - Compilación y ejecución básica.

Similarmente para HWMP se utiliza el script “basev11-hwmp”, el cual adicionalmente se le debe indicar la opción “--root=true” para activar el modo proactivo, o “--root=false” para el modo reactivo. En la ilustración 4-26 se aprecia el comando para la ejecución del simulador con el protocolo HWMP en modo proactivo.

```
tesis@tesis:~/source/ns-3.19$ ./waf --run “basev11-hwmp --
root=true”
```

Ilustración 4-26 - Uso de la opción "root".

Entre otras opciones se tiene la posibilidad de definir el archivo de salida y su ubicación con la opción “--stats-file=[archivo]”. Es de notar que solo es necesario el nombre del archivo, no su extensión, como se ve en la ilustración 4-27.

```
tesis@tesis:~/source/ns-3.19$ ./waf --run “basev11-olsr --stats-
file=resultados/basev11-olsr-3x3”
```

Ilustración 4-27 - Uso de la opción "stats-file".

Las opciones “--mesh-width=[X]” y “--mesh-height=[Y]” son utilizadas para definir el tamaño de la red mallada en termino de número de nodos, como se muestra en la siguiente ilustración 4-28.

```
tesis@tesis:~/source/ns-3.19$ ./waf --run “basev11-olsr --mesh-
width=3 --mesh-height=3”
```

Ilustración 4-28 - Uso de la opción "mesh-width" y "mesh-height".

Otra opción básica es la posibilidad de definir que aplicaciones deben activarse en la simulación, si solo las de descarga (*download*), las de carga (*upload*), ambas (*both*), o ninguna (*none*) utilizando la opción “--flow-direction=[download | upload | both | none]”. En la ilustración 4-29 se muestra una llamada con la opción de solo descargas.

```
tesis@tesis:~/source/ns-3.19$ ./waf --run “basev11-hwmp --flow-
direction="download"”
```

Ilustración 4-29 - Uso de la opción "flow-direction".

El archivo de salida puede ser rellenado por varias simulaciones sucesivas, manteniendo los datos de las simulaciones anteriores, o se puede comenzar con un archivo de salida en blanco y solo con la simulación actual. La opción para activar o desactivar esta creación de un archivo nuevo en blanco es “--new-flow-file=[true | false]”, que se muestra en la siguiente ilustración 4-30.

```
tesis@tesis:~/source/ns-3.19$ ./waf --run “basev11-hwmp --new-
flow-file=true”
```

Ilustración 4-30 - Uso de la opción "new-flow-file".

Si, por ejemplo, es necesario ejecutar dos veces la simulación y obtener todos los resultados dentro de un mismo archivo, se utilizarían los siguientes dos comandos de la ilustración 4-31:

```
tesis@tesis:~/source/ns-3.19$ ./waf --run “basev11-olsr --new-
flow-file=true --stats-file=basev11-olsr”
Waf: Entering directory `/home/tesis/source/ns-3.19/build'
Waf: Leaving directory `/home/tesis/source/ns-3.19/build'
'build' finished successfully (1.256s)
Initial random seed: 192
PING  1.1.1.1 56(84) bytes of data.
PING  10.0.0.10 56(84) bytes of data.
.
.
.
29 packets transmitted, 17 received, 41% packet loss, time 29000ms
rtt min/avg/max/mdev = 5/58/402/103.3 ms
tesis@tesis:~/source/ns-3.19$ ./waf --run “basev11-olsr --stats-
file=basev11-olsr”
Waf: Entering directory `/home/tesis/source/ns-3.19/build'
Waf: Leaving directory `/home/tesis/source/ns-3.19/build'
'build' finished successfully (1.512s)
Initial random seed: 982
PING  1.1.1.1 56(84) bytes of data.
PING  10.0.0.10 56(84) bytes of data.
.
.
.
29 packets transmitted, 17 received, 41% packet loss, time 29000ms
rtt min/avg/max/mdev = 5/58/402/103.3 ms
tesis@tesis:~/source/ns-3.19$ _
```

Ilustración 4-31 - Ejecutando varias veces un escenario.

Con esta última secuencia de comandos, se obtendría en el archivo de salida los resultados de dos ejecuciones de la simulación. Para realizar N simulaciones basta con repetir la última línea de comando N-1 veces (la primera ejecutada junto al comando de creación de archivo).

En la siguiente ilustración 4-32, se listan los archivos resultantes a una ejecución completa con el protocolo OLSR, utilizando cada uno de los escenarios enumerados anteriormente en el presente documento:

```
tesis@tesis:~/source/ns-3.19/resultados$ ls
basev11-olsr-1d-3x3_flows.csv  basev11-olsr-2d-3x3_flows.csv
basev11-olsr-1d-4x4_flows.csv  basev11-olsr-2d-4x4_flows.csv
basev11-olsr-1d-5x5_flows.csv  basev11-olsr-2d-5x5_flows.csv
basev11-olsr-1d-6x6_flows.csv  basev11-olsr-2d-6x6_flows.csv
basev11-olsr-1u-3x3_flows.csv  basev11-olsr-2u-3x3_flows.csv
basev11-olsr-1u-4x4_flows.csv  basev11-olsr-2u-4x4_flows.csv
basev11-olsr-1u-5x5_flows.csv  basev11-olsr-2u-5x5_flows.csv
basev11-olsr-1u-6x6_flows.csv  basev11-olsr-2u-6x6_flows.csv
tesis@tesis:~/source/ns-3.19/resultados$ _
```

Ilustración 4-32 - Total de archivos de salida para un protocolo en cada escenario.

Estos archivos serán importados a la herramienta de hojas de cálculo Microsoft Excel 2010, donde se realizará el ordenamiento y análisis de la información extraída.

5. **Análisis de Resultados**

En el presente capítulo se especifican los resultados obtenidos luego de simulados los escenarios anteriormente establecidos. En la siguiente sección se describen estos clasificados por métrica seleccionada (rendimiento o *throughput*, pérdida de paquetes, retraso punto a punto, fluctuación del retraso punto a punto o *jitter*, y retraso inicial), luego para finalizar se analizan de forma general los datos para obtener resultado final.

5.1. Resultados de los casos estudiados

A continuación se presentan los resultados acumulados de las pruebas. En cada gráfico se puede observar el comportamiento de las tres modalidades de protocolos de enrutamiento, en conjunto a los casos de 9, 16, 25 y 36 nodos clientes mallados.

5.1.1. Análisis del rendimiento (throughput)

En las ilustraciones 5-1, 5-2, 5-3 y 5-4 se observa el rendimiento o *throughput* promedio de los flujos de datos (de carga o descarga, según sea el caso del gráfico) en función del número de clientes mallados presentes en la red y el protocolo utilizado.

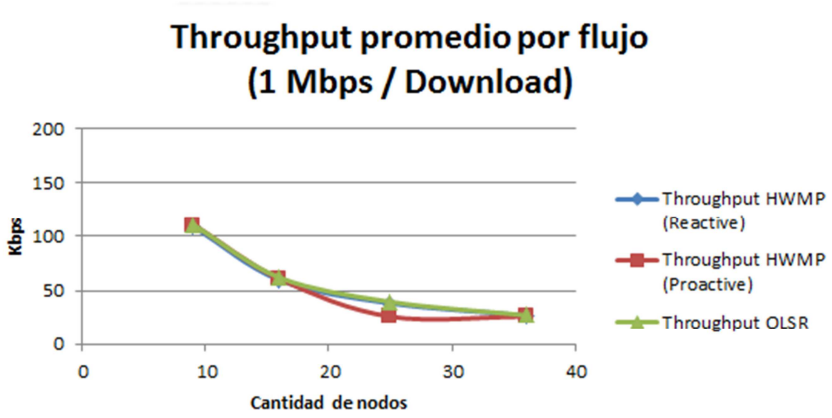

Ilustración 5-1 - Resultados del rendimiento para 1 Mbps con descargas.

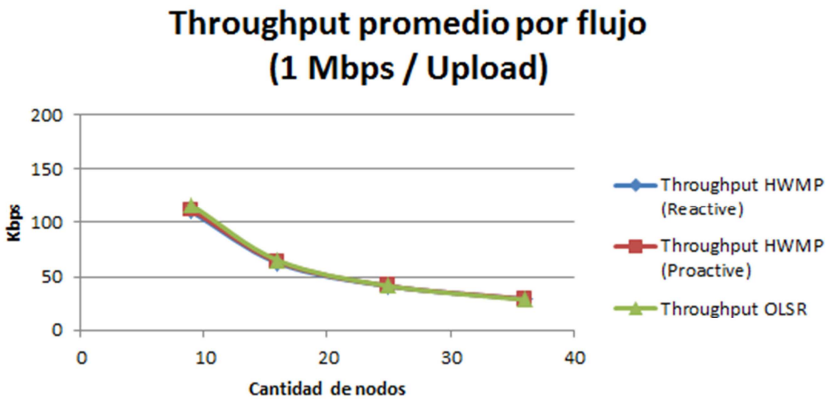

Ilustración 5-2 - Resultados del rendimiento para 1 Mbps con cargas.

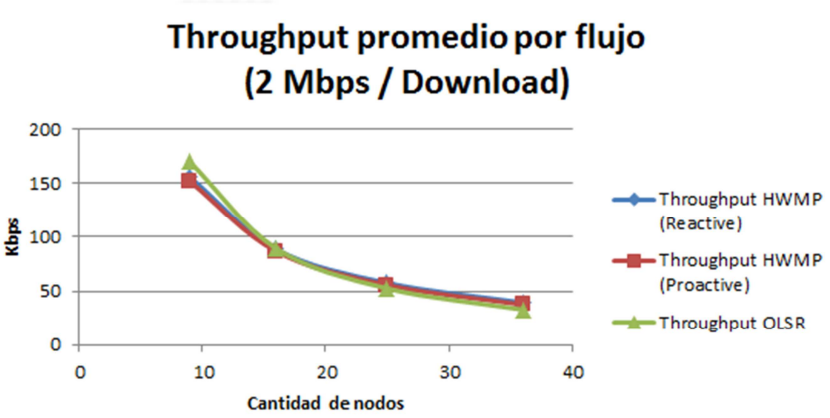

Ilustración 5-3 - Resultados del rendimiento para 2 Mbps con descargas.

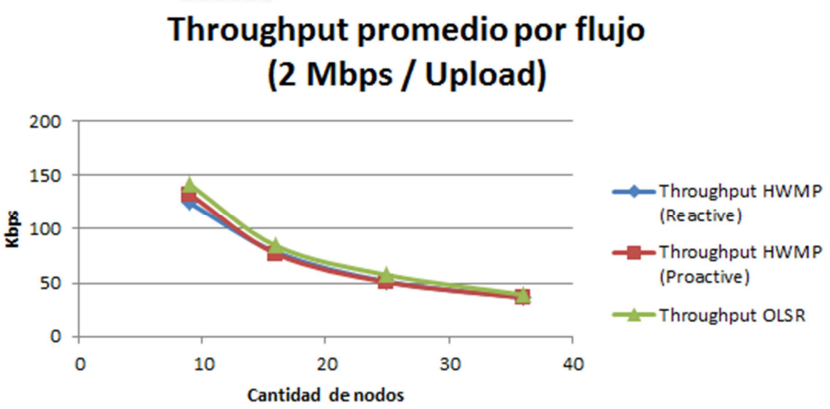

Ilustración 5-4 - Resultados del rendimiento para 2 Mbps con cargas.

Como era de esperar, el *throughput* decrece a medida que el número de nodos aumenta, independientemente del protocolo utilizado en la red mallada, aunque con el protocolo OLSR sin utilizar 802.11s el rendimiento es un poco mayor en ciertos casos.

La igualdad en el *throughput* también es un efecto generado por la restricción del ancho de banda en el enlace de salida, ya que independientemente de la velocidad dentro de la red inalámbrica siempre va a existir la restricción mayor al tratar de accesar el nodo Internet. El tráfico máximo hacia y desde Internet siempre va a estar limitado a este “embudo”.

5.1.2. Análisis de la pérdida de paquetes

En las ilustraciones 5-5, 5-6, 5-7 y 5-8 se observa la pérdida de paquetes promedio de los flujos de datos (de carga o descarga, según sea el caso del gráfico) en función del número de clientes mallados presentes en la red y el protocolo utilizado.

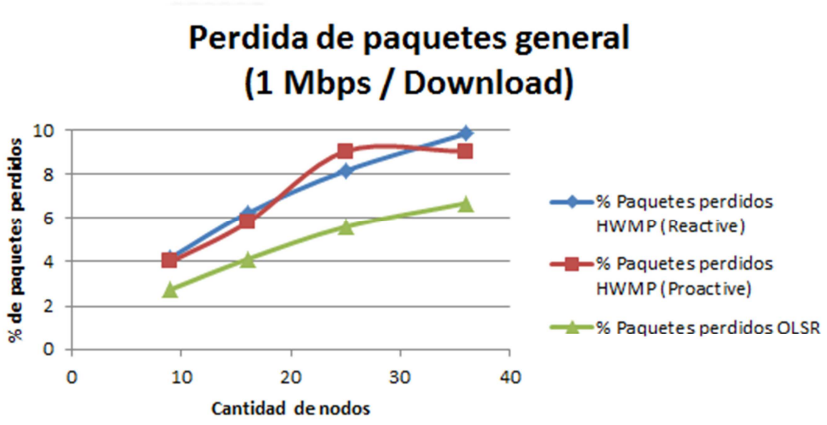

Ilustración 5-5 - Resultados de perdidas de paquetes para 1 Mbps con descargas.

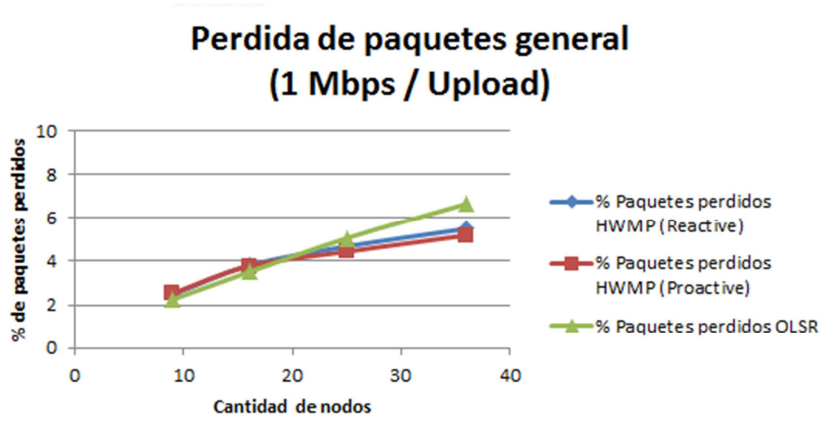

Ilustración 5-6 - Resultados de perdidas de paquetes para 1 Mbps con cargas.

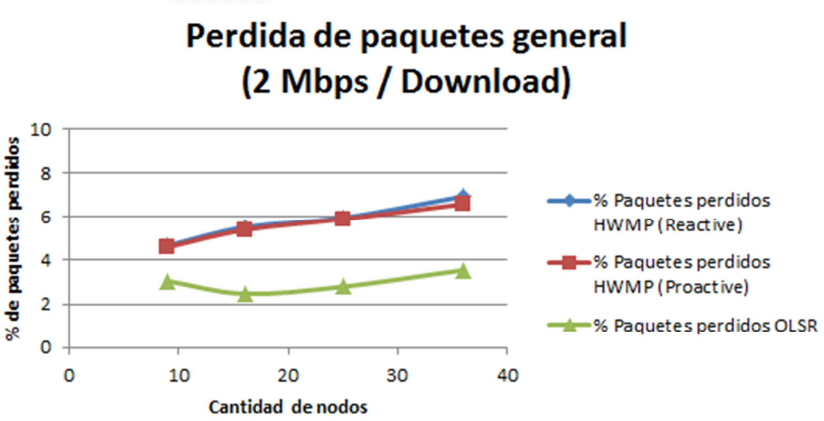

Ilustración 5-7 - Resultados de perdidas de paquetes para 2 Mbps con descargas.

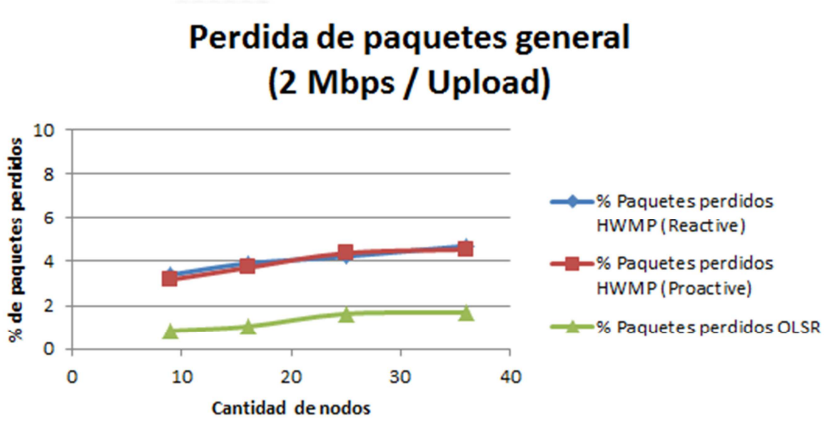

Ilustración 5-8 - Resultados de perdidas de paquetes para 2 Mbps con cargas.

El protocolo HWMP junto a 802.11s exhibió una mayor cantidad de paquetes perdidos en casi todos los casos, independientemente de la modalidad, velocidad del enlace a Internet, o dirección del flujo del tráfico. Esto demuestra que para este aspecto el uso de OLSR únicamente podría llegar a ser una ventaja. Sin embargo, si se toma en cuenta que existen métodos en capas superiores para reenviar la información perdida, este punto puede no afectar tanto al usuario final en usos donde la pérdida de paquetes y retransmisión no presente inconvenientes mayores, como por ejemplo navegación. Pero incluso está situación podría suponer un problema adicional, ya que al crear más tráfico por las retransmisiones se debería reducir el rendimiento general de la red, sin embargo este tipo de análisis queda fuera del alcance del presente documento.

5.1.3. Análisis del retraso punto a punto

En las ilustraciones 5-9, 5-10, 5-11 y 5-12 se observa el retraso punto a punto promedio de los flujos de datos (de carga o descarga, según sea el caso del gráfico) en función del número de clientes mallados presentes en la red y el protocolo utilizado.

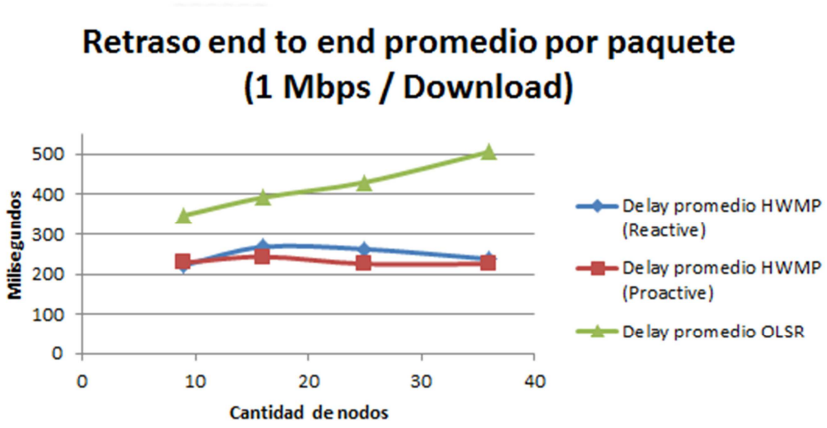

Ilustración 5-9 - Resultados del retraso para 1 Mbps con descargas.

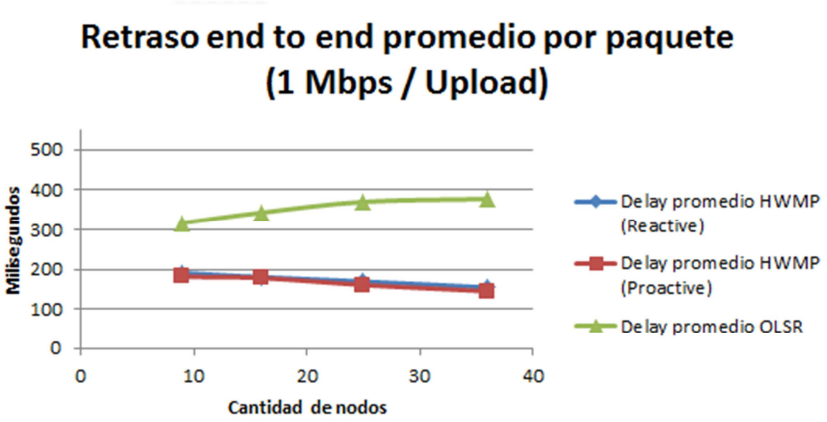

Ilustración 5-10 - Resultados del retraso para 1 Mbps con cargas.

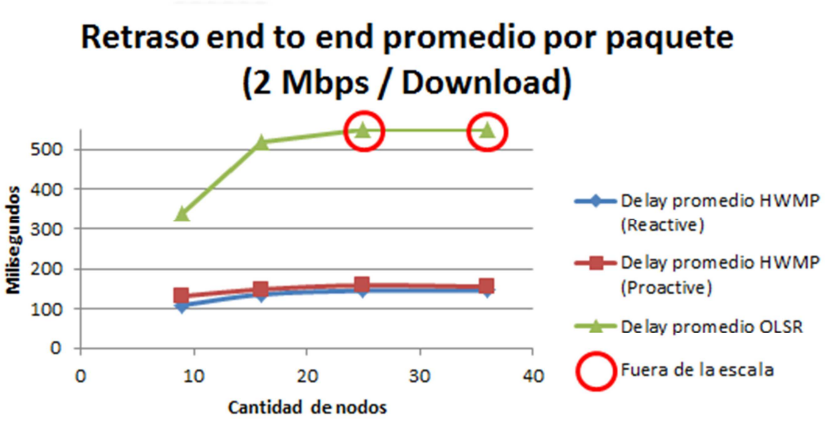

Ilustración 5-11 - Resultados del retraso para 2 Mbps con descargas.

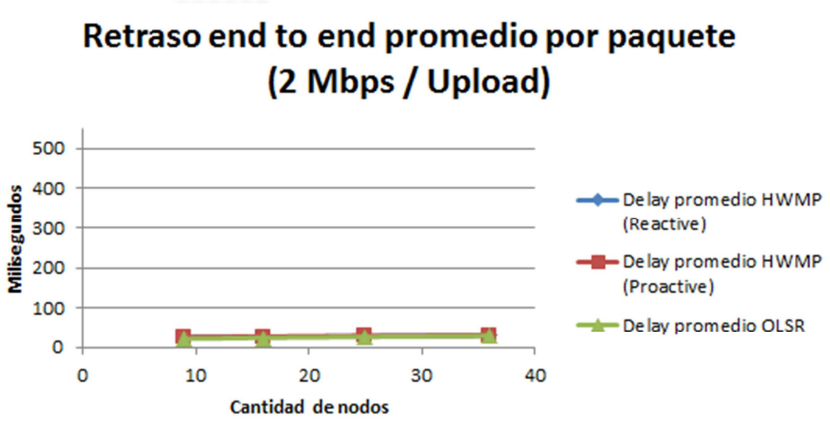

Ilustración 5-12 - Resultados del retraso para 2 Mbps con cargas.

En cuanto al retraso desde el origen al destino promedio, HWMP (muy similar en sus ambas modalidades) supera ampliamente a OLSR en casi todas las comparaciones realizadas, y en la única que es superada es por un pequeño margen. Se cree que es debido a la mejor métrica utilizada en HWMP, en la que se eligen enlaces más fiables y de calidad para establecer las rutas, mientras que OLSR solo toma el número de saltos, con lo que es posible que la ruta escogida se encuentre ya congestionada.

Un aspecto a destacar es lo estable del retraso promedio en HWMP en comparación a OLSR. Se puede puede observar la constancia en los valores entre los resultados del primero mientras varía el número de nodos, al contrario de OLSR que puede variar notablemente entre cada caso.

5.1.4. Analisis de la fluctuación (jitter)

En las ilustraciones 5-13, 5-14, 5-15 y 5-16 se observa la fluctuación o *jitter* promedio de los flujos de datos (de carga o descarga, según sea el caso del gráfico) en función del número de clientes mallados presentes en la red y el protocolo utilizado.

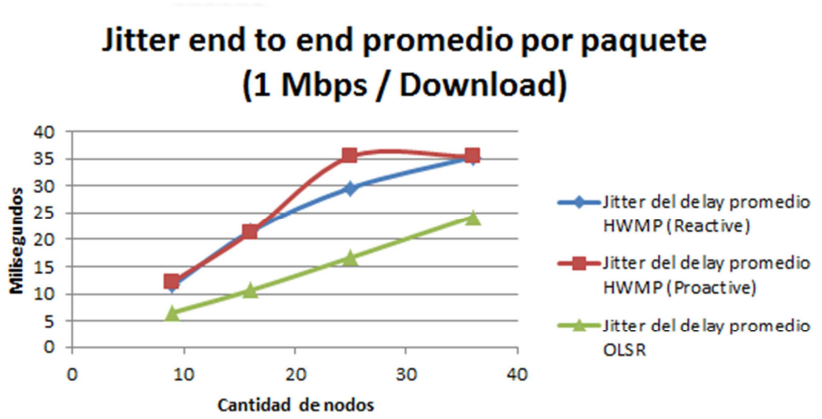

Ilustración 5-13 - Resultados del jitter para 1 Mbps con descargas.

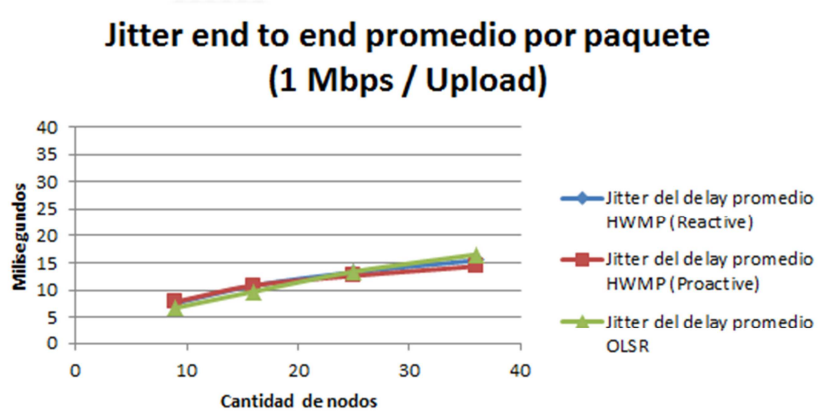

Ilustración 5-14 - Resultados del jitter para 1 Mbps con cargas.

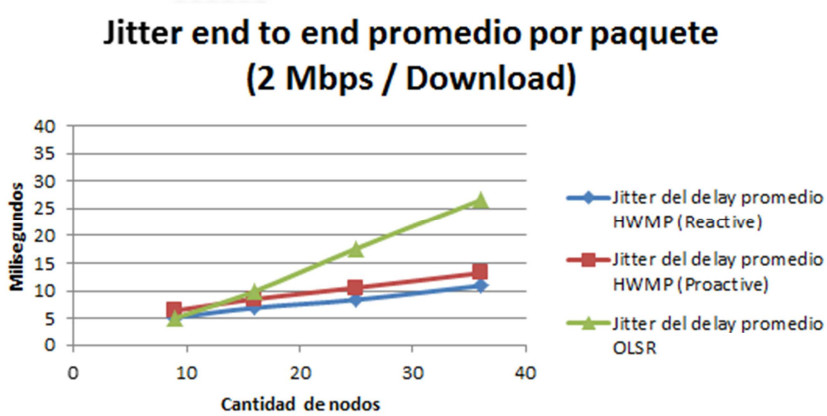

Ilustración 5-15 - Resultados del jitter para 2 Mbps con descargas.

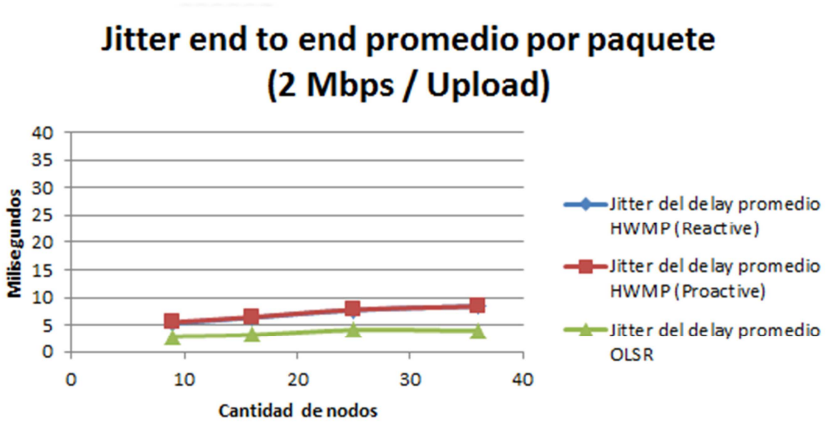

Ilustración 5-16 - Resultados del jitter para 2 Mbps con cargas.

Para el *jitter*, se tienen resultados que varían en todos los casos. En general la tendencia es que este aumenta entre más nodos se tienen en la red mallada. Cuando el flujo es de subida también es de notar que el *jitter* suele ser mucho menor que con tráfico de descarga cuando se tiene la misma velocidad en el enláce a Internet. Esto puede indicar que el *jitter* es afectado mayormente por otro(s) elemento(s) aparte del tipo de protocolo de enrutamiento.

5.1.5. Analisis del retraso inicial

En las ilustraciones 5-17, 5-18, 5-19 y 5-20 se observa el retraso al inico de una transmisión promedio de los flujos de datos (de carga o descarga, según sea el caso del gráfico) en función del número de clientes mallados presentes en la red y el protocolo utilizado.

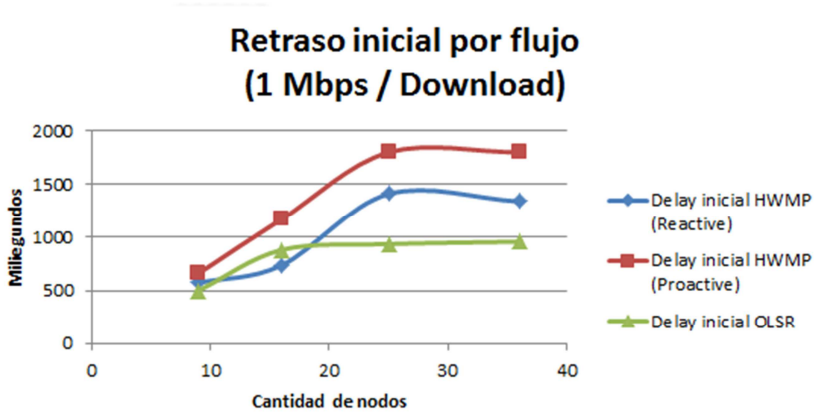

Ilustración 5-17 - Resultados del retraso inicial para 1 Mbps con descargas.

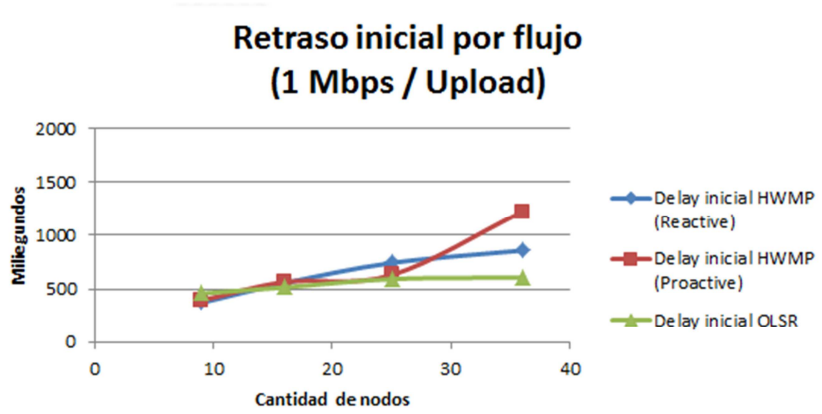

Ilustración 5-18 - Resultados del retraso inicial para 1 Mbps con cargas.

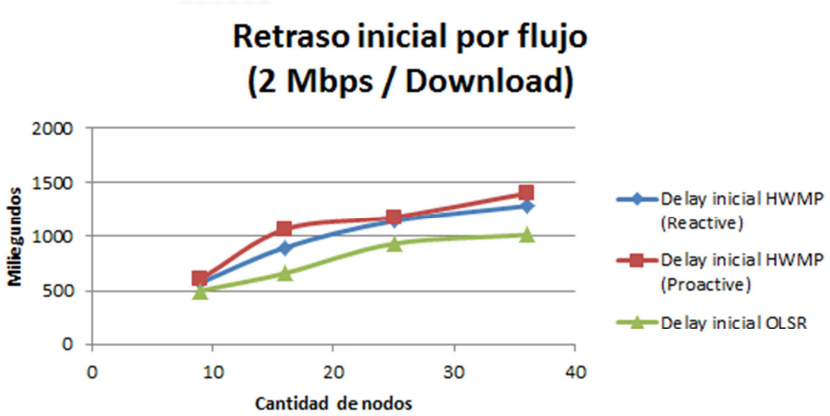

Ilustración 5-19 - Resultados del retraso inicial para 2 Mbps con descargas.

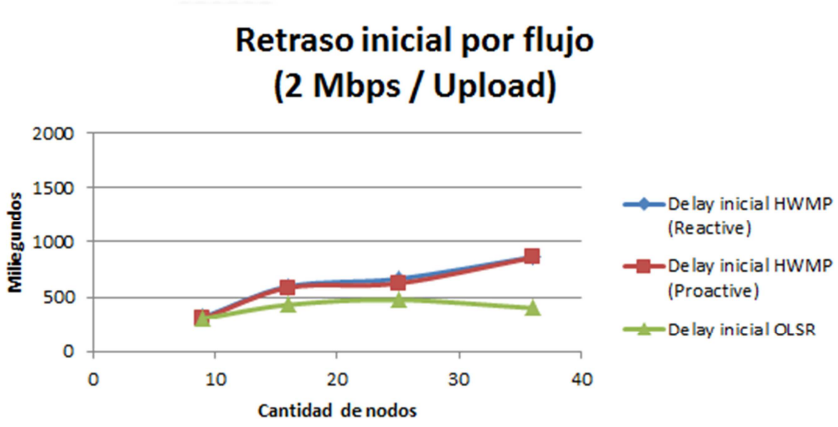

Ilustración 5-20 - Resultados del retraso inicial para 2 Mbps con cargas.

En el apartado del retraso inicial, vemos como OLSR supera a HWMP en general. La razón principal es que OLSR es un protocolo proactivo, teniendo todas las rutas conocidas previamente. Otra posible razón para esto es la sobrecarga al tráfico de red que coloca no solo HWMP, sino también 802.11s. HWMP necesita un tiempo para tomar las métricas, además de las funciones adicionales propias de 802.11s que no se realiza en el caso de OLSR. Por tanto, aunque los valores pueden arrojar una conclusión directa que HWMP es superado por OLSR, es importante tomar en cuenta que los tiempos adicionales en HWMP en los retrasos iniciales puede ser provocada por funciones extras adicionales que no se tienen el OLSR.

5.2. Resumen de los resultados y análisis final

La tabla 5-1 muestra en resumen los resultados finales de cada caso por ancho de banda y tipo de flujo, indicando que protocolo sobresalió en cada escenario. Es importante destacar que para los resultados donde un protocolo supera por un margen despreciable a otro, se concluye que tienen estos tienen el mismo rendimiento en ese escenario específico.

Protocolos con mejor rendimiento por escenario						
Ancho de banda	Tipo de flujo	Throughput	Pérdida de paquetes	Retraso end-to-end	Jitter	Retraso inicial
1 mbps	Carga	Igual	Igual (<20 nodos) HWMP (>20 nodos)	Ambos HWMP	OLSR (<20 nodos) HWMP (>20 nodos)	OLSR
	Descarga	Igual	OLSR	HWMP Proactivo	OLSR	OLSR
2 mbps	Carga	Ligeramente OLSR	OLSR	OLSR	OLSR	OLSR
	Descarga	Igual	OLSR	Ambos HWMP	HWMP Reactivo	OLSR

Tabla 5-1 - Protocolos con mejor rendimiento por escenario.

Se puede apreciar en términos generales que tanto para la pérdida de paquetes, *jitter* y retraso inicial OLSR aventaja a HWMP. En cuanto al retraso end-to-end, HWMP exhibe no solo un mejor comportamiento, sino también se muestra mucho más estable a través de los casos, demostrado en la constancia de los datos gráficados. Para el *throughput*, todos los protocolos presentaron un rendimiento muy similar, aunque OLSR aventajó por muy poco a HWMP (reactivo y proactivo) en sólo uno de los casos.

Teniendo que pasar obligatoriamente todo el tráfico de carga o descarga por el enláce de Internet, este se convierte en un “embudo” el cual limita el *throughput* completamente, independientemente de protocolo utilizado. Por esta razón se observa prácticamente el mismo rendimiento entre HWMP y OLSR en todos los escenarios.

Como era de esperarse, el *throughput* se reduce a medida que aumenta el número de nodos. Igualmente, el resto de los parámetros aumentan con el número nodos, con excepción de la estabilidad que HWMP demuestra en el retraso end-to-end.

Si bien OLSR supera a HWMP en un mejor retraso inicial, HWMP cuenta con un retraso end-to-end mucho menor en los flujos de datos. Se debe tomar en cuenta que mientras OLSR tiene un funcionamiento básico (no cuenta con 802.11s por ejemplo), HWMP debe negociar no solo el enrutamiento sino las funciones adicionales propias (por ejemplo, medir cada enlace para calcular la métrica) y de 802.11s (como asociación de nodos, seguridad, etc.). Este caso es un ejemplo que el costo en la espera inicial en HWMP trae beneficios durante el resto de la transmisión, al contrario de OLSR donde, aunque inicia más rápidamente por ser relativamente menos complejo, a largo plazo induce más retraso en la transmisión.

Como conclusión, se tiene un comportamiento general muy parecido entre ambos modos de HWMP y OLSR, cada uno con sus puntos fuertes. La sobrecarga que impone en la red los mecanismos adicionales de HWMP y 802.11s no son diferenciadores importantes para afirmar la superioridad de OLSR ante estos. Más bien, teniendo comportamientos tan cercanos en los protocolos, y sumadas las ventajas que se obtienen de utilizar HWMP en 802.11s, se puede concluir que esta es la mejor opción a elegir para el desarrollo de una solución como se plantea en el presente documento.

6. Conclusiones

En el presente trabajo de investigación se simuló y estudió el rendimiento de una red mallada inalámbrica con varios protocolos de enrutamiento, mientras que el acceso a Internet es restringido, y el tráfico es principalmente desde o hacia Internet.

Se tomaron protocolos de enrutamiento específicos para redes malladas que fueran soportados por el simulados NS-3. De estos, se seleccionaron los protocolos con correcto funcionamiento bajo el escenario requerido. Estos protocolos fueron HWMP en sus dos modalidades y OLSR.

Se definieron los casos a probar con cada uno de los protocolos, junto a los parámetros a establecer y métricas a tomar. Luego, se aplicó el escenario junto a los casos seleccionados para las pruebas en el ambiente simulado.

Una vez finalizadas las pruebas y analizados los resultados, es de notar la similitud de estos entre los protocolos seleccionados. Aunque los resultados son similares y cada protocolo presenta ventajas y desventajas, tomando en cuenta los beneficios que proporciona adicionalmente 802.11s con HWMP se recomienda el uso de estos para la implementación de una solución como la planteada en el presente Trabajo Especial de Grado.

6.1. Contribución

- Se realizó un análisis base para el nuevo estándar IEEE 802.11s en comparación de un protocolo de enrutamiento básico.
- Se comprobó la posibilidad de utilizar ambientes simulados para la realización de estudios en el campo de redes de telecomunicaciones.
- Se demostró la posibilidad de utilizar redes malladas inalámbricas para extender la zona de servicio de un punto de acceso que cuente con servicio de Internet.

6.2. Limitaciones

A continuación, algunas de las limitaciones que se presentaron durante el desarrollo del presente Trabajo Especial de Grado:

- Debido a limitaciones con el funcionamiento del simulador, un escenario adicional en el que se evaluaba la red mallada con dos puntos de acceso mallado a Internet independientes no se logró completar.

- Igualmente, por errores propios de NS3 [25][26], no se lograron implementar otros algoritmos de enrutamiento inalámbrico que ya existían dentro de Network Simulator 3. Aunque se desarrollaron los *scripts* necesarios para estos protocolos adicionales, los escenarios planteados no se ejecutaron debido a las limitaciones del simulador.

6.3. Trabajos futuros

En la actualidad, para obtener redes malladas inalámbricas estadarizadas y de buen rendiemiento aún hace falta camino que recorrer. Y aunque se ha logrado un largo avance en los últimos años en este tema, aún son necesarios estudios y practicas para poner a pruebas las distintas soluciones propuestas o definidas actualmente.

Dada la flexibilidad del estándar 802.11s, se propone realizar la comparación de HWMP con otros protocolos de enrutamiento mallado inalámbrico, pero con la variante de ser implementados *dentro* de 802.11s, como protocolo de enrutamiento alterno a HWMP. Así, se lograría ver exactamente que diferencias llevarían estos protocolos entre ellos usando el mismo terreno base de 802.11s.

Adicionalmente, también se propone tomar como base para el estudio otros aspectos aparte del protocolo de enrutamiento, ya que elementos como la capacidad de lidiar con enlaces caidos o de baja calidad, nodos que se desconectan de la red, etc., pueden resultar fundamentales en el funcionamiento de una red mallada bajo el escenario planteado.

Otra propuesta que surge del trabajo con el simulador NS3, es la implementación de nuevos protocolos de enrutamiento dentro del simulador. Este podría ser una evolución de otro protocolo ya desarrollado, o bien un protocolo totalmente nuevo, en el que se use NS3 como base para verificar su desarrollo y comportamiento.

7. Referencias

[1] A. Tanenbaum. Redes de computadoras (4ta. Edición). Pearson Educación. Mexico. 2003.

[2] M. Villapol. Introducción a las redes móviles e inalámbricas. Universidad Central de Venezuela. Septiembre 2010.

[3] Pei Zheng, Larry L. Peterson, Bruce S. Davie, Adrian Farrel. Wireless Networking Complete. Morgan Kauffman. Agosto 2009.

[4] Byeong Gi Lee, Sunghyun Choi. Broadband Wireless Access and Local Networks: Mobile WiMax and WiFi. Artech House. Enero 2008.

[5] http://www.ieeeghn.org/wiki/index.php/Wireless_LAN_802.11_Wi-Fi/ [En Línea].

[6] http://technet.microsoft.com/en-us/library/cc757419(v=ws.10).aspx [En Línea]

[7] http://airmagnet.flukenetworks.com/assets/whitepaper/WP-802.11nPrimer.pdf [En Línea]

[8] http://www.ieee802.org/11/Reports/802.11_Timelines.htm [En Línea].

[9] Rich Watson. White Paper - Understanding the IEEE 802.11ac Wi-Fi Standard. Meru Networks. Septiembre 2012.

[10] I. Akyildiz. A Survey on Wireless Mesh Networks. Georgia Institute Of Technology.

[11] W. Stallings. Comunicación y Redes de Computadoras (6ta. Edición). Septiembre 2011.

[12] T. Clausen, Ed.P. Jacquet, Ed. Optimized Link State Routing Protocol (OLSR). RFC 3626. Octubre 2003.

[13] C. Perkins, E. Belding-Royer, S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561. Julio 2003.

[14] A. Joshi, H. Gossain, J. Jetcheva, M. Audeh, M. Bahr, J. Kruys, A. Lim, S. Rahman, J. Kim, S. Conner, G. Strutt, H. Liu, S. Hares. IEEE P802.11 Wireless LANs. Noviembre 2006.

[15] Joseph D. Camp, Edward W. Knightly. The IEEE 802.11s Extended Service Set Mesh Networking Standard. Electrical and Computer Engineering, Rice University, Houston, TX. Abril 2007.

[16] http://standards.ieee.org/develop/wg/WG802.11.html/ [En Línea].

[17] Calsoft Labs. Whitepaper - 802.11s Wireless Mesh Solution. 2012.

[18] http://www.nsnam.org/ [En Línea].

[19] http://www.nsnam.org/docs/release/3.19/tutorial/html/index.html/ [En Línea].

[20] http://www.cantv.com.ve/seccion.asp?pid=1&sid=607 [En Línea].

[21] http://www.inter.com.ve/residenciales/internet/index.php [En Línea].

[22] S. Bradner. Benchmarking Terminology for Network Interconnection Devices. RFC 1242. Julio 1991.

[23] Comer, Douglas E. Computer Networks and Internets. Prentice Hall. 2008.

[24] Carolina Balderrama, Mariana Colombo. Estudio de los Protocolos de Enrutamiento para Redes Malladas con Restricciones de Ancho de Banda en el Borde. Universidad Central de Venezuela. 2010.

[25] https://groups.google.com/forum/#!topic/ns-3-users/7Iv2mZBcBTU [En Línea].

[26] https://www.nsnam.org/bugzilla/show_bug.cgi?id=1911 [En Línea].

ANEXO A

- *Barrio Bello Campo:*

Distáncia mínima promedio medida entre edificios y barriada: 50 metros.

- *Barrio Brisas del Paraíso:*

Distáncia mínima promedio medida entre edificios y barriada: 70 metros.

- *Barrio Chapellín:*

Distáncia mínima promedio entre edificios y barriada: 70 metros.

- *Barrio La Coromoto:*

Distáncia mínima promedio entre edificios y barriada: 60 metros.

- *Barrio La Cruz:*

Distáncia mínima promedio entre edificios y barriada: 36 metros.

- *Barrio León Droz Blanco:*

Distáncia mínima promedio entre edificios y barriada: 46 metros.

ANEXO B

- *Contenido del archivo basev11-hwmp.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
 *
 *
 * Net topology
 *
 * With a 3x3 mesh grid (size can vary):
 *
 *             n2   n3   n4
 *
 *
 * n0----n1 )))  n5   n6   n7
 *
 *
 *             n8   n9   n10
 *
 *
 *
 *  n0: internet node
 *  n1: access node (a static mesh node with a P2P link).
 *  n2-n10: mesh nodes
 *
 *  n0 has ip 1.1.1.1 to emulate internet access.
 *  n1 is always at the middle of the mesh, horizontal distance to the mesh can be set.
 *
 *  There are traffic generators in each node but n1 (no need)
 *  so they can emulate uploads and downloads to internet.
 *
 *  Two IPv4 ping app is enable to check connectivity and keep track of
 *  simulation as it runs. They ping "internet" (AKA 1.1.1.1) from and to the last node every 1 sec.
 *
 *  In this scenario, static routes are used as L3 routing.
 *  Every node has n1 as default route (gateway)
 *  Meanwhile, 802.11s and HWMP do the routing in L2 inside the mesh
 *  If proactive is enable, root will be n1
 *
 *  Performance is measured with Flowmon. Every node (but n1) has sink for packets.
 *  Flows are output in a .CSV file, using tabs as separators
 *
 *  Author: Carlos Cordero
 *  Email: carlos.cordero.0@gmail.com
 *
```

```
 */

#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
#include "ns3/v4ping-helper.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/ipv4-flow-classifier.h"
#include "ns3/flow-monitor.h"
#include "ns3/mesh-helper.h"
#include <stdlib.h>

using namespace ns3;

int main (int argc, char *argv[])
{
  NS_LOG_COMPONENT_DEFINE ("TesisBase");
  LogComponentEnable ("V4Ping", LOG_LEVEL_DEBUG);

// Variable definition (all generic variables starts with "m_")
  int m_xNodes = 3; // Mesh width in number of nodes
  int m_yNodes = 3; // Mesh heigth in number of nodes
  int m_distNodes = 35; // Distance between nodes (mts)
  int m_distAP = 50; // Distance between access nodes and mesh (mts)
  int m_totalTime = 120; // Time to simulate (segs) (time >= 35)
  std::string m_stack = "ns3::Dot11sStack"; // Mesh stack
  bool m_root = false; // Set true if proactive mode is enable
  int m_packetSize = 256; // Packet size for tests
  bool m_drawAnim = false; // Enable netanim .xls output
  bool m_newFlowFile = false; // Clear flow .csv file
  std::string m_flow = "both"; // Set traffic direction: download, upload, both
  std::string m_txAppRate = "128kbps"; // Traffic generation speed for apps
  std::string m_txInternetRate = "1Mbps"; // Transmision speed for n0 <-> n1 link
  std::string m_animFile = "resultados/basev11-hwmp-anim.xml"; // File for .xml
  std::string m_routeFile = "resultados/basev11-hwmp-route.xml"; // File for .xml routing
  std::string m_statsFile = "resultados/basev11-hwmp-3x3"; // Prefix for statistics output files
  std::string m_flowmonFile = "resultados/basev11-hwmp-flow.flowmon"; // File for flowmon output
  int tmp_x;
  char tmp_char [30] = "";

  srand (time (NULL)); // Ensure random results in every run
  tmp_x = rand () % 1000;
  printf ("Initial random seed: %d\n", tmp_x);
  RngSeedManager::SetSeed (tmp_x);

// Command line options
  CommandLine cmd;
  cmd.AddValue ("flow-direction", "Directions of traffic (download / upload / both)", m_flow);
  cmd.AddValue ("mesh-width", "Number of node in mesh width", m_xNodes);
  cmd.AddValue ("mesh-height", "Number of node in mesh height", m_yNodes);
  cmd.AddValue ("node-distance", "Distance between nodes (horizontally / vertically)", m_distNodes);
  cmd.AddValue ("ap-distance", "Distance between access nodes and mesh (horizontally)", m_distAP);
  cmd.AddValue ("time", "Total time to simulate", m_totalTime);
  cmd.AddValue ("app-packet-size", "Set packet size to tx from apps", m_packetSize);
  cmd.AddValue ("app-tx-rate", "Set speed of traffic generation", m_txAppRate);
  cmd.AddValue ("link-speed", "Transmision speed over P2P link", m_txInternetRate);
```

```
  cmd.AddValue ("enable-anim", "Enable output for .xml animation", m_drawAnim);
  cmd.AddValue ("anim-file", "Set output name for .xml animation file", m_animFile);
  cmd.AddValue ("route-file", "Set output name for route file", m_routeFile);
  cmd.AddValue ("stats-file", "Set output prefix for .csv flows results file", m_statsFile);
  cmd.AddValue ("new-flow-file", "Clear .csv flows results file", m_newFlowFile);
  cmd.AddValue ("flow-file", "Set output name for flow monitor .flowmon file", m_flowmonFile);
  cmd.AddValue ("root", "Set if root is enable in HWMP", m_root);
  cmd.Parse (argc, argv);

 // Node container creation (all node containers starts with "nc_")
  NodeContainer nc_all; // Contains every node (starting with internet node, access nodes and then mesh nodes
  NodeContainer nc_mesh; // Contains only mesh nodes (x*y nodes)
  NodeContainer nc_wireless; // Contains access and mesh nodes
  NodeContainer nc_internet; // Contains internet nodes
  NodeContainer nc_destiny; // Contains nodes with sink for apps (internet and mesh nodes)
  nc_mesh.Create (m_xNodes * m_yNodes);
  nc_all.Create (2);
  nc_all.Add (nc_mesh);
  nc_wireless.Add (nc_all.Get (1));
  nc_wireless.Add (nc_mesh);
  nc_internet.Add (nc_all.Get (0));
  nc_internet.Add (nc_all.Get (1));
  nc_destiny.Add (nc_all.Get (0));
  nc_destiny.Add (nc_mesh);

 // Create P2P links between n0 <-> n1 (all devices starts with "de_")
  PointToPointHelper p2pinternet;
  p2pinternet.SetDeviceAttribute ("DataRate", StringValue (m_txInternetRate));
  p2pinternet.SetChannelAttribute ("Delay", StringValue ("1ms"));
  NetDeviceContainer de_internet;
  de_internet = p2pinternet.Install (nc_internet);

 // Set channel, phy and mac layers
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
  wifiChannel.AddPropagationLoss ("ns3::RangePropagationLossModel","MaxRange", DoubleValue (65));
  wifiPhy.SetChannel (wifiChannel.Create ());
  MeshHelper mesh;
  mesh = MeshHelper::Default ();
  mesh.SetStandard (WIFI_PHY_STANDARD_80211g);
  mesh.SetMacType ("RandomStart", TimeValue (Seconds(0.1)));
  mesh.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode", StringValue ("ErpOfdmRate6Mbps"),
                                "ControlMode",StringValue ("ErpOfdmRate6Mbps")); //Valid ErpOfdmRate: 6 9 12 18 24 36 48 54
  if (m_root)
  { // If proactive mode is enable, access node will be root 00:00:00:00:00:03
    printf ("Proactive mode set!\n");
    mesh.SetStackInstaller (m_stack, "Root", Mac48AddressValue (Mac48Address ("00:00:00:00:00:03")));
  }
  else
  {
    printf ("Reactive mode set!\n");
    mesh.SetStackInstaller (m_stack);
  }
  mesh.SetSpreadInterfaceChannels (MeshHelper::ZERO_CHANNEL);
  mesh.SetMacType ("RandomStart", TimeValue (Seconds (2)));
  mesh.SetNumberOfInterfaces (1);
  NetDeviceContainer de_wireless = mesh.Install (wifiPhy, nc_wireless);

 // Set startup positions for nodes
  MobilityHelper mobilityMesh; //Position mesh nodes
```

```
  mobilityMesh.SetPositionAllocator ("ns3::GridPositionAllocator",
                                     "MinX", DoubleValue (m_distAP),
                                     "MinY", DoubleValue (0.0),
                                     "DeltaX", DoubleValue (m_distNodes),
                                     "DeltaY", DoubleValue (m_distNodes),
                                     "GridWidth", UintegerValue (m_xNodes),
                                     "LayoutType", StringValue ("RowFirst"));
  mobilityMesh.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobilityMesh.Install (nc_mesh);
  MobilityHelper mobilityInternet; //Position internet and access node
  tmp_x = m_yNodes % 2;
  if (tmp_x == 1) // This centers the access node to the middle of the mesh, in line with a mesh node
  { // with impair number of nodes over the Y axis, node is at the center
    mobilityInternet.SetPositionAllocator ("ns3::GridPositionAllocator",
                                     "MinX", DoubleValue (m_distAP * -1),
                                     "MinY", DoubleValue ((m_distNodes * (m_yNodes - 1)) / 2),
                                     "DeltaX", DoubleValue (m_distAP),
                                     "DeltaY", DoubleValue (0.0),
                                     "GridWidth", UintegerValue (2),
                                     "LayoutType", StringValue ("RowFirst"));
  }
  else
  { // with pair nomber of nodes over the X axis, node is align with the node before the center of the mesh
     tmp_x = m_yNodes - 1;
     mobilityInternet.SetPositionAllocator ("ns3::GridPositionAllocator",
                                     "MinX", DoubleValue (m_distAP * -1),
                                     "MinY", DoubleValue ((m_distNodes * (tmp_x - 1)) / 2),
                                     "DeltaX", DoubleValue (m_distAP),
                                     "DeltaY", DoubleValue (0.0),
                                     "GridWidth", UintegerValue (2),
                                     "LayoutType", StringValue ("RowFirst"));
  }
  mobilityInternet.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobilityInternet.Install (nc_internet);

 // Set Internet stack
  InternetStackHelper internetStack;
  internetStack.Install (nc_all);

 // Config IPs for interfaces (all interfaces starts with "if_")
  Ipv4AddressHelper addrMesh;
  addrMesh.SetBase ("10.0.0.0", "255.255.255.0");
  Ipv4InterfaceContainer if_mesh = addrMesh.Assign (de_wireless);

  Ipv4AddressHelper addrInternet;
  addrInternet.SetBase ("1.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer if_internet = addrInternet.Assign (de_internet);

 // Set default L3 routes
  Ipv4StaticRoutingHelper ipv4RoutingHelper;
  Ptr<Ipv4> ipv4;
  Ptr<Ipv4StaticRouting> staticRouting;
  // Default gateway for node 0 (send all to node 1)
  ipv4 = nc_all.Get (0)->GetObject<Ipv4> ();
  staticRouting = ipv4RoutingHelper.GetStaticRouting (ipv4);
  staticRouting->SetDefaultRoute (Ipv4Address ("1.1.1.2"), 1, 1);

  // Default gateway for mesh nodes (send all to access point)
  for (tmp_x = 2; tmp_x < m_xNodes * m_yNodes + 2; tmp_x++)
  {
    ipv4 = nc_all.Get (tmp_x)->GetObject<Ipv4> ();
```

```
    staticRouting = ipv4RoutingHelper.GetStaticRouting (ipv4);
    staticRouting->SetDefaultRoute (Ipv4Address ("10.0.0.1"), 1, 1);
  }

// Install applications
// Set sinks for receiving data
  PacketSinkHelper sinkTcp ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), 9));
  ApplicationContainer ac_sinkTcp = sinkTcp.Install (nc_destiny);
  ac_sinkTcp.Start (Seconds (0));
  ac_sinkTcp.Stop (Seconds (m_totalTime));

// Set traffic generator apps
  Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue (m_packetSize));
  Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue (m_txAppRate));
  Config::SetDefault ("ns3::OnOffApplication::OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=10.0]"));
  Config::SetDefault ("ns3::OnOffApplication::OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));

  if (m_flow == "upload" || m_flow == "both")
  {
    ApplicationContainer ac_onoffMesh [m_xNodes * m_yNodes]; // Creates 1 OnOff App in each mesh node TO internet
    OnOffHelper onoffMesh ("ns3::TcpSocketFactory", Address (InetSocketAddress (if_internet.GetAddress (0), 9)));
    for (tmp_x = 0; tmp_x < m_xNodes * m_yNodes; tmp_x++)
    {
      ac_onoffMesh [tmp_x] = onoffMesh.Install (nc_mesh.Get (tmp_x));
      ac_onoffMesh [tmp_x].Start (Seconds (15 + rand () % 10));
      ac_onoffMesh [tmp_x].Stop (Seconds (m_totalTime - 1 - rand () % 10));
    }
  }

  if (m_flow == "download" || m_flow == "both")
  {
    ApplicationContainer ac_onoffInternet [m_xNodes * m_yNodes]; // Creates 1 OnOff App for each mesh node FROM
internet
    for (tmp_x = 0; tmp_x < m_xNodes * m_yNodes; tmp_x++)
    {
      OnOffHelper onoffInternet ("ns3::TcpSocketFactory", Address (InetSocketAddress (if_mesh.GetAddress (tmp_x + 1),
9)));
      ac_onoffInternet [tmp_x] = onoffInternet.Install (nc_all.Get (0));
      ac_onoffInternet [tmp_x].Start (Seconds (15 + rand () % 10));
      ac_onoffInternet [tmp_x].Stop (Seconds (m_totalTime - 1 - rand () % 10));
    }
  }

/////PING FOR TESTS
  V4PingHelper ping1 (if_internet.GetAddress (0));
  ping1.SetAttribute ("Verbose", BooleanValue (true));
  ping1.SetAttribute ("Interval", TimeValue (Seconds (1)));
  ApplicationContainer appPingInternet1 = ping1.Install (nc_all.Get (m_xNodes * m_yNodes + 1));
  appPingInternet1.Start (Seconds (0));
  appPingInternet1.Stop (Seconds (m_totalTime - 1));

  V4PingHelper ping2 (if_mesh.GetAddress (m_xNodes * m_yNodes));
  ping2.SetAttribute ("Verbose", BooleanValue (true));
  ping2.SetAttribute ("Interval", TimeValue (Seconds (1)));
  ApplicationContainer appPingInternet2 = ping2.Install (nc_all.Get (0));
  appPingInternet2.Start (Seconds (0));
  appPingInternet2.Stop (Seconds (m_totalTime - 1));
/////END PING FOR TESTS


// Set simulation time
```

```
  Simulator::Stop (Seconds (m_totalTime));

  for (tmp_x = 2; tmp_x < m_xNodes * m_yNodes + 2; tmp_x++)
  {
    sprintf (tmp_char, "%d-STA", tmp_x);
    AnimationInterface::SetNodeDescription (nc_all.Get (tmp_x), tmp_char);
  }
  AnimationInterface::SetNodeDescription (nc_all.Get (0), "0-Internet");
  AnimationInterface::SetNodeDescription (nc_all.Get (1), "1-AP");
  AnimationInterface::SetNodeColor (nc_mesh, 0, 255, 0);
  AnimationInterface::SetNodeColor (nc_internet.Get (0), 255, 0, 0);
  AnimationInterface::SetNodeColor (nc_internet.Get (1), 0, 0, 255);
  if (m_drawAnim)
  {
    AnimationInterface anim (m_animFile);
    anim.EnablePacketMetadata(true);
    anim.EnableIpv4RouteTracking (m_routeFile, Seconds (0), Seconds (m_totalTime), Seconds (0.25));
  }

 // Flow Monitor
  FlowMonitorHelper flowmon;
  Ptr<FlowMonitor> monitor = flowmon.Install (nc_destiny);
  monitor->Start (0);
  monitor->Stop (m_totalTime);
  if (m_drawAnim)
  {
                monitor->SerializeToXmlFile (m_flowmonFile, true, true);
  }

 // Run the simulation
  Simulator::Run ();

 /////////// Log data
 // If requested, clean the flow file for a new use
  if (m_newFlowFile)
  {
    std::ostringstream os_clear;
    os_clear << m_statsFile << "_flows.csv";
    std::ofstream of_clear (os_clear.str().c_str(), std::ios::out | std::ios::trunc);
    of_clear << """Src IP""\t""Dest IP""\t";
    of_clear << """Protocol""\t""SrcPort-DestPort""\t";
    of_clear << """First Tx Pkt""\t""Last Tx Pkt""\t";
    of_clear << """First Rx Pkt""\t""Last Rx Pkt""\t";
    of_clear << """Total Tx Bytes""\t""Total Rx Bytes""\t";
    of_clear << """Total Tx Packets""\t""Total Rx Packets""\t";
    of_clear << """Total Delay""\t""Total Jitter""\n";
    of_clear.close ();
  }

 // Open file to append new data
  std::ostringstream os;
  os << m_statsFile << "_flows.csv";
  std::ofstream of (os.str().c_str(), std::ios::out | std::ios::app);

 //Print per flow statistics
  monitor->CheckForLostPackets ();
  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)
  {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
```

```
    of << t.sourceAddress << "\t" << t.destinationAddress << "\t";
    if (t.protocol == 6)
    {
      of << "TCP\t";
    }
    else
    {
      if (t.protocol == 17)
      {
        of << "UDP\t";
      }
      else
      {
        of << "N/A\t";
      }
    }
    of << t.sourcePort << "-" << t.destinationPort << "\t";
    of << i->second.timeFirstTxPacket.GetSeconds() << "\t" << i->second.timeLastTxPacket.GetSeconds() << "\t";
    of << i->second.timeFirstRxPacket.GetSeconds() << "\t" << i->second.timeLastRxPacket.GetSeconds() << "\t";
    of << i->second.txBytes << "\t" << i->second.rxBytes << "\t";
    of << i->second.txPackets << "\t" << i->second.rxPackets << "\t";
    of << i->second.delaySum << "\t" << i->second.jitterSum << "\n";
  }
  of.close ();
////////////// End Log data

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

ANEXO C

- *Contenido del archivo basev11-olsr.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
 *
 *
 * Net topology
 *
 * With a 3x3 mesh grid (size can vary):
 *
 *             n2   n3   n4
 *
 *
 * n0----n1 )))  n5   n6   n7
 *
 *
 *             n8   n9   n10
 *
 *
 *
 *  n0: internet node
 *  n1: access node (a static mesh node with a P2P link).
 *  n2-n10: mesh nodes
 *
 *  n0 has ip 1.1.1.1 to emulate internet access.
 *  n1 is always at the middle of the mesh, horizontal distance to the mesh can be set.
 *
 *  There are traffic generators in each node but n1 (no need)
 *  so they can emulate uploads and downloads to internet.
 *
 *  Two IPv4 ping app is enable to check connectivity and keep track of
 *  simulation as it runs. They ping "internet" (AKA 1.1.1.1) from and to the last node every 1 sec.
 *
 *  In this scenario, OLSR routes are used as L3 routing protocol for everyone.
 *  Knowing next hop neighbour makes posible to solve L2 routing.
 *
 *  Performance is measured with Flowmon. Every node (but n1) has sink for packets.
 *  Flows are output in a .CSV file, using tabs as separators
 *
 *  Author: Carlos Cordero
 *  Email: carlos.cordero.0@gmail.com
 *
 */
```

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
#include "ns3/v4ping-helper.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/ipv4-flow-classifier.h"
#include "ns3/flow-monitor.h"
#include "ns3/olsr-helper.h"
#include <stdlib.h>

using namespace ns3;

int main (int argc, char *argv[])
{
  ns3::PacketMetadata::Enable ();
  NS_LOG_COMPONENT_DEFINE ("TesisBase");
  LogComponentEnable ("V4Ping", LOG_LEVEL_DEBUG);

// Variable definition (all generic variables starts with "m_")
  int m_xNodes = 3; // Mesh width in number of nodes
  int m_yNodes = 3; // Mesh heigth in number of nodes
  int m_distNodes = 35; // Distance between nodes (mts)
  int m_distAP = 50; // Distance between access nodes and mesh (mts)
  int m_totalTime = 120; // Time to simulate (segs) (time >= 35)
  int m_packetSize = 256; // Packet size for tests
  bool m_drawAnim = false; // Enable netanim .xls output
  bool m_newFlowFile = false; // Clear flow .csv file
  std::string m_flow = "both"; // Set traffic direction: download, upload, both
  std::string m_txAppRate = "128kbps"; // Traffic generation speed for apps
  std::string m_txInternetRate = "1Mbps"; // Transmision speed for n0 <-> n1 link
  std::string m_animFile = "resultados/basev11-olsr-anim.xml"; // File for .xml
  std::string m_routeFile = "resultados/basev11-olsr-route.xml"; // File for .xml routing
  std::string m_statsFile = "resultados/basev11-olsr-3x3"; // Prefix for statistics output files
  std::string m_flowmonFile = "resultados/basev11-olsr-flow.flowmon"; // File for flowmon output
  int tmp_x;
  char tmp_char [30] = "";

  srand (time (NULL)); // Ensure random results in every run
  tmp_x = rand () % 1000;
  printf ("Initial random seed: %d\n", tmp_x);
  RngSeedManager::SetSeed (tmp_x);

// Command line options
  CommandLine cmd;
  cmd.AddValue ("flow-direction", "Directions of traffic (download / upload / both)", m_flow);
  cmd.AddValue ("mesh-width", "Number of node in mesh width", m_xNodes);
  cmd.AddValue ("mesh-height", "Number of node in mesh height", m_yNodes);
  cmd.AddValue ("node-distance", "Distance between nodes (horizontally / vertically)", m_distNodes);
  cmd.AddValue ("ap-distance", "Distance between access nodes and mesh (horizontally)", m_distAP);
  cmd.AddValue ("time", "Total time to simulate", m_totalTime);
  cmd.AddValue ("app-packet-size", "Set packet size to tx from apps", m_packetSize);
  cmd.AddValue ("app-tx-rate", "Set speed of traffic generation", m_txAppRate);
  cmd.AddValue ("link-speed", "Transmision speed over P2P link", m_txInternetRate);
  cmd.AddValue ("enable-anim", "Enable output for .xml animation", m_drawAnim);
  cmd.AddValue ("anim-file", "Set output name for .xml animation file", m_animFile);
  cmd.AddValue ("route-file", "Set output name for route file", m_routeFile);
```

```
  cmd.AddValue ("stats-file", "Set output prefix for .csv flows results file", m_statsFile);
  cmd.AddValue ("new-flow-file", "Clear .csv flows results file", m_newFlowFile);
  cmd.AddValue ("flow-file", "Set output name for flow monitor .flowmon file", m_flowmonFile);
  cmd.Parse (argc, argv);

 // Node container creation (all node containers starts with "nc_")
  NodeContainer nc_all; // Contains every node (starting with internet node, access nodes and then mesh nodes
  NodeContainer nc_mesh; // Contains only mesh nodes (x*y nodes)
  NodeContainer nc_wireless; // Contains access and mesh nodes
  NodeContainer nc_internet; // Contains internet nodes
  NodeContainer nc_destiny; // Contains nodes with sink for apps (internet and mesh nodes)
  nc_mesh.Create (m_xNodes * m_yNodes);
  nc_all.Create (2);
  nc_all.Add (nc_mesh);
  nc_wireless.Add (nc_all.Get (1));
  nc_wireless.Add (nc_mesh);
  nc_internet.Add (nc_all.Get (0));
  nc_internet.Add (nc_all.Get (1));
  nc_destiny.Add (nc_all.Get (0));
  nc_destiny.Add (nc_mesh);

 // Create P2P links between n0 <-> n1 (all devices starts with "de_")
  PointToPointHelper p2pinternet;
  p2pinternet.SetDeviceAttribute ("DataRate", StringValue (m_txInternetRate));
  p2pinternet.SetChannelAttribute ("Delay", StringValue ("1ms"));
  NetDeviceContainer de_internet;
  de_internet = p2pinternet.Install (nc_internet);

 // Set channel, phy and mac layers
  WifiHelper wifi;
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
  NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
  wifi.SetStandard (WIFI_PHY_STANDARD_80211g);
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue ("ErpOfdmRate6Mbps"),
                                "ControlMode",StringValue ("ErpOfdmRate6Mbps")); //Valid ErpOfdmRate: 6 9 12 18 24 36 48 54
  wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

  wifiChannel.AddPropagationLoss ("ns3::RangePropagationLossModel","MaxRange", DoubleValue (65));
  wifiMac.SetType ("ns3::AdhocWifiMac");
  wifiPhy.SetChannel (wifiChannel.Create ());
  NetDeviceContainer de_wireless = wifi.Install (wifiPhy, wifiMac, nc_wireless);

 // Set startup positions for nodes
  MobilityHelper mobilityMesh; //Position mesh nodes
  mobilityMesh.SetPositionAllocator ("ns3::GridPositionAllocator",
                                     "MinX", DoubleValue (m_distAP),
                                     "MinY", DoubleValue (0.0),
                                     "DeltaX", DoubleValue (m_distNodes),
                                     "DeltaY", DoubleValue (m_distNodes),
                                     "GridWidth", UintegerValue (m_xNodes),
                                     "LayoutType", StringValue ("RowFirst"));
  mobilityMesh.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobilityMesh.Install (nc_mesh);
  MobilityHelper mobilityInternet; //Position internet and access node
  tmp_x = m_yNodes % 2;
  if (tmp_x == 1) // This centers the access node to the middle of the mesh, in line with a mesh node
  { // with impair number of nodes over the Y axis, node is at the center
    mobilityInternet.SetPositionAllocator ("ns3::GridPositionAllocator",
                                     "MinX", DoubleValue (m_distAP * -1),
```

```
                                "MinY", DoubleValue ((m_distNodes * (m_yNodes - 1)) / 2),
                                "DeltaX", DoubleValue (m_distAP),
                                "DeltaY", DoubleValue (0.0),
                                "GridWidth", UintegerValue (2),
                                "LayoutType", StringValue ("RowFirst"));
  }
  else
  { // with pair nomber of nodes over the X axis, node is align with the node before the center of the mesh
    tmp_x = m_yNodes - 1;
    mobilityInternet.SetPositionAllocator ("ns3::GridPositionAllocator",
                                "MinX", DoubleValue (m_distAP * -1),
                                "MinY", DoubleValue ((m_distNodes * (tmp_x - 1)) / 2),
                                "DeltaX", DoubleValue (m_distAP),
                                "DeltaY", DoubleValue (0.0),
                                "GridWidth", UintegerValue (2),
                                "LayoutType", StringValue ("RowFirst"));
  }
  mobilityInternet.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobilityInternet.Install (nc_internet);

// Set Internet stack and OLSR protocol
  OlsrHelper olsr;
  InternetStackHelper internetStack;
  internetStack.SetRoutingHelper (olsr);
  internetStack.Install (nc_all);

// Config IPs for interfaces (all interfaces starts with "if_")
  Ipv4AddressHelper addrMesh;
  addrMesh.SetBase ("10.0.0.0", "255.255.255.0");
  Ipv4InterfaceContainer if_mesh = addrMesh.Assign (de_wireless);

  Ipv4AddressHelper addrInternet;
  addrInternet.SetBase ("1.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer if_internet = addrInternet.Assign (de_internet);

// Install applications
// Set sinks for receiving data
  PacketSinkHelper sinkTcp ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), 9));
  ApplicationContainer ac_sinkTcp = sinkTcp.Install (nc_destiny);
  ac_sinkTcp.Start (Seconds (0));
  ac_sinkTcp.Stop (Seconds (m_totalTime));

// Set traffic generator apps
  Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue (m_packetSize));
  Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue (m_txAppRate));
  Config::SetDefault ("ns3::OnOffApplication::OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=10.0]"));
  Config::SetDefault ("ns3::OnOffApplication::OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));

  if (m_flow == "upload" || m_flow == "both")
  {
    ApplicationContainer ac_onoffMesh [m_xNodes * m_yNodes]; // Creates 1 OnOff App in each mesh node TO internet
    OnOffHelper onoffMesh ("ns3::TcpSocketFactory", Address (InetSocketAddress (if_internet.GetAddress (0), 9)));
    for (tmp_x = 0; tmp_x < m_xNodes * m_yNodes; tmp_x++)
    {
      ac_onoffMesh [tmp_x] = onoffMesh.Install (nc_mesh.Get (tmp_x));
      ac_onoffMesh [tmp_x].Start (Seconds (15 + rand () % 10));
      ac_onoffMesh [tmp_x].Stop (Seconds (m_totalTime - 1 - rand () % 10));
    }
  }

  if (m_flow == "download" || m_flow == "both")
```

```
 {
   ApplicationContainer ac_onoffInternet [m_xNodes * m_yNodes]; // Creates 1 OnOff App for each mesh node FROM
internet
   for (tmp_x = 0; tmp_x < m_xNodes * m_yNodes; tmp_x++)
   {
     OnOffHelper onoffInternet ("ns3::TcpSocketFactory", Address (InetSocketAddress (if_mesh.GetAddress (tmp_x + 1),
9)));
     ac_onoffInternet [tmp_x] = onoffInternet.Install (nc_all.Get (0));
     ac_onoffInternet [tmp_x].Start (Seconds (15 + rand () % 10));
     ac_onoffInternet [tmp_x].Stop (Seconds (m_totalTime - 1 - rand () % 10));
   }
 }

/////PING FOR TESTS
 V4PingHelper ping1 (if_internet.GetAddress (0));
 ping1.SetAttribute ("Verbose", BooleanValue (true));
 ping1.SetAttribute ("Interval", TimeValue (Seconds (1)));
 ApplicationContainer appPingInternet1 = ping1.Install (nc_all.Get (m_xNodes * m_yNodes + 1));
 appPingInternet1.Start (Seconds (0));
 appPingInternet1.Stop (Seconds (m_totalTime - 1));

 V4PingHelper ping2 (if_mesh.GetAddress (m_xNodes * m_yNodes));
 ping2.SetAttribute ("Verbose", BooleanValue (true));
 ping2.SetAttribute ("Interval", TimeValue (Seconds (1)));
 ApplicationContainer appPingInternet2 = ping2.Install (nc_all.Get (0));
 appPingInternet2.Start (Seconds (0));
 appPingInternet2.Stop (Seconds (m_totalTime - 1));
/////END PING FOR TESTS


// Set simulation time
 Simulator::Stop (Seconds (m_totalTime));

// Set animation configs and create .xml file for NetAnim
 for (tmp_x = 2; tmp_x < m_xNodes * m_yNodes + 2; tmp_x++)
 {
   sprintf (tmp_char, "%d-STA", tmp_x);
   AnimationInterface::SetNodeDescription (nc_all.Get (tmp_x), tmp_char);
 }
 AnimationInterface::SetNodeDescription (nc_all.Get (0), "0-Internet");
 AnimationInterface::SetNodeDescription (nc_all.Get (1), "1-AP");
 AnimationInterface::SetNodeColor (nc_mesh, 0, 255, 0);
 AnimationInterface::SetNodeColor (nc_internet.Get (0), 255, 0, 0);
 AnimationInterface::SetNodeColor (nc_internet.Get (1), 0, 0, 255);
 if (m_drawAnim)
 {
   AnimationInterface anim (m_animFile);
   anim.EnablePacketMetadata(true);
   anim.EnableIpv4RouteTracking (m_routeFile, Seconds (0), Seconds (m_totalTime), Seconds (0.25));
 }

// Flow Monitor
 FlowMonitorHelper flowmon;
 Ptr<FlowMonitor> monitor = flowmon.Install (nc_destiny);
 monitor->Start (0);
 monitor->Stop (m_totalTime);
 if (m_drawAnim)
 {
              monitor->SerializeToXmlFile (m_flowmonFile, true, true);
 }
```

```
// Run the simulation
  Simulator::Run ();

////////////// Log data
// If requested, clean the flow file for a new use
  if (m_newFlowFile)
  {
    std::ostringstream os_clear;
    os_clear << m_statsFile << "_flows.csv";
    std::ofstream of_clear (os_clear.str().c_str(), std::ios::out | std::ios::trunc);
    of_clear << """Src IP""\t""Dest IP""\t";
    of_clear << """Protocol""\t""SrcPort-DestPort""\t";
    of_clear << """First Tx Pkt""\t""Last Tx Pkt""\t";
    of_clear << """First Rx Pkt""\t""Last Rx Pkt""\t";
    of_clear << """Total Tx Bytes""\t""Total Rx Bytes""\t";
    of_clear << """Total Tx Packets""\t""Total Rx Packets""\t";
    of_clear << """Total Delay""\t""Total Jitter""\n";
    of_clear.close ();
  }

// Open file to append new data
  std::ostringstream os;
  os << m_statsFile << "_flows.csv";
  std::ofstream of (os.str().c_str(), std::ios::out | std::ios::app);

//Print per flow statistics
  monitor->CheckForLostPackets ();
  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)
  {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
    of << t.sourceAddress << "\t" << t.destinationAddress << "\t";
    if (t.protocol == 6)
    {
      of << "TCP\t";
    }
    else
    {
      if (t.protocol == 17)
      {
        of << "UDP\t";
      }
      else
      {
        of << "N/A\t";
      }
    }
    of << t.sourcePort << "-" << t.destinationPort << "\t";
    of << i->second.timeFirstTxPacket.GetSeconds() << "\t" << i->second.timeLastTxPacket.GetSeconds() << "\t";
    of << i->second.timeFirstRxPacket.GetSeconds() << "\t" << i->second.timeLastRxPacket.GetSeconds() << "\t";
    of << i->second.txBytes << "\t" << i->second.rxBytes << "\t";
    of << i->second.txPackets << "\t" << i->second.rxPackets << "\t";
    of << i->second.delaySum << "\t" << i->second.jitterSum << "\n";
  }
  of.close ();
////////////// End Log data

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