UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA: INGENIERÍA ELECTRÓNICA

SIMULACIÓN Y ANÁLISIS DE REDES ESPORÁDICAS MÓVILES AD-HOC.

TESIS PREVIA A LA OBTENCION DEL TITULO DE:

INGENIERO ELECTRONICO

Autores:

Bismar Orlando Carrión Torres.

Luis Alberto Delgado Alvarez.

Director:

Ing. Juan Inga Ortega.

CUENCA, ENERO DE 2015

AUTORIA

Las ideas y contenidos expuestos en el presente proyecto, son de exclusiva responsabilidad de los autores y el patrimonio intelectual le pertenece a la Universidad Politécnica Salesiana

Cuenca, Enero de 2015

Bismar Carrión Torres.

Luis Delgado Alvarez.

CERTIFICO:

En calidad de DIRECTOR DE LA TESIS “Simulación y análisis de redes esporádicas móviles Ad-Hoc”, elaborada por Bismar Carrión y Luis Delgado, declaro y certifico la aprobación del presente trabajo de tesis basándose en la supervisión y revisión de su contenido.

Cuenca, Enero del 2015

_____

Ing. Juan Inga Ortega.

DIRECTOR DE TESIS

AGRADECIMIENTOS

Agradezco a mis padres por todo el apoyo brindado a lo largo de mi vida y formación profesional, a mi director de tesis Juan Inga por sus consejos brindados a lo largo del proyecto, a Esteban Ordoñez por su incondicional apoyo, a Víctor Saians por su gran ayuda y aporte al proyecto. Sin su apoyo no hubiera sido posible culminar esta etapa de mi vida. Gracias.

Bismar Carrión T.

Doy gracias a mi familia, profesores y amigos que han estado a lo largo de este camino recorrido, Juan y Esteban por ayudarnos con este proyecto, al grupo de investigaciones SSI de la Universidad de Vigo sobre todo a Víctor por su ayuda y apoyo, y como no Bismar por trabajar conmigo en este proyecto. Gracias.

Luis Delgado A.

DEDICATORIA

A mis padres, por su dedicación, trabajo y esfuerzo. Por incentivarme a cumplir con mis objetivos. Son el pilar de mi vida.

Por inspirarme a ser mejor, gracias por estar siempre a mi lado, Gabriela.

“Mucha gente pequeña en muchos lugares pequeños harán cosas pequeñas que transformarán al mundo”

Bismar Carrión T.

A mis padres, mi familia y amigos que me han apoyado para cumplir este trabajo, pues este es el principio de la vida profesional.

“Si no tuviste errores, nunca intentaste algo nuevo”

Luis Delgado A.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ABREVIATURAS USADAS

AODV	Ad-Hoc On Demand Distance Vector
AP	Access Point
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSDV	Destiantion Sequenced Distance Vector
DSR	Dynamic Source Routing
DT	Tabla de Distancia
HSLS	Hazy Sighted Link State Protocol
LCT	Tabla de Coste de un Enlace
MAC	Media Access Control
MANET	Mobile Ad-Hoc Network
MPR	Multipoint Relay
MRL	Lista de Mensaje de Retransmision
NaaS	Network as Service
NAM	Network Animator
NL	Node Lider
NS2	Network Simulator 2
NS3	Network Simulator 3
OLSR	Optimized Link State Routing
OSPF	Open Shortest Path First
P2P	Peer to Peer
REM	Res Esporádica Movil
RIP	Routing Information Protocol
RPGM	Reference Point Group Mobility
RREP	Route Reply
RREQ	Route Request
RSE	Red Social Esporádica
RT	Tabla de Ruta
SPORANGIUM	Sporadic Social Networks in the Next-Generation Information services for Users on the Move
SUMO	Simulation Of Urban Mobility

SynACK	ACK de Sincronizacion
SynData	Sincronizacion de Datos
SynRequest	Solicitud de Sincronizacion
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UWB	Ultra Wide Band Radio
VANET	Vehicular Ad-Hoc Network
VN	Nodos Virtual
VNAODV	AODV Basado en VNLayer
VNLayer	Capa de Nodo Virtual
VNLayer +	Capa de Nodo Virtual plus
VNRouting	Enrrutamiento de Nodo Virtual
WRP	Wireless Routing Protocol
ZRP	Zone Routing Protocol

RESUMEN

El presente proyecto “Simulación y Análisis de Redes Esporádicas Móviles Ad-Hoc” propone la implementación de una capa virtual para el establecimiento de rutas para encaminar información dentro de redes esporádicas móviles a ser implementadas por personas.

Es decir que dentro de esta propuesta, los nodos de enrutamiento fueron implementados con la capa virtual VNLayer dentro del simulador de redes de comunicaciones NS3.

Además, mediante el uso de BonnMotion se emuló el movimiento de los usuarios en una determinara área, representando el comportamiento humano en localidades como un museo, coliseo, centro de exposiciones, etc. En este sentido, el movimiento que describen los nodos móviles está relacionado con el Modelo de Movilidad de grupo de punto de Referencia (RPGM) que describe cierto comportamiento similar a la conducta humana en lo referente a la conformación del agrupamiento de personas.

Los resultados obtenidos en las diversas simulaciones y bajo diferentes escenarios planteados, se ha conseguido mejorar el desempeño de la red, además de verificar que protocolo de enrutamiento es el más adecuado para que la red Ad-Hoc pueda adaptarse al dinamismo de los nodos, propio de una red inalámbrica sin infraestructura física confirmado la potencialidad en el uso de la capa virtual.

INTRODUCCIÓN

En el desarrollo de los sistemas de comunicación y de las redes inalámbricas han transformado la vida de las personas gracias a la capacidad de movilidad que permiten a los usuarios. Además, la flexibilidad de los dispositivos de comunicación como smartphones, tablets y otros dispositivos móviles han permitido transformando la forma de comunicarse de las personas [19] [42].

En este sentido, las redes inalámbricas han iniciado con una infraestructura fija, donde existe un punto de acceso centralizado para que los dispositivos finales (usuarios) puedan transferir información de diferente índole. Por otra parte, en la actualidad, ya se conoce y se maneja el concepto de redes inalámbricas sin infraestructura totalmente fija. Esto permite la conexión de dispositivos finales sin la necesidad de un punto de acceso centralizado. De esta manera, cada dispositivo (nodo) dentro de la red puede comportarse como emisor, receptor o enrutador de datos [58] [59].

Así, las redes sin infraestructura o Ad-Hoc aprovechan el gran crecimiento de memoria y procesamiento de los dispositivos móviles [1] para así eliminar el punto de acceso centralizado, y realizar estos procesos de cómputo dentro de los dispositivos finales.

La aplicación de redes inalámbricas sin infraestructura afectan el ámbito social en forma positiva pues el usuario final busca tener la mejor experiencia (calidad de servicio, velocidad de datos) en su dispositivo móvil y estas redes ofrecen esa mejora de servicios al crear una red entre usuarios cercanos. La necesidad de la interacción de usuarios cercanos y considerando la falta de infraestructura fija, aparece el

concepto de Redes Sociales Esporádicas (RSE) [19], agrupando usuarios con afinidades e interés en común.

Considerando que las redes esporádicas permiten movilidad a los usuarios, es necesario que las rutas de encaminamiento de datos puedan adaptarse a estos movimientos que no son predecibles con facilidad y por tanto implica una topología de red cambiante. Esto puede provocar que el encaminamiento de la información sobrecargue la red debido a los constantes descubrimientos de ruta, por lo tanto una red con bajo rendimiento y baja calidad de servicio [59].

Para solucionar esto se propone el uso de nodos virtuales (VN) [4] [55] para solventar el repentino cambio de la topología de la red debido al movimiento de los nodos móviles implementando una capa virtual (VNLayer) para el encaminamiento de las redes esporádicas móviles REM [19].

CAPÍTULO 1. ESTADO DEL ARTE

En la actualidad, con el rápido incremento e innovación de tecnologías de comunicación inalámbrica y el desarrollo de nuevos dispositivos móviles con mayor capacidad de memoria y de procesamiento computacional [1], las sociedades y el estilo de vida de las personas se ha visto transformado. Dada la flexibilidad y ubicuidad de los dispositivos móviles, el usuario móvil se ve beneficiado, facilitando su acceso a la información ya sea para realizar tareas de trabajo, ocio o interacción social.

La evolución de las redes inalámbricas marca una nueva tendencia de conectividad entre dispositivos móviles, dándole al usuario la posibilidad de interactuar con otros por medio de la implementación de redes, específicamente mediante una red móvil Ad-Hoc (MANET). Este tipo de redes móviles normalmente se conforman por usuarios que tengan afinidades en común con el propósito de compartir información (imágenes, datos, video) y cuyo radio de interacción física está pensado para áreas pequeñas [19].

Cabe señalar que este tipo de redes carecen de una infraestructura de red fija, todos los procesos computacionales y funcionalidades de red están soportados por los dispositivos móviles. Existe una subclase, las redes vehiculares Ad-Hoc (VANET's) en la cual, los vehículos simulan un nodo móvil y son los encargados de brindar la funcionalidad de red.

Por otra parte, la conformación de una red esporádica móvil (REM) cambia el paradigma de las actuales redes sociales. Esto permite a los usuarios móviles una mayor interacción social e intercambio de información, agrupando a individuos con afinidades o intereses en común. Estos grupos de usuarios pueden compartir información en cualquier lugar como por ejemplo en un coliseo, en un museo, en un teatro, en una cafetería, etc.

Así, en un ambiente de Smart City, las REM's podrían aprovechar los recursos de pequeñas cloudlets[1], de modo que los usuarios podrían interactuar con otros usuarios

[1] Se entiende como una pequeña nubecilla con capacidad de brindar recursos y aplicaciones formada por un clúster de ordenadores conectado a Internet que se encuentran a un salto inalámbrico de sus dispositivos móviles asociados [38].

cercanos para compartir y procesar información, además de abrir nuevos canales de e-commerce [19].

Un inconveniente propio de las redes móviles MANET, es la movilidad de los usuarios y los cambios en la topología de red [2]. Esto conlleva un efecto significativo en la conformación de REM's ya que el inherente efecto de la movilidad y los cambios de velocidad pueden causar perdida de paquetes, retrasos en la conformación y establecimiento de nuevas rutas de encaminamiento, latencias en la red, colisiones de paquetes o simplemente la extinción de la misma.

De acuerdo a lo mencionado, en este capítulo expondremos la situación actual y los estudios realizados sobre las redes MANET y los conceptos básicos que relacionan a las redes Ad-Hoc. En este contexto se definirán los protocolos de encaminamiento y las tecnologías de acceso inalámbrico que sostiene la creación de dichas redes. Considerando estos aspectos, en capítulos posteriores se define una herramienta de simulación de red y que sea capaz de establecer escenarios de movilidad.

Además, en la sección 1.1 se realiza una descripción de la situación y trabajos realizados en el área de las redes Ad-Hoc, en la sección 1.2 se describen las redes Ad-Hoc; en la sección 1.3 se analizan los protocolos de encaminamiento, en la sección 1.4 se describen las tecnologías de acceso inalámbrico y finalmente en 1.5 se muestran los simuladores de movilidad, red y transporte.

1.1. Estudios realizados sobre redes Ad-Hoc

Como parte del Grupo de Investigación en Telecomunicaciones de la Universidad Politécnica Salesiana y siguiendo el mismo modelo de estudio en los trabajos desarrollados para ambientes vehiculares desarrollado por CALDAS-ZARUMA y HURTADO-SIGUENZA, en [3] y [4] se ha pensado en el uso de software libre para los respectivos análisis de manera que integre los conceptos de movilidad de red.

Wu en [55] utiliza el concepto de virtualización de red, planteando el uso de VNLayer (en la sección 3.1 se realiza un estudio más profundo) como una

herramienta de encaminamiento reactivo implementando el protocolo de encaminamiento VNAODV y comparándolo con su símil AODV.

Mediante el simulador NS2, HURTADO-SIGUENZA [4] analizan el concepto de virtualización para hacer frente a los problemas que presentan las redes MANET's en entornos vehiculares en cuanto a la variación de su topología.

Varios son los estudios realizados en el campo de las redes MANET, especialmente en entornos vehiculares. Según Bravo, et.al [53] [54] el concepto de virtualización de red se adaptan a las necesidades de las redes MANET brindando soluciones a los problemas debido al rápido cambio de las topología por el movimiento de los nodos participantes en la red.

Otro enfoque similar al estudio de estas redes es la presentada en el proyecto SPORANGIUM[2] [19] [42] del grupo SSI de la Universidad de Vigo. En su nivel de arquitectura más bajo se plantea el reto de comunicación entre dispositivos móviles en ausencia de una infraestructura de red fija. El estudio de este equipo de trabajo converge en el uso de conceptos de virtualización, integrando escenarios de movilidad vehicular y pedestre. Plantean el uso de una VNLayer + superior a su antecesora la VNLayer presentada por Wu [55].

1.2. Redes Ad-Hoc

Gracias a los avances tecnológicos, sobre todo en el área militar, las redes Ad-Hoc no son una excepción, y esto es debido a las prestaciones y mejora de ciertos servicios como la cobertura y el tráfico de datos hacia la nube, que se ve minimizado mediante el uso de redes Ad-Hoc, frente a otro tipo de infraestructuras (3G, Wifi).

[2] Sporadic Social Networks in the Next-Generation Information services for Users on the Move. Busca habilitar nuevas formas de interacción social cambiando el paradigma de mundo virtual que las actuales redes sociales ofrecen a través del impulso de Redes Sociales Esporádicas (RSE). Ver Sección 2.1

Las redes Ad-Hoc pertenecen a un grupo de redes inalámbricas de multisalto en donde la información es enviada y reenviada por los dispositivos próximos [20]. Por ello, una red MANET se considera como un sistema autónomo de nodos móviles, conectados a través de enlaces inalámbricos.

Así, se establece una red de forma arbitraria debido al movimiento aleatorio de los nodos (velocidad y dirección aleatoria). Por lo tanto es una topología inestable y difícil de predecir, ya que no existe una infraestructura fija. La información es transmitida en multisaltos de manera peer to peer (P2P), entendiéndose como una red de dispositivos que funcionan sin clientes ni servidores fijos, sino que son serie de nodos con un comportamiento igual entre sí [21].

Una red Ad-Hoc es una red de área local de conexión inalámbrica o de conexión fija temporal, en la cual dispositivos móviles o portátiles son parte de la red mientras estos estén cercanos [21], se podría decir que los usuarios a formar parte de la red pueden ser parte de la misma con o sin el consentimiento de los usuarios. En consecuencia, una MANET puede enviar la información de una fuente a un destino usando nodos intermedios como su vía de conexión, y transmitiendo con el método de guardar-enviar [20].

1.3. Protocolos de Encaminamiento Para MANETs

Los nodos Ad-Hoc actúan como Routers (enrutadores), y debido a la movilidad no pueden disponer de las características de dispositivos que forman una infraestructura fija como son Gateways (Compuertas de enlace/Conversor de protocolos), DHCP (Dynamic Host Configuration Protocol), DNS (Domain Name System). Por lo tanto, lo nodos Ad-Hoc deben incluir todas las funcionalidades de ruteo y direccionamiento. Estos nodos Ad-Hoc deben tener toda la información de las rutas, así como los mecanismos para descubrir rutas para otros nodos que están cerca y se conectan a la red [22].

De acuerdo con lo anterior, es necesario que los nodos dispongan de protocolos de comunicación adecuados. Entonces, como se ve en [23] se pueden diferenciar tres grandes grupos de protocolos existentes para crear redes MANETs:

- Proactivos.
- Reactivos.
- Híbridos.

1.3.1. Protocolos Proactivos

Los nodos mantienen información de encaminamiento hacia todos los dispositivos de la red. Dentro de este grupo se encuadran aquellos protocolos que basan su funcionamiento en el intercambio periódico de mensajes para actualizar sus tablas de rutas.

En este sentido, la característica principal es que, aunque no haya movilidad entre los dispositivos que componen la red o no cambie la topología, siempre va a haber un número mínimo de envío de mensajes para mantener la conexión y tener constancia de los nodos que forman la red [23] [24].

Los protocolos de enrutamiento en Internet (es decir, RIP o Routing Information Protocol basado en vectores distancia y OSPF u Open Shortest Path First basados en estado del enlace) entran dentro de la categoría de protocolos proactivos. Sin embargo, estos protocolos no son adecuados para las redes de bajos recursos y redes móviles Ad-Hoc, debido a sus grandes sobrecargas y el no poder entregar la información de manera más directa a los nodos involucrados [26] [27]. Dentro de los Protocolos Proactivos se ha visto conveniente revisar los que han sido de mayor uso en las redes móviles Ad-Hoc, y como estos pueden ser adoptados dependiendo del tipo de topología.

1.3.2. DSDV (Destination Sequenced Distance-Vector)

La meta de DSDV es desarrollar un protocolo que conserve la sencillez del protocolo RIP, garantizando al mismo tiempo estar libre de los bucles. La idea principal es el uso de números de secuencia de destino, es decir saber cuántos saltos necesito para llegar a un nodo destino, para lograr que la red formada esté libre de bucles sin ningún tipo de coordinación entre los nodos [25].

DSDV basa su funcionamiento en la actualización de una tabla de rutas donde aparece cada destino con su distancia (número de saltos), su número de secuencia que indica cuál es la información más actualizada, y el siguiente salto hacia el nodo final. Debido a que está dentro del grupo de proactivos, tiene siempre constancia de todos los dispositivos involucrados en la topología.

Cada nodo mantiene un número de secuencia que asciende de manera implícita, también mantiene el mayor número de secuencia conocido para cada destino en la tabla de enrutamiento.

La información de distancia o métrica (es decir, número de saltos hacia los nodos), para cada destino, normalmente intercambiados a través de actualizaciones de enrutamiento entre los vecinos con los protocolos de vector de distancia, se etiqueta con el número de secuencia de destino correspondiente, dicha información en caso de no estar siempre actualizadas en los nodos podría provocar una pérdida de paquetes o peor aún no tener rutas para llegar a los nodos destino que podrían ya no estar presentes.

Estos números de secuencia se utilizan para determinar qué tan actualizada esta la información de distancia generada por dos nodos para el mismo destino (el nodo con un número de secuencia de destino superior tiene la información más reciente).

Por lo general los números de secuencia de destino a lo largo de cualquier ruta válida aumenten hasta llegar al destino. Esto se da por la generación de bucles de enrutamiento, producto de que un grupo de enrutadores no envían datos al destino sino que transmiten información solo entre ellos. En consecuencia se incrementa el tiempo de transmisión de datos y probablemente se podrían perder los datos. Para

esto hay que prevenir bucles de enrutamiento a través de una elección y gestión adecuada de los protocolos de enrutamiento [25].

Además DSDV, como en DBF (Distributed Bellman-Ford) [28], un nodo puede recibir primero una ruta con un número de saltos más largo en lugar de una ruta con el menor número de saltos. Por lo tanto, siempre propagar la información de distancia inmediatamente después de un cambio puede verse reflejado en muchas actualizaciones que se propagarán a través de la red, lo que resulta en una enorme sobrecarga.

Así, "*DSDV estima un tiempo de establecimiento de la ruta (tiempo que tarda en llegar la ruta con la distancia más corta después de obtener la ruta con una distancia superior) basado en la información pasada y lo utiliza para evitar la propagación de todas las mejoras en la información de distancia*." [24].

1.3.2.1.WRP (Wireless Routing Protocol) [29] [38]

WRP se diferencia de DSDV en la forma de mantener las rutas y en los procedimientos de actualización. Mientras que DSDV solamente contiene una tabla, WRP mantiene varias con el fin de tener información más precisa. Las tablas que utiliza son:

- Tabla de distancias (DT).
- Tabla de rutas (RT).
- Tabla de coste de un enlace (LCT).
- Lista de mensajes de retransmisión (MRL).

Los algoritmos WRP utilizan información del siguiente salto, y del penúltimo salto para superar el problema de conteo de saltos hasta el infinito. Cada nodo se actualiza con la trayectoria más corta de las ramas (diferentes rutas hasta uno o varios destinos) de cada uno de sus vecinos. Además, utiliza el costo de sus eslabones adyacentes (tiempo en llegar a otros nodos) junto con las diferentes rutas más cortas reportadas por los vecinos para actualizar su propias ramas; el nodo reporta los

cambios de su propia rama a todos los vecinos en forma de actualizaciones que contienen la distancia y la información del penúltimo salto a cada destino.

De esta manera WRP mejora mediante la verificación del penúltimo salto que es reportado por todos los vecinos para actualizar tablas propias de cada nodo. Con este mecanismo, WRP reduce la posibilidad de bucles de enrutamiento temporales (rutas que ya están obsoletas o distancias muy largas), que a su vez se traduce en menor tiempo de entrega de información. Una desventaja importante de WRP es su requisito de tener siempre una conectividad fiable hacia otro nodo (vecino) para la entrega fiable y ordenada de mensajes de enrutamiento, ya que de otra manera se tendrá como entrada una distancia infinita en sus tablas de enrutamiento.

1.3.2.2.OLSR (Optimized Link State Routing) [30]

Este protocolo se caracteriza por intentar optimizar el número de mensajes de control que se producen en la red. El hecho de que todos los dispositivos estén continuamente retransmitiendo mensajes de control para informar sobre la topología, es un proceso costoso desde el punto de vista de la señalización implicada. OLSR intenta solucionar este problema eligiendo estratégicamente un conjunto de nodos MultiPoint Relays (MPR's), mediante los cuales se puede llegar a cualquier punto de la malla (topología de la red).

Los nodos designados como MPR (cada nodo elige un pequeño subconjunto de nodos, suficiente para cubrir una distancia de hasta dos saltos) por algún nodo, se les permiten generar actualizaciones de estado de enlaces. Además, las actualizaciones de estado de enlace sólo contienen los enlaces entre nodos MPR y sus nodos selectores con el fin de mantener el tamaño de la tabla de actualización pequeño. Por lo tanto, sólo información parcial de la topología se pone a disposición en cada nodo. Sin embargo, esta información es suficiente para calcular localmente el camino más corto para todos los demás nodos que están conectados a los nodos MPR, en donde al menos una ruta consta solo de nodos MPR.

OLSR utiliza solamente actualizaciones periódicas para la diseminación del estado de enlace. Esto se da debido a que la sobrecarga total se determina por el producto del número de nodos que generan las actualizaciones, el número de nodos que reenvío cada actualización y el tamaño de cada actualización.

El protocolo OLSR reduce la sobrecarga en comparación con un protocolo de estado de enlace cuando la red es densa ya que se utiliza actualizaciones solo en los nodos MPR. Para una red dispersa, OLSR degenera el protocolo de estado de enlace tradicional, por último, utilizando únicamente las actualizaciones periódicas hace que la elección de actualización de intervalo (actualización de rutas en tiempos más largos) crítica al reaccionar a cambios en la topología, ya que las rutas pueden ya no existir o ser más cortas.

1.3.3. Protocolos Reactivos

En este tipo de protocolos los nodos actualizan las tablas de encaminamiento solamente en caso de necesidad, también se lo llama a este protocolo como enrutamiento en demanda. Su funcionamiento se basa en que si un nodo origen necesita enviar un paquete, primero busca en su tabla de enrutamiento y en caso de no tener una ruta, este nodo origen lleva a cabo un descubrimiento de ruta para encontrar un camino hasta su destino, lo cual mejora el desempeño de la red al no tener un envió innecesario de datos por mantenimiento de rutas, en caminos que no tienen flujo de datos.

La característica principal es que, si hay movilidad o cambio en la topología, los dispositivos no actualizan sus tablas de encaminamiento a menos que lo requieran. De hecho, no se conoce cómo es la red completa porque no se sabe cuáles son las tablas de los demás nodos, por lo que únicamente hay disponible información sobre las rutas activas en cada nodo [21].

1.3.3.1.DSR (Dynamic Source Routing) [33] [34]

Este protocolo está diseñado para restringir el ancho de banda consumido por paquetes de control eliminando las actualizaciones periódicas. Para obtener información, se aprovecha de los mensajes de datos mandados por el camino que se está creando, de este modo, lo que hace es extraer de estos mensajes toda la información relevante para los dispositivos origen y destino de la comunicación. Intenta incorporar las ventajas de los protocolos proactivos y reactivos.

Cuando un nodo de la red Ad-Hoc intenta enviar un paquete de datos a un destino para el que aún no se conoce ruta, se utiliza un proceso dinámico para el descubrimiento de ruta. Este mecanismo de descubrimiento de ruta empieza cuando un nodo origen necesita transmitir un paquete de datos hacia un nodo destino pero no conoce la ruta que debe seguir el paquete para llegar al destino.

En primera instancia el nodo origen inunda la red con la transmisión de paquetes RREQ (Route Request) a sus vecinos, conteniendo el identificador del nodo origen, el del nodo destino y la ruta parcialmente calculada. Cada vez que un RREQ llega a un nodo, este añade su identificador a la ruta contenida en el paquete y transmite a sus vecinos el nuevo RREQ. Este proceso se repite hasta que el nodo destino recibe uno el paquete.

Una vez que el nodo destino recibe el RREQ, responde mediante un Route Reply (RREP) hacia el nodo origen, mediante la ruta que tomo RREQ con el fin de informar al nodo origen la ruta que debe tomar para el envío de paquete.

1.3.3.2.AODV (Ad-hoc On-demand Distance Vector) [31] [32]

Es uno de los que se encuadran dentro de los reactivos. Por lo tanto, no utiliza mensajes de control hasta que un nodo determinado no necesita una ruta hacia un destino.

La idea principal de enrutamiento bajo demanda es encontrar y mantener solo rutas que se necesitan. Los protocolos de enrutamiento proactivos mantienen todas las rutas sin tener en cuenta su uso final. La ventaja evidente con el descubrimiento de

rutas en demanda, es para evitar incurrir en el costo de mantenimiento de rutas que no se utilizan. Este enfoque es atractivo cuando el tráfico de la red es esporádica, de ráfagas y dirigida principalmente hacia un pequeño subconjunto de nodos.

Utiliza tablas de enrutamiento tradicionales, una entrada por cada destino. Esto está en contraste con DSR, que puede mantener múltiples entradas de caché de ruta para cada destino. Sin enrutamiento de origen, AODV se basa en entradas de la tabla de enrutamiento para propagar un paquete de respuesta o Route Reply (RREP) de vuelta a la fuente y, posteriormente, enrutar paquetes de datos hasta el destino.

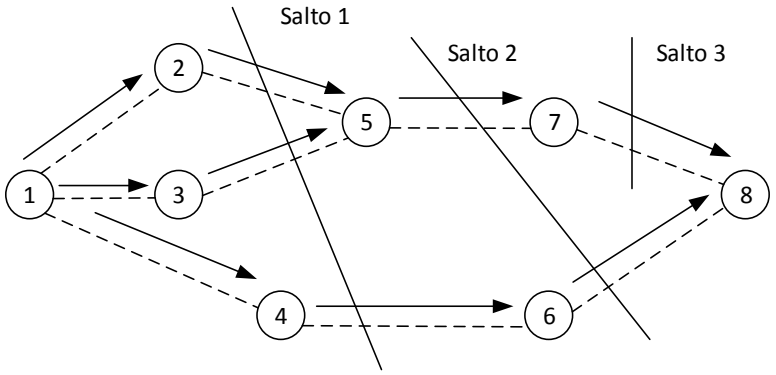
Además, ODV utiliza números de secuencia de destino como en DSDV para evitar bucles de enrutamiento y para determinar la frescura de la información de enrutamiento. Estos números de secuencia se realizan por todos los paquetes de enrutamiento.

Se utiliza los números de secuencia para inferir la frescura de la información de enrutamiento y los nodos sólo mantienen la información de ruta para un destino correspondiente al último número de secuencia conocida; rutas con números de secuencia más viejos se descartan a pesar de que aún sea válida. AODV también utiliza un mecanismo de expiración de ruta basada en un temporizador para purgar rápidamente rutas antiguas. De esta forma, si un valor bajo se elige para el tiempo de espera, las rutas válidas pueden ser innecesariamente descartados.

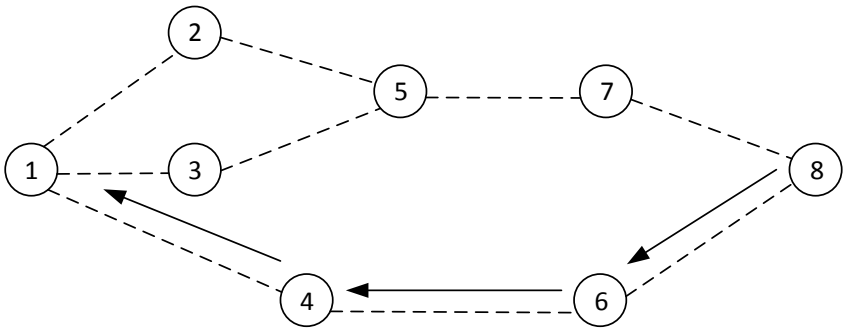
En manera sintetizada este protocolo envía primero una petición de ruta RREQ, como se ve en la Figura 1.1 (a), la cual se propaga por toda la red. Los números de secuencia de destino se utilizan para garantizar que todas las rutas están libre de bucles y contienen la información más reciente de la ruta. Cada nodo tiene un número de secuencia único y un ID de difusión (broadcast), que se incrementa cada vez que el nodo inicia un RREQ. El ID de difusión, junto con la dirección IP del nodo, identifica de manera única cada RREQ. El nodo iniciador incluye en el RREQ su número de secuencia, su ID de difusión y el número de secuencia más reciente.

Los nodos intermedios responden sólo si tienen una ruta hacia el destino con un número de secuencia mayor o al menos igual a la contenida en el RREQ, los nodos intermedios registran la dirección del nodo vecino del que reciben la primera copia del paquete de difusión. Esto establece el mejor camino de regreso.

Una vez que el RREQ llega al destino o un nodo intermedio con una ruta actualizada hasta el destino, el nodo intermedio o el nodo destino envía una respuesta de ruta (RREP) en unicast[3] de vuelta al vecino del que recibió el primer RREQ. A medida que el RREP viaja en camino de regreso al origen, los nodos intermedios en esta ruta establecida mantienen sus entradas de ruta para indicar el nodo desde el que se ha recibido el RREP, es decir ya crean la ruta que se establecerá entre ese nodo origen y destino, como se observa en la Figura 1.1 (b).



(a)



(b)

Figura 0.1 (a) Propagación del paquete RREQ y (b) Camino establecido por el paquete RREP [32]

1.3.4. Protocolos Híbridos

En general, lo que se pretende es dividir la red en grupos en los que de forma local se utilice un protocolo proactivo, mientras que para conocer las rutas hacia los demás

[3] Envió de información de un único emisor a un único receptor.

grupos se utilice uno reactivo. De esta manera, localmente se tendrá constancia de todos los nodos que forman el grupo en el que se encuentra el dispositivo. Debido a este hecho, no se tendrá que inundar la totalidad de la topología con mensajes de control para conocer los caminos hacia las demás comunidades [21] [24].

1.3.4.1.ZRP (Zone Routing Protocol) [35] [36]

Como tal, utiliza un tipo de encaminamiento proactivo en una zona delimitada y para el exterior, uno reactivo. Este protocolo está pensado para redes grandes con una movilidad elevada. De hecho, su principal logro es que la zona delimitada no es un elemento estático, sino que va modificándose según las condiciones de movilidad y tráfico. Los nodos, por sí solos, van eligiendo cuáles son los dispositivos periféricos que permiten llegar a los demás nodos fuera de la comunidad. Para ello, el radio (número de saltos máximo dentro de una zona) se configura automáticamente.

ZRP define una zona para cada nodo A que incluye todos los nodos que se encuentran a cierta distancia en saltos, denominado radio de la zona, alrededor del nodo A. Los nodos que están exactamente en el radio de la zona de distancia desde el nodo A se llaman nodos de frontera de la zona de A. Un protocolo proactivo de estado de enlace se utiliza para mantener cada nodo alerta de la topología completa dentro de su zona.

Cuando un nodo A necesita obtener una ruta a otro nodo B que no está en su zona, inicia un descubrimiento de ruta reactiva que funciona de forma similar a las inundaciones salvo que involucra sólo nodos de frontera de A y los nodos de frontera del nodo B y así sucesivamente.

La consulta de ruta acumula la ruta atravesada en su camino hacia el exterior desde A (como en el enrutamiento de origen) y cuando la consulta finalmente llega a un nodo de frontera que se encuentra en la zona de destino de B, este nodo de frontera envía de vuelta una respuesta a través de la ruta acumulada de las consultas. Dependiendo de la elección del radio de la zona, ZRP puede comportarse como un protocolo puramente proactivo, un puramente reactivo, o una combinación de los dos.

1.3.4.2.HSLS (Hazy Sighted Link State Protocol) [37]

HSLS es un protocolo de estado de enlace basado en difusión limitada. Aunque HSLS de por sí no tiene ningún componente reactivo como en ZRP, este parcialmente exhibe un comportamiento típico de protocolos reactivos, específicamente el uso de rutas óptimas.

La idea principal es controlar el alcance de difusión de estado de enlace en el espacio y el tiempo de modo que los nodos más cercanos se envían actualizaciones de estado de enlace con mayor frecuencia en comparación con los nodos que se encuentran a mayor distancia.

Así nodos distantes a través de actualizaciones infrecuentes sólo se proporcionan "sugerencias" para encaminar un paquete más hacia el destino. Mientras el paquete se acerca al destino, se aprovecha de la información progresiva de enrutamiento reciente que mejora sus posibilidades de llegar al destino.

Una consecuencia de esta estrategia de difusión limitada es que un paquete puede tomar rutas sub-óptimas inicialmente, pero eventualmente llega a su destino a través de una ruta óptima. De este modo se permite una cierta cantidad de enrutamiento sub-óptimo para reducir la sobrecarga de control global.

En la tabla 1.1 se puede apreciar de una mejor manera las diferencias en las características de los protocolos Proactivos y Reactivos [41].

Características	Proactivos	Reactivos
Estructura	Plana y en algunos casos jerárquica (OLSR).	En la mayoría de los casos plana, excepto en CBRP (*Cluster Based Routing Protocol*)
Disponibilidad de las Rutas	Siempre disponibles.	Se determinan cuando se necesiten.
Volumen de tráfico de control	Alto.	Menor que los protocolos proactivos.
Actualizaciones periódicas	Se utilizan algunos protocolos tienen mecanismos para usar las actualizaciones en función de la necesidad (OLSR).	Normalmente no son utilizados. Tan solo para comprobar la disponibilidad de los nodos vecinos.
Sobrecarga por datos de control	Alta.	Baja.

Tiempo en obtener un ruta	Bajo.	Alto.
Requerimientos de almacenamiento	Altos.	Depende del número de rutas que se mantengan. Menor que los protocolos proactivos.
Ancho de banda necesario	Alto.	Bajo.
Energía consumida	Alta.	Baja.
Nivel de retardo	Bajo si las rutas están determinadas.	Mayor que en los proactivos.
Problemas de escalabilidad	Por encima de los 100 nodos.	Por encima de varios cientos de nodos.

Tabla 0.1 Características de los protocolos Proactivos y Reactivos

1.4.Tecnologías de Acceso Inalámbrico Para Redes Ad-Hoc

Como se expuso en la sección 1.2, MANET se caracteriza por no presentar una infraestructura de red fija. Se presentan diversas tecnologías de acceso inalámbrico capaces de sustentar la creación o conformación de REMs, entre las que se destacan por características de ubicuidad a tecnologías como Bluetooth, Wifi, Wi-Fi Direct, y Ultra-WideBand Radio (UWB) [13].

1.4.1. Bluetooth (Estándar IEEE 802.15.1)

El desarrollo de la tecnología Bluetooth y el interés de la industria por proporcionar conectividad inalámbrica de corto alcance reflejan una expectativa prometedora para las redes Ad-Hoc. Esta tecnología ofrece ventajas considerables para la formación de redes Ad-Hoc, ofreciendo ubicuidad y localización de forma transparente de los dispositivos cercanos. Bluetooth es una tecnología inalámbrica de corto alcance pensada para interconectar dispositivos como PDAs, computadoras portátiles, Smartphones, etc. Diseñada para un alcance de 10 metros, opera en la banda de frecuencias ISM (industrial, scientific and medical) de 2.4 GHz [11].

Además, Bluetooth distingue tres tipos de topología de red, topología P2P, piconet y scatter-net. Entendiéndose por piconet a la topología formada hasta por 7

dispositivos esclavos y un maestro, teniendo en cuenta que los dispositivos esclavos no pueden establecer canales lógicos entre sí, necesariamente deben pasar por el nodo maestro; mientras que scatter-net es la unión de varias piconet, donde se distingue la configuración maestro-esclavo, en la que un dispositivo siendo el esclavo de una red puede ser el maestro para otra [12]. La descripción anterior se puede apreciar en la figura 1.2.

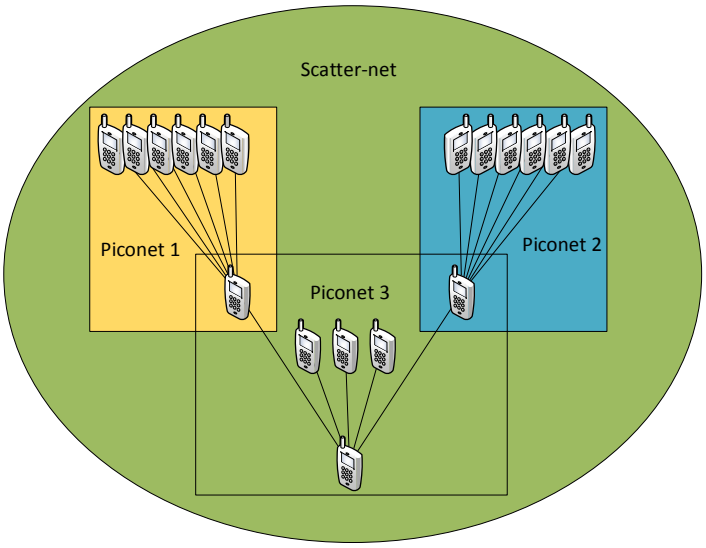


Figura 0.2 Dispositivos en topología Scatter-net, formada por 3 pico-nets

1.4.2. Wifi (Estándar IEEE 802.11a, 802.11b, 802.11g, 802.11g)

Para la implementación de redes Ad-Hoc, Wifi presenta dos topologías de red, una en modo infraestructura que utiliza un punto de acceso central y otra topología sin infraestructura, pensada para la creación de redes Ad-Hoc tanto P2P o tipo mesh (red inalámbrica tipo malla). En la figura 1.3 se muestra una red mesh Ad-Hoc.

Wifi Ad-Hoc en configuración mesh es capaz de transmitir mensajes de un dispositivo a otro por diferentes caminos, de forma de que en caso de que un dispositivo falle o desaparezca este no afecta a los demás dispositivos. Esto significa que el ancho de banda de la red sea compartido, la configuración se vuelve más compleja volviendo a los equipos más costosos y sumando latencias debido a los diferentes saltos que pueden existir en la red [13].

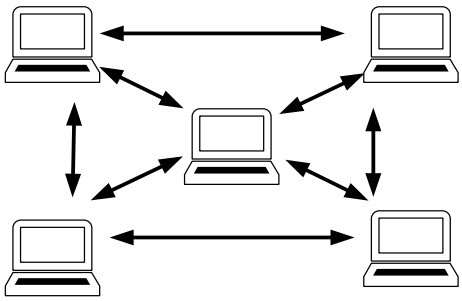


Figura 0.3 Red Ad-Hoc en configuración mesh

1.4.3. Wifi Direct

El desarrollo de los actuales Smartphones, la necesidad de los usuarios por mantener mayor conectividad y la falta de una tecnología adecuada de acceso inalámbrico capaz de soportar redes Ad-Hoc dieron paso a Wifi-Direct.

Wifi Direct consiste en la creación de un grupo P2P para dar soporte a los nodos que desean realizar una conexión Ad-Hoc. Uno de los nodos debe asumir la función de propietario de dicho grupo, transformándose en un soft-AP[4]. Posteriormente se incorporan los clientes permitiendo una conexión P2P entre los nodos y el propietario, similar al proceso de Bluetooth [14].

En la actualidad, Wifi-Direct está muy limitado a la hora de establecer redes MANET o configuraciones tipo mesh, ya que de momento soporta topologías tipo estrella como se muestra en la figura 1.4 [15]. A pesar de ser una tecnología nueva en los dispositivos móviles, WiFi Direct vislumbra muchas posibilidades de desarrollo ya que la mayoría de Smartphones de gama alta ya la incluyen.

[4] También conocido como enrutador virtual, permite al equipo con una única tarjeta de red inalámbrica emular un punto de acceso inalámbrico [39].

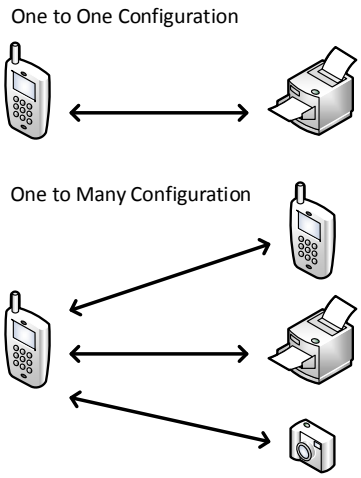


Figura 0.4 Configuración de topologías para Wifi-Direct

1.4.4. Ultra-WideBand Radio (UWB)

Esta novedosa tecnología de radio, es la basada en el estándar 802.15.3 que emite a muy baja potencia y por tanto su consumo de energía es reducido. Presenta un inconveniente en su alcance de transmisión ya que se limita a distancias menores a 10 metros, pero las tasas de transferencia son muy elevadas llegando a los 480 Mbps [16] [17].

UWB hace uso de un espectro de frecuencias que van desde 3.1 GHz hasta 10.6 GHz. Alcanzado un máximo 7.5 GHz de ancho de banda. Cada canal de radio tiene una anchura de más de 500 MHz. La técnica de transmisión UWB un candidato prometedor para MANETs multi-hop en escenarios de corto alcance [18].

1.5.Simuladores de Redes de Telecomunicaciones y Movilidad

Existen varias consideraciones tras la selección de las herramientas de simulación tanto para redes de telecomunicaciones como los distintos escenarios de movilidad. Es necesario la disponibilidad de licencias de software de código abierto, compatibilidad con el respectivo sistema operativo LINUX, disponibilidad de una interfaz gráfica de simulación, capacidad de modelar diferentes escenarios de

movilidad, disponer herramientas de análisis para medir el rendimiento de la red y el efecto de los protocolos de encaminamiento según las trazas de movilidad. Son necesarios estos aspectos ya que la elección de correcta de las herramientas de simulación podrá concatenar diferentes investigaciones futuras.

1.5.1. Network Simulator 2 (NS-2)

NS-2 es un simulador de eventos discretos originalmente desarrollado dentro del proyecto VINT[5], de uso extendido en la comunicad académica para la investigación de redes de telecomunicaciones. Es una herramienta basada en software libre, capaz de realizar simulaciones de redes cableadas e inalámbricas. Maneja protocolos de encaminamiento y multidifusión, etc. Para la simulación maneja un lenguaje escrito en C++, que sirve para modelar el comportamiento de los nodos, y además comandos oTcl (Object-oriented Tool Command Language) que permite al usuario, interactuar con el simulador mediante un script en el que se detalla los parámetros de la topología a simular. La primera versión salió en 1995 y en noviembre de 2011 se presentó la última versión 2.35 de NS-2 [6].

1.5.2. Network Simulator 3 (NS-3)

NS-3 es un simulador de evento discreto y desarrollado en C++ y de código abierto. Las simulaciones de red en NS-3 pueden ser implementadas puramente en C++, y de forma opcional, partes de la simulación se puede realizar utilizando Python. Es necesario recalcar que NS-3 no es una extensión compatible con su predecesor NS-2, a pesar de que ambos sean escritos en C++.

NS-3 posee la capacidad de simular redes cableadas o inalámbricas, modelando todos los elementos que conforman una red de computadoras, tanto como dispositivos finales o nodos centrales. Permite también, la simulación de canales

[5] Virtual Internetwork Testbeb (VINT) es un proyecto de investigación cuyo objetivo era la creación de un simulador para el estudio de la escalabilidad y la interconexión entre protocolos de redes actuales y futuras [40]

Wifi, P2P, y demás basados en el estándar IEEE 802.11. Otra ventaja de NS-3 es que se puede simular modelos de movilidad. El proyecto NS-3 empezó en el 2006 y actualmente está en su versión NS-3.20. [7].

1.5.3. GloMoSim

Siendo la versión gratuita de QualNet[6], ofrece una alternativa a NS-2 pero con capacidades reducidas debido a la menor penetración de esta herramienta en áreas investigativas, lo que conlleva que el conjunto de protocolos disponibles sean reducido [5].

1.5.4. OMNeT++

Es un simulador de eventos discretos, modular y orientado a objetos. Normalmente se utiliza para modelar tráfico en redes de telecomunicaciones, protocolos, multiprocesadores, permite evaluar aspectos de rendimiento de sistemas software complejos.

Está basada en C++ y dedicada principalmente a la creación de simulaciones de red incluyendo no sólo redes cableadas e inalámbricas, sino redes de colas, redes de sensores, redes inalámbricas Ad-Hoc, redes fotónicas, etc.

El simulador funciona en Windows y en varias distribuciones de Unix soportando diversos compiladores de C++. Además es libre para uso académico y en la actualidad una plataforma ampliamente utilizada por la comunidad académica [10].

1.5.5. MobiREAL

[6] Software de licencia pagada para la simulación de redes fijas y móviles permitiendo integración con aplicaciones de terceros, tales como analizadores de protocolos y sistemas de gestión de red. http://web.scalable-networks.com/content/qualnet

Este novedoso simulador (desarrollado en C++), es capaz de proporcionar diferentes modelos de movilidad, contemplando escenarios para la movilidad real de personas y vehículos. Permite cambiar su comportamiento en función de una determinada aplicación. También permite una descripción detallada de las aplicaciones de red, protocolos de enrutamiento, infraestructuras de red, etc.

MobiREAL puede simular una amplia variedad de redes móviles basados en reglas probabilísticas para describir el comportamiento de los nodos móviles mediante la adición de modelos de movilidad al simulador de redes GTNetS. Por medio de, MobiREAL Animator se puede visualizar el movimiento del nodo, la conectividad, y la transmisión de paquetes, facilitando la comprensión de los resultado. Para entornos de movilidad vehicular, se utiliza la modificación de esta herramienta que incluye un simulador de tráfico llamado NetStream. Es de uso limitado por no ser un software libre [5].

1.5.6. SUMO (Simulation of Urban Mobility)

SUMO es una herramienta muy conocida, distribuida bajo licencia GLP para la generación de redes peatonales y vehiculares. Con la característica de ofrecer una rápida GUI o por medio de línea de comandos permite simular varios tipos de vehículos, aceras, calles de varios carriles, cruces y zonas de paseo.

Posee modelos peatonales como [9]:

- Modelo de No-Interacción: En este modelo no hay interacción entre peatones ni vehículos, el peatón en el cruce de calle salta hacia la acera evitando el contacto con el vehículo. Por lo anterior tiene un alto nivel de velocidad en su ejecución.
- Modelo striping: Este modelo asigna coordenadas 2D con un camino para vereda y cruces para cada peatón. Posee la capacidad de evitar colisiones, existe interacción entre otros peatones y con los vehículos. En los cruces el peatón ya no salta, sino que espera a que no existan vehículos, si un vehículo

se aproxima y el peatón aún no termina de cruzar el vehículo se detiene y el cruce se ve como bloqueado hasta que el peatón haya cruzado.

1.5.7. TraNS

Esta herramienta desarrollada en Java, sustenta la unificación del generador de movilidad SUMO con el simulador de redes NS-2. Por medio de la interfaz denominada *Interpreter*, ambos simuladores se interconectan. Las trazas de movilidad de SUMO son transmitidas a NS-2, y de igual forma las instrucciones de NS-2 se envían a SUMO [56].

1.5.8. BonnMotion

BonnMotion es un software escrito en Java que crea y analiza escenarios de movilidad y comúnmente utilizado como herramienta de investigación para redes móviles Ad-Hoc. Los escenarios de movilidad pueden ser exportados a distintos simuladores, tales como NS-2, NS-3, GloMoSim / QualNet, COOJA, MIXIM, y UNO. Estos simuladores cuentan con el apoyo de varios modelos de movilidad, entre ellos [8]:

- The Random Waypoint model (“RandomWaypoint”)
- The Manhattan Grid model (“ManhattanGrid”)
- Gauss-Markov models
- The Reference Point Group Mobility model (“RPGM”)
- Static scenarios (“Static”)
- Static scenarios with drift (“StaticDrift”)
- Disaster Area model (“DisasterArea”)
- Random Street (“RandomStreet”)
- Tactical Indoor Mobility Model (“TIMM”)
- Self-similar Least Action Walk (“SLAW”)

- Map-based Self-similar Least Action Walk (“MSLAW”)
- SMOOTH Model (“SMOOTH”)
- Steady-State Random Waypoint Model (“SteadyStateRandomWaypoint”)
- Random Direction Model (“RandomDirection”)
- Random Walk Model (“RandomWalk”)
- Probabilistic Random Walk Model (“ProbRandomWal”)
- Boundless Simulation Area Model (“Boundless”)
- Column Mobility Model (“Column”)
- Nomadic Community Mobility Model (“Nomadic”)
- Pursue Mobility Model (“Pursue”)
- Chain Model (“ChainScenario”)

CAPÍTULO 2. RED ESPORÁDICA

Las redes Ad-Hoc se ven afectadas directamente por la movilidad de los participantes de la red. En este sentido, la movilidad afecta a la topología de la red y al desempeño de los protocolos de enrutamiento. Por esto es importante definir el concepto de los distintos modelos de movilidad para posteriormente entender como estos afectan a la topología de la red.

Así, en este capítulo se establecen la topología de red y los modelos de movilidad que cumplan con las condiciones de una red esporádica móvil (REM).

En la sección 2.1 se define el concepto de redes sociales esporádicas (RSE) y redes esporádicas móviles, en la sección 2.2 se analiza las topologías de una red Ad-Hoc diferenciando entre ellas a la topología plana y la jerárquica. En la sección 2.3 se analiza los diferentes modelos de movilidad sintéticos tanto de entidad como de grupo, cabe mencionar que los modelos de movilidad de grupo son los que mejor se adaptan al comportamiento humano.

2.1.Redes sociales esporádicas (RSE)

Una RSE es aquella donde un individuo comparte información, documentos, fotos, vídeos, audios, etc. Dicho individuo que difunde tiene la particularidad de elegir exactamente cuánto tiempo estará disponible en la red, una vez llegado ese momento en el tiempo, simplemente desaparece.

Además, existen diversos grupos de investigación, uno de ellos es el Grupo de Servicios para la Sociedad de la Información (Grupo SSI) que trabaja en el desarrollo de la plataforma SPORANGIUM [42] mencionado previamente en el capítulo 1. El cual busca habilitar nuevas formas de interacción social cambiando el paradigma de mundo virtual que las actuales redes sociales ofrecen. La plataforma SPORANGIUM impulsa las RSE al espacio tecnológico de las redes sociales, permitiendo a las RSE apoyarse sobre redes ad-hoc (sin infraestructura) [19].

2.1.1. Red Esporádica Móvil (REM)

Una de las características intrínsecas de las redes MANET es la movilidad de los nodos. Uno de los objetivos de esta tesis es conocer el efecto de esta movilidad en la conformación y mantenimiento de las REM. Al igual que las redes sociales esporádicas, los participantes de una REM poseen la particularidad de entrar y salir de la red en el momento que ellos deseen, teniendo en cuenta que la movilidad afecta a la topología de la red.

2.2.Topologías de red Ad-Hoc

Debido a que la topología de la red puede cambiar rápidamente por el comportamiento dinámico de los usuarios, los algoritmos de enrutamiento se ven afectados. Por esta razón es necesario entender las características de las topologías con las que se puede diseñar las redes MANET y en lo posterior determinar la más adecuada para el proceso de simulación.

Así, se distinguen dos tipos de topologías básicas:

- Topología plana
- Topología jerárquica.

2.2.1. Topología Plana

Es una topología de red plana que no presenta jerarquías y todos los nodos o usuarios que forman parte de la red, presentan un roll de enrutamiento en condiciones iguales. Es tipo de topología funcionan de forma adecuada cuando la red conformada es relativamente pequeña, de tal forma que el enrutamiento se puede considerar óptimo y el consumo de potencia es bajo [58].

El principal inconveniente que presenta este tipo de topología es su escalabilidad, ya que la tabla de enrutamiento es proporcional al número de nodos o usuarios móviles participantes de la red [58] [59]. Entonces, cuando la red se vuelve más grande, las transmisiones requerirán un mayor ancho de banda y el mantenimiento de las tablas de enrutamiento tendrán un efecto considerable en la red, a pesar de ello, se considera necesario el estudio de este efecto en la formación de REM. En la figura 2.1 se muestra el esquema de una topología plana formada por dispositivos móviles.

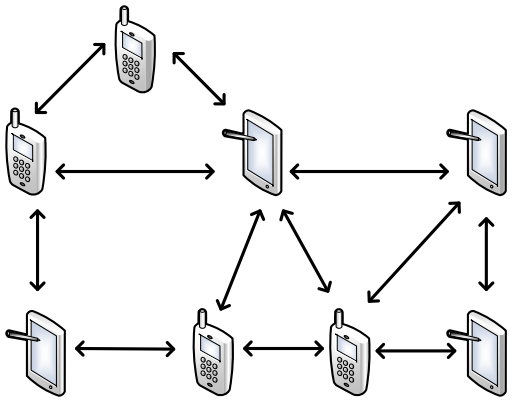


Figura 0.5 Topología de red plana

2.2.2. Topología Jerárquica

Debido a que una topología plana no resulta óptima cuando se pretende manejar un gran número de nodos o usuarios móviles porque la información se transmite por multihop y la cantidad de enlaces es relativamente grande causando un costo mayor del manejo de las tablas de enrutamiento; la alternativa es implementar una topología jerárquica, la cual divide el número total de nodos de la red en conjuntos de nodos en los cuales cada nodo tiene conocimiento de sus vecinos, facilitando la escalabilidad y el enrutamiento. Estos conjuntos o grupos de nodos se los conoce como clusters.e

En MANET, una topología jerárquica reduce significativamente las tablas de enrutamiento, debido a que un nodo solo necesita conocer la información del enrutamiento de su propio cluster, el mismo que es relativamente pequeño [59]. Esta

topología facilita el manejo del tráfico de la red, a un nivel interior llamado intracluster o exterior, llamado intercluster.

En esta topología cada cluster debe elegir a un líder de cluster, el mismo que es conocido como clusterhead. Estos clusterheads son los nodos responsables de transmitir los paquetes entre los demás clusters que conforman la red, en la figura 2.2 se presenta el esquema de una topología jerárquica.

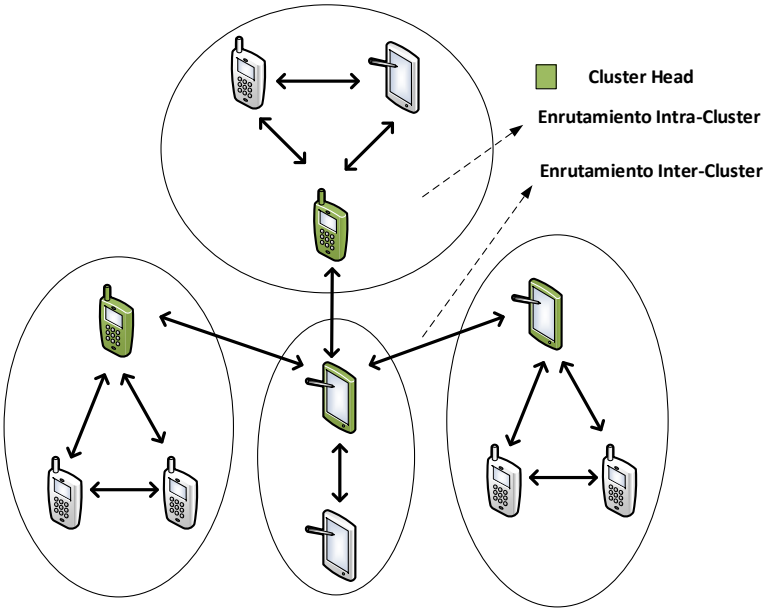


Figura 0.6 Topología de red jerárquica formada por dispositivos móviles

2.2.3. Topología bajo el concepto de virtualización de red

Bajo el concepto de virtualización propuesto por Wu [55], la topología de una REM se describe en la figura 2.3. De esta forma, los nodos más próximos entre ellos emulan un nodo virtual encargado del encaminamiento.

Los problemas de escalabilidad que presenta la topología plana pueden ser mitigados ya que esta topología posee cierto nivel de jerarquía. Por otro lado, en una topología clásica jerárquica existe el inconveniente que cuando se retira o se desconecta el cluster head los demás nodos que forman parte del mismo pueden quedar incomunicados.

En la sección 3.1 se profundiza más sobre el concepto de virtualización y la emulación de los nodos virtuales.

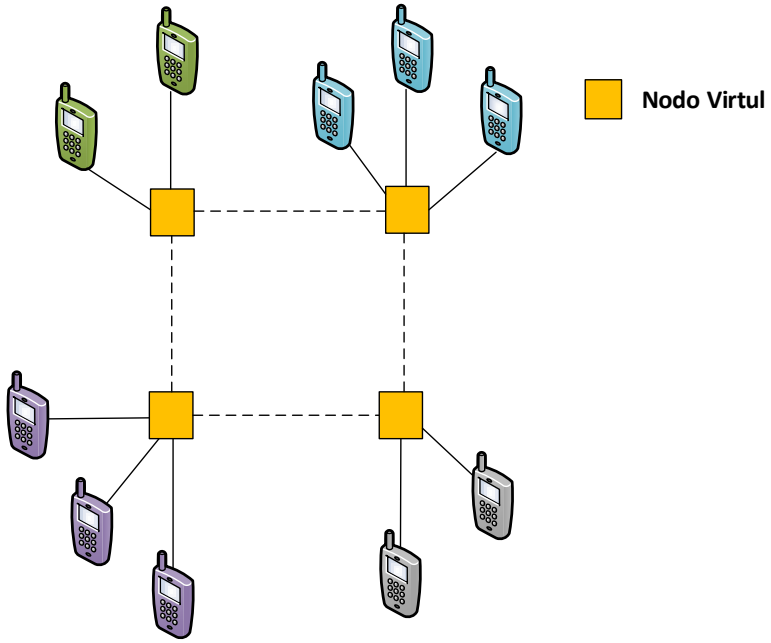


Figura 0.7 Topología de red virtualizada

2.3.Modelos de Movilidad

La movilidad de los nodos participantes en una red Ad-Hoc afecta tanto a la capa física como a la capa de enrutamiento, debido al efecto Doppler y a los repentinos cambios en la topología de la red [60].

Se diferencian dos tipos de modelos de movilidad, los basados en trazas y los sintéticos. Los modelos basados en trazas se apoyan en patrones de movilidad observados en sistemas reales involucrando un alto número de participantes y un largo periodo de observación.

Los modelos sintéticos intentan emular el comportamiento de los usuarios móviles sin el uso de trazas, mediante ecuaciones matemáticas que representan diferentes escenarios de movilidad.

A su vez, los modelos sintéticos se dividen en dos categorías [43]:

- Los modelos de movilidad entidad
- Los modelos de movilidad de grupo

2.3.1. Modelos de movilidad entidad

Esta categoría de modelos de movilidad representan nodos o usuarios móviles cuyos movimientos son independientes entre sí, considerados no realistas ya que se alejan del comportamiento humano [44]. El análisis de este tipo de modelos no será considerado en este trabajo ya que estos modelos representan movimientos de entes opuestos a los movimientos humanos debido a su alta aleatoriedad en dirección, velocidad y aceleración. Sin embargo se los menciona porque los modelos de grupo que son los de interés de este trabajo se basan en los de entidad.

2.3.1.1.Modelo de Movilidad Ramdom Walk

Es un modelo aleatorio, cuyos parámetros de posición, dirección, velocidad y aceleración son escogidos al azar. Es una descripción más simple del movimiento browniano[7], que fue descrito matemáticamente por Einstein en 1926 [52]. Creado con el fin de describir el movimiento de ciertos organismos erráticos. El patrón de movilidad de este modelo no tiene memoria ya que no conserva ninguna información acerca de las ubicaciones anteriores ni los valores de velocidad. Una vez inicializado este modelo puede generar movimientos irreales tales como paradas repentinas y giros bruscos [45].

2.3.1.2.Modelo de Movilidad Waypoint Ramdom Walk

[7] El movimiento browniano es un modelo estadístico que Einstein desarrollo para describir el comportamiento aleatorio de ciertas partículas inmersas en un fluido debido a la interacción térmica entre las moléculas del fluido y las partículas [52].

Basado en Ramdom Walk, una vez inicializado, el nodo en la posición P0 con coordenadas (x0 , y0) se desplaza por un periodo de tiempo T1, a una velocidad V1 y ángulo de desplazamiento θ_1, cuando este se ha desplazado, se realiza una pausa en el punto P1 y se calcula la velocidad V2 y ángulo de desplazamiento θ_2 (entre 0 y 180 grados) para así llegar a la posición P2 con coordenadas (x2 , y2). En P2 se vuelve a realizar una pausa para luego realizar un nuevo cálculo de posición y de velocidad, continuando el proceso en cada nueva posición del nodo móvil [44]. En la figura 2.4 se sintetiza el modelo descrito.

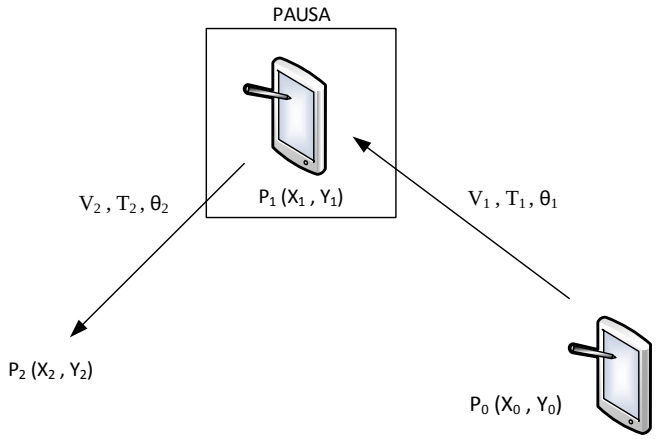


Figura 0.8 Representación de un usuario móvil Waypoint Ramdom Walk

2.3.1.3.Modelo de Movilidad Gauss-Markov

Mediante un parámetro de refinamiento para variar su grado de aleatoriedad (tuning parameter), este modelo asigna una velocidad y dirección a cada nodo, luego de cierto intervalo de tiempo fijo "t" el movimiento se produce mediante la actualización de la velocidad y dirección de cada nodo. Así el siguiente movimiento toma en cuenta sus valores anteriores (t-1) y una variable gaussiana [46].

Para modelar este patrón de movimiento se toma en cuenta la velocidad y la dirección del nodo en un tiempo "t" a través de las siguientes ecuaciones:

- $S_t = \alpha S_{t-1} + (1-\alpha)\bar{S} + \sqrt{1-\alpha^2}\, S_{x_{t-1}}$

Donde:

S_t = Velocidad del nodo en un tiempo "t"

α = Variación de aleatoriedad ($0 \leq \alpha \geq 1$)

S_{t-1} = Velocidad del nodo en un tiempo "$t-1$"

$\bar{S}$ = Velocidad media cuando $t \rightarrow \infty$

$S_{x_{t-1}}$ = Variable aleatoria gaussiana

- $d\_dev_t = \alpha d\_dev_{t-1} + (1-\alpha)\overline{d\_dev_t} + \sqrt{1-\alpha^2}\, d\_dev_{x_{t-1}}$

  Donde:

  d_dev_t = Dirección del nodo en un tiempo "t"

  α = Variación de aleatoriedad ($0 \leq \alpha \geq 1$)

  d_dev_{t-1} = Dirección del nodo en un tiempo "$t-1$"

  $\overline{d\_dev_t}$ = Dirección media cuando $t \rightarrow \infty$

  $d\_dev_{x_{t-1}}$ = Variable aleatoria gaussiana

2.3.1.4.Modelo de Movilidad Smooth random [44]

Por medio de este modelo se intenta que el movimiento de los nodos se menos brusco. Permite controlar los cambios repentinos de velocidad en el instante que se realiza un cambio de dirección. La velocidad y el cambio de dirección son probabilísticos respondiendo a un Proceso de Poisson.

En este modelo, los nodos participantes pueden desplazarse a cualquier lugar en el plano de simulación y no existe una correlación entre los diferentes nodos, es decir, efectos como "nodo siguiente" o "movimiento de los grupos" no están modelados. Además utilizan dos procesos estocásticos: un proceso determina en qué momento

una estación móvil cambia su velocidad y el otro proceso determina cuándo se cambia la dirección.

Cuando se inicializa la simulación, un nodo móvil parte de la posición P0 (x0 , y0) con una velocidad V1 y una dirección θ_1 hasta llegar a la posición P1 (x1 , y1) en un tiempo T1 con una velocidad V2, mediante un proceso estocástico el nodo cambia su dirección hasta llegar a la posición P2 (x2 , y2) con una velocidad V3 en un tiempo T2. El proceso se repite hasta que la simulación termine. En la figura 2.5 se sintetiza el modelo descrito.

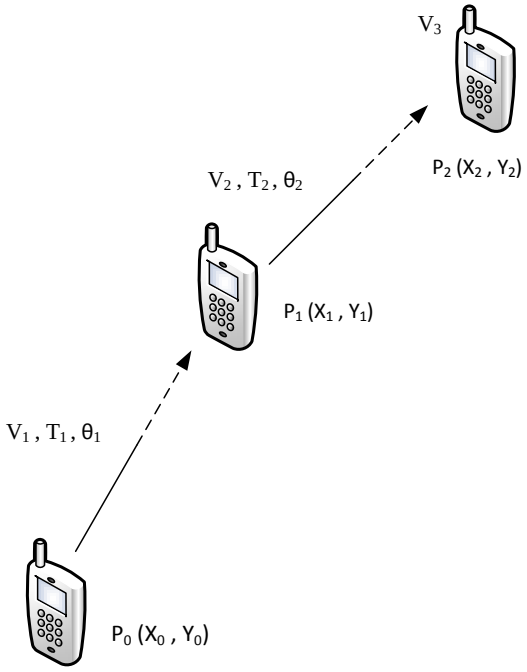


Figura 0.9 Representación del movimiento en el modelo movilidad Smooth Random

2.3.2. Modelos de movilidad de grupo

Este modelo de movilidad representa nodos o usuarios móviles cuyos movimientos dependen unos de otros, considerados realistas ya que se asemejan al comportamiento de los seres humanos [44]. Resulta importante analizar estos modelos de movilidad ya que los cambios en la topología de red en la conformación de las REM's se relacionan con estos patrones de movimiento.

2.3.2.1.Modelo de Movilidad de Columna

El Modelo de movilidad de Columna, representa un conjunto de nodos que se mueven alrededor de una línea dada (o columna), que se está moviendo en cierta dirección (siguen a un nodo de referencia, por ejemplo, una fila de soldados que marchan juntos hacia su enemigo, o un grupo de robots que desactivan bombas).

La aplicación de este modelo de movilidad se puede extender para el estudio de redes VANET, en la que los nodos móviles se desplazan por autopistas o carreteras de alta velocidad.

Para la implementación de este modelo, se define una cuadricula inicial de referencia. Cada nodo se coloca a continuación en relación con su punto de referencia en la cuadricula.

Dentro de la cuadricula, se le permite al nodo moverse al azar en torno a su punto de referencia a través de un modelo de movilidad, al nodo "líder" se le asigna una "orbita de referencia" que permite al resto de nodos seguir a este nodo principal llamado de referencia. En la figura 2.6 se representa el modelo de movilidad de columna [47].

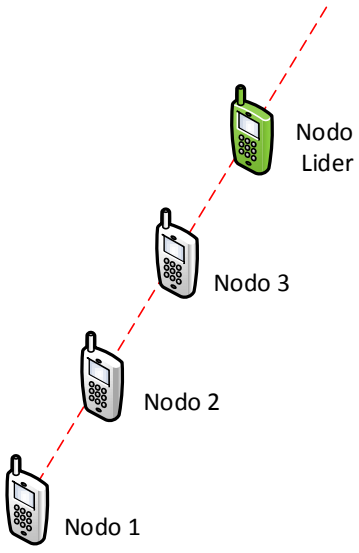


Figura 0.10 Modelo de Movilidad Columna

2.3.2.2.Modelo de Movilidad de Comunidad Nómada [47]

Este modelo representa grupos de nodos que colectivamente se mueven de un punto a otro. Dentro de cada comunidad o grupo de nodos, los individuos mantienen su propia área en donde se mueven de manera aleatoria. Existen numerosas aplicaciones para este tipo de escenario. Por ejemplo, grupo de estudiantes recorriendo un museo de arte o un pabellón de exhibición, los estudiantes se moverán juntos de un lugar a otro, sin embargo, los estudiantes podrían dar vueltas por un lugar determinado de forma individual; otro ejemplo puede ser en el área militar donde se mueve un grupo hasta cierta localización y luego tienen que dispersarse por un tiempo determinado.

En este modelo, cada nodo utiliza un modelo de movilidad para vagar alrededor de un punto de referencia o nodo líder. Cuando se cambia el punto de referencia, todos los nodos en el grupo se mueven a la nueva área definida por el nodo líder. En comparación con el Modelo de Movilidad de Columna, los nodos comparten un punto común de referencia frente a un punto de referencia individual en una columna. Es decir que, los nodos persiguen a un nodo de referencia o nodo líder pero manteniendo su espacio personal donde se mueven de forma aleatoria. En la figura 2.7 se sintetiza el patrón del modelo de movilidad de comunidad nómada.

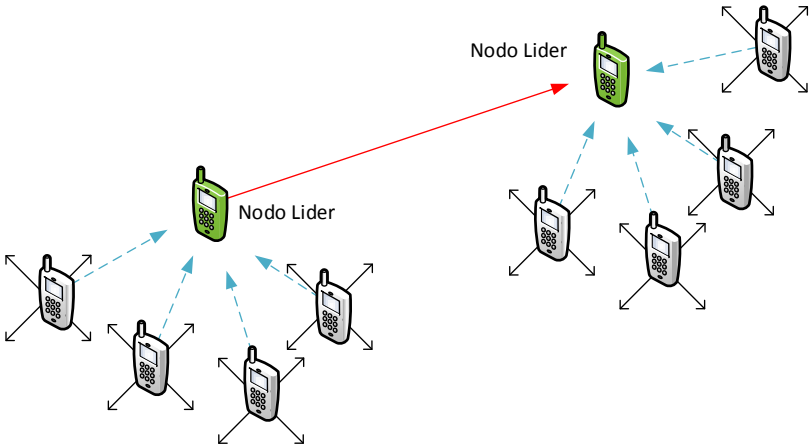


Figura 0.11 Modelo de Movilidad de Comunidad Nómada

2.3.2.3.Modelo de Movilidad de Persecución [47]

Como su nombre lo indica, el Modelo de Movilidad de persecución intenta representar los nodos en seguimiento de un objetivo particular. Por ejemplo, este modelo podría representar policías intentando atrapar a un criminal que ha escapado. El modelo de persecución describe escenarios donde varios nodos intentan capturar un nodo móvil o nodo destino.

En este modelo se considera que un nodo perseguido se mueve libremente de acuerdo con el modelo Random Waypoint Walk de tal modo que los nodos perseguidores intentan interceptar al nodo destino. En la figura 2.8 se describe el modelo de persecución. Este modelo se describe mediante la siguiente ecuación [50]:

- $P_i^t = P_i^{t-1} + v_i^t\left(P_{target}^t - P_i^{t-1}\right) + w_i^t$

  Donde:

  P_{target}^t = Poscicion esperada del nodo objetivo en un tiempo t

  w_i^t = Vector aleatorio utilizado para compensar el movimiento del nodo móvil i

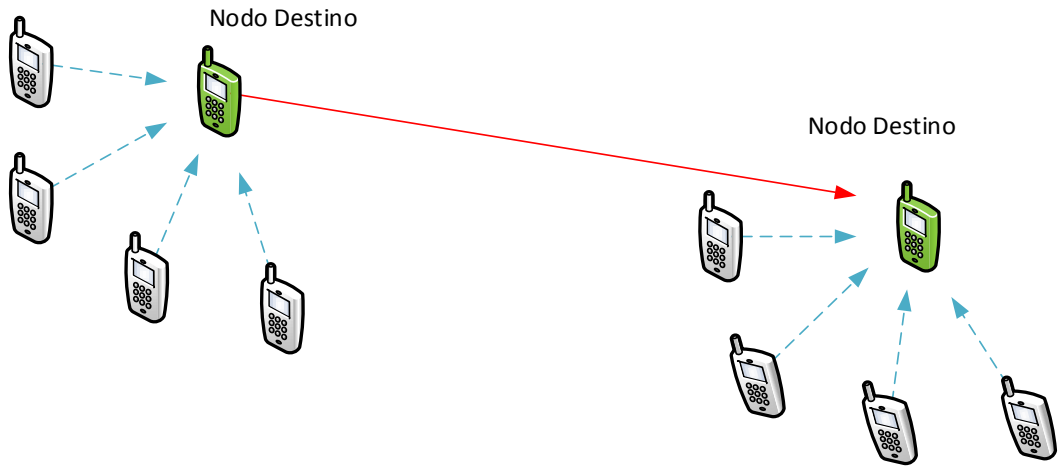


Figura 0.12 Modelo de Movilidad de Persecución

2.3.2.4.Modelo de Movilidad de grupo de punto de Referencia

(RPGM) [48]

El modelo conocido como RPGM representa el movimiento aleatorio de un grupo de nodos así como el movimiento aleatorio de cada nodo individual dentro del grupo. Los movimientos del grupo se basan en el camino recorrido por un centro lógico, el centro lógico del grupo se utiliza para calcular el movimiento de grupo a través de un vector de movimiento. El movimiento del centro del grupo caracteriza su dirección y velocidad. Los nodos individuales se mueven aleatoriamente cerca de sus propios puntos de referencia predefinidos, cuyos movimientos dependen del movimiento del grupo.

A diferencia del modelo de comunidad nómada, este modelo utiliza el centro lógico del grupo para calcular la dirección y la velocidad de cada nodo, en cambio el modelo de comunidad nómada cada nodo puede tener su propio modelo de movilidad.

En este modelo, cada grupo está compuesto por un líder y un número de miembros. El movimiento de la líder del grupo determina el comportamiento de la movilidad de todo el grupo [50].

El modelo RPGM puede adaptarse a diversos escenarios de movilidad como por ejemplo en operaciones de rescate de, bomberos, policías, asistentes médicos o un pelotón de soldados, etc. En la figura 2.9 presenta el modelo de movilidad de grupo de punto de Referencia.

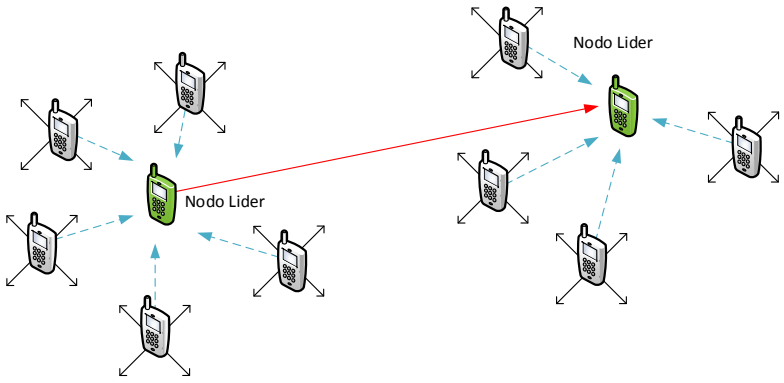


Figura 0.13 Modelo de Movilidad de Grupo de Punto de Referencia

2.3.2.5.Modelo de Movilidad Ad-Hoc basados en teoría de Redes Sociales

Existen también propuestas de modelos de movilidad que están basados en teoría de redes sociales, esto debido a que los seres humanos realmente no tienen movimientos aleatorios como indica el movimiento browniano [52].

Debido a que en la actualidad un gran número que las personas interactúan o se relacionan con otras a través de redes sociales, que en casos pueden ser grupos grandes o pequeños, y de la misma manera una persona se puede movilizar con un grupo de personas o cambiar de grupos de movilidad. Entonces el modelo toma en cuenta la información social de los movimientos reales y sintetizados de personas tal como se ve en [49] donde se genera un software que puede recrear el movimiento de las personas que interactúan.

Por lo tanto, al tener en cuenta este movimiento de las personas es posible determinar cómo afecta esto al desempeño de los sistemas de comunicación. Además de la interacción de personas se debe tomar en cuenta el ambiente en el que se mueven las personas ya que esto puede afectar el desempeño de una red de manera significativa. Sin embargo en [51] se explica que en escala micro los usuarios no afectan de manera significativa a la red, pero se ve la necesidad de definir un amplio banco de huellas de referencia para la evaluación de la movilidad en diferentes escenarios en lugar de tratar de obtener un modelo que sea compatible para todo tipo de movilidad de los peatones.

2.3.2.6.Modelos de Movilidad en REM’s

La relación que existe entre los modelos de movilidad y las personas es parte importante para el desarrollo de este trabajo. Por esta razón esta sección presenta un breve análisis de los modelos de las REM´s antes descritos.

Entonces al hacer una recapitulación, se puede decir que en diversos escenarios (museos, cafeterías, coliseos, conciertos, rutas turísticas, etc.) en los que pueden ser

aplicadas REM´s encontramos que la interacción de las personas también se ve afectado dependiendo del lugar o situación en la que se encuentran las personas.

Por otra parte los modelos de movilidad que mejor se adaptan al comportamiento de grupo de personas son: Comunidad Nómada y el modelo RPGM.

Sin embargo, aunque ambos poseen características similares, la principal diferencia radica en que el modelo RPGM, los nodos individuales se mueven dependiendo de un centro lógico caracterizado por su nodo líder, mientras que en el modelo de comunidad nómada, los nodos individuales pueden moverse de forma arbitraria pero manteniendo una relación de posición con respecto a su nodo líder.

Estas características destacan a estos modelos de movilidad como los candidatos sobre los cuales se realizaran los análisis de los protocolos de enrutamiento en la conformación de REM's.

En la figura 2.10 se muestra una representación de un entorno físico posible. El propósito es llegar a simular ambientes similares en los cuales las personar puedan interactuar entre sí por medio de REM's.

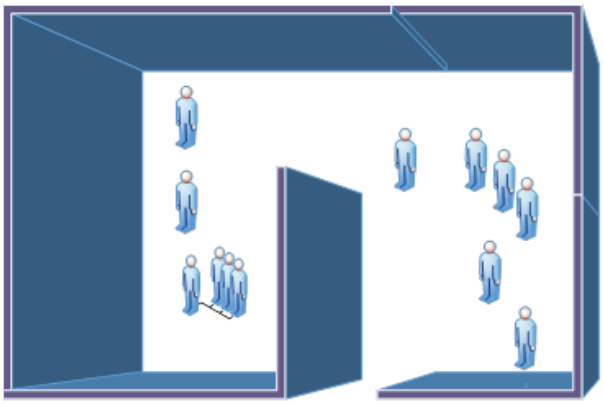


Figura 0.14 Escenario Ficticio para pruebas de REM's

CAPÍTULO 3. COMPOSICIÓN DEL SIMULADOR

El problema de encaminamiento presentado por las redes MANET puede ser solventado a través de diverso enfoques. El análisis del concepto de virtualización por medio del uso de la capa del nodo virtual (VNLayer) es uno de los métodos [4] [55]. Esto es posible al proporcionar mecanismos escalables en los cuales los dispositivos móviles pueden apoyar las comunicaciones por medio de múltiples saltos [19].

Por tanto, en la sección 3.1 se expondrá los conceptos de VNLayer y todo lo referente al funcionamiento lógico de esta capa. Además, en la sección 3.2 se expondrán los conceptos necesarios para la creación de las trazas de movilidad en el simulador BonnMotion.

3.1. VNLayer (Capa de Nodo Virtual)

La VNLayer se concibe como una capa abstracta para MANET donde se definen procedimientos para nodos físicos móviles (llamado nodos clientes cuyos movimientos no pueden ser predecibles) los mismos que están encargados de emular nodos virtuales (nodos predecibles) los cuales pueden ser abordados como servidores estáticos virtuales [53].

El trabajo realizado por HURTADO-SIGUENZA [4] define los beneficios y limitaciones que posee la VNLayer, a continuación se desarrolla una ampliación del análisis de capa virtual VNLayer.

Según [4] "*un nodo virtual –nodo abstracto- es creado por la capa Virtual y es emulado por varios nodos físicos. El nodo virtual permite desarrollar aplicaciones como si hubiese un servidor en una localización geográfica fija*."

Geográficamente, VNLayer divide el área donde se establece la MANET en rejillas (regiones o cuadrados) de forma que los nodos virtuales puedan ser implementados. Es necesario tomar en cuenta que es necesario de nodos físicos para dar soporte a los nodos virtuales [3].

Entonces, el nodo virtual de cada región es emulado por los nodos físicos de la misma. En cada región, se elige un nodo físico que hace la función de líder, el mismo que se encarga del establecimiento de rutas de comunicación con los otros nodos virtuales

De esta manera, en la figura 3.1 se muestro un escenario dividido en 4 regiones en los cuales se establecen los nodos líderes y los nodos virtuales. La VNLayer se encarga de asignar el nodo físico líder. En la realidad los nodos se encuentran en movimiento y responden a patrones de movilidad tales como el RPGM.

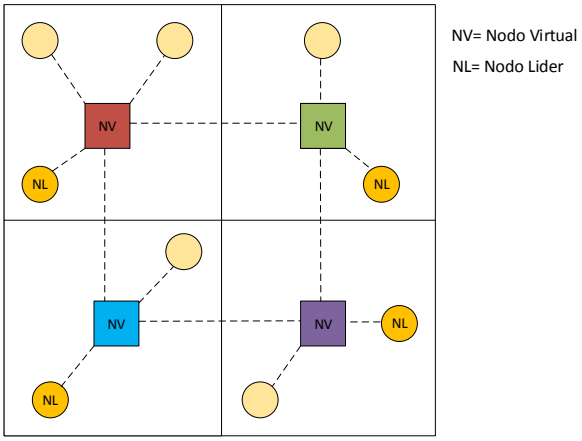


Figura 0.15Representación de la VNLayer en un escenario de 4 regiones

3.1.1. Virtualizando la red (VNLayer en funcionamiento)

El propósito de VNLayer es mitigar el constante cambio de rutas que se producen en una MANET, conllevando a una disminución del tráfico producido por el protocolo de encaminamiento.

De acuerdo con esto y sabiendo que una red inalámbrica de corto alcance, los nodos o dispositivos móviles se encuentran en continuo movimiento, (algo que no sucede

en un sistema de comunicación con medios guiados o de área extensa), se plantea el caso de la figura 3.2. En este caso se plantea que el nodo 1 desea comunicarse con el nodo 10, donde interviene los nodos 2-3-6-7-8-9 para esta comunicación. Además, para este mismo ejemplo y bajo el supuesto de que el nodo 6 salga o se desconecte de la red, la ruta establecida queda inutilizada.

Entonces, para que la comunicación se vuelva a establecer es necesario un Nuevo descubrimiento de ruta por parte del protocolo de encaminamiento, lo cual supone una sobrecarga de tráfico en la red.

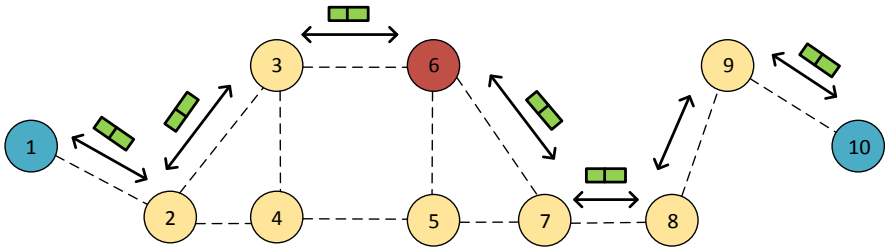


Figura 0.16 Comunicación en una red inalámbrica entre el nodo 1 y el nodo 10 una vez establecido el proceso de descubrimiento de ruta

Por su parte, la VNLayer posee la capacidad de crear nodos virtuales facilitando el encaminamiento. En la figura 4.3 se muestra la misma topología de red, la cual ha sido virtualizada. Del mismo modo la comunicación es entre el nodo 1 y 10 pero esta vez se hace uso de los nodos virtuales, en caso de que el nodo 6 salga de la red la comunicación seguirá existiendo, incluso si la conexión del nodo 5 se extinguiera la comunicación seguiría establecida ya que el nodo 7 es capaz de emular un nodo virtual en su región. Por tanto, solo cuando no exista ningún nodo en la región la ruta quedara inactiva perdiéndose la comunicación.

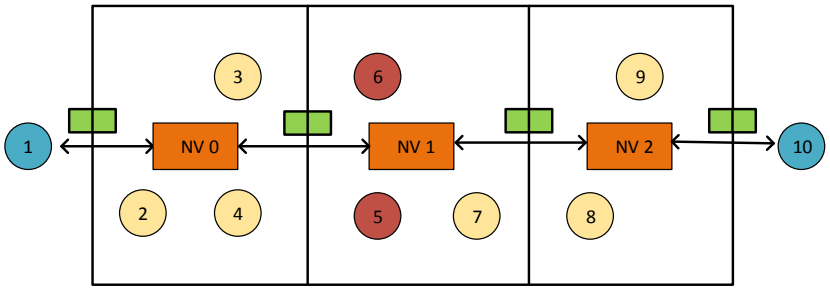


Figura 0.17 Emulación de nodos virtuales por medio de la VNLayer. Los nodos 2-3-4 emulan el nodo virtual 0, los nodos 5-6-7 emulan el nodo virtual 1 y los nodos 8-9 emulan el nodo virtual 2. La ruta establecida es nodo 1- NV0- NV1- NV2- nodo 10

3.1.2. Lógica de la VNLayer

La VNLayer está concebida para que exista un nodo virtual por cada región, siempre y cuando exista por lo menos un nodo físico en dicha región. En este caso, en cada región se asigna a un nodo físico como nodo "Líder" que es encargado de participar de los procesos de descubrimiento de ruta y reenvío de mensajes.

Los nodos "No-líder" que forman parte de una región, pueden ser asignados como "Backup". En caso de que el nodo Líder se desconecte o abandone la región, el Backup asumirá el roll de Líder [4].

3.1.2.1.Elección del Líder [4]

En el proceso de elección de líder, se diferencian 5 estados:

- INITIAL: estado inicial donde los nodos físicos no conocen el estado de su región.
- REQUEST: estado en el que los nodos físicos han determinado su región de pertenencia.
- LEADER: estado en el cual los nodos de una región determinada eligen cuál de ellos asumirá el roll de líder.
- NONLEADER: estado en el cual por alguna razón cualquiera la región se ha quedado sin un líder.
- UNSTABLE: estado intermedio que sirve de apoyo al estado NONLEADER evitando bucles y duplicación de nodos líderes.

Además, hay dos temporizadores que intervienen en el proceso de elección: TimerRequesWait (controla el tiempo que un nodo espera antes de decidir que no

escuchar los mensajes del nodo líder) y TimerHeartbeatWait (utilizado por nodos líderes y no líderes para controlar para reafirmar el liderazgo en una determinada región).

Los mensajes enviados por los nodos para la elección del líder son:

- LeaderRequest: utilizado para solicitar el liderazgo de la región.
- HeartBeat: utilizado por el líder el cual periódicamente reclama su liderazgo.
- LeaderLeft: utilizado por el líder para anunciar que está dejando la región.
- LeaderReply: enviado y utilizado por el líder para rechazar una solicitud de liderato.

Cuando un nodo empieza a participar en una REM, no conoce su región (estado INITIAL). Cuando se determina su región (estado REQUEST), cada nodo configura un TimerRequestWait. Si el tiempo de TimerRequestWait expira (estado LEADER), el nodo envía un mensaje de Heartbeat y coloca su región actual de manera que, únicamente los nodos que pertenezcan a la misma región escucharan el mensaje. Por otro lado cuando un nodo en estado REQUEST escucha un Heartbeat (estado NONLEADER) este lee el mensaje para determinar su líder [53].

Cuando un nodo no líder, dentro un de un intervalo de tiempo no escucha un Heartbeat, asume que el nodo líder abandonó la región y pasa a estado REQUEST. Por esto, de forma periódica el nodo líder de la región envía mensajes Heartbeat confirmando su liderazgo. De esta manera, cada vez que un nodo no líder escucha un Heartbeat se restablece el tiempo de espera (TimerHeartBeatWait) de modo que el periodo de tiempo en enviar un Heartbeat es menor al periodo de tiempo de espera de un no líder [4] [53].

Considerando que un nodo en estado REQUEST está en la misma región que un nodo líder, se puede dar el caso de que el nodo no escuche el Heartbeat causando un líder duplicado, y existan dos líderes en una misma región. Este problema se solventa por medio del campo Time_state que se encarga de almacenar los tiempos que se dan en cada transición de estados. Entonces, cuando el líder duplicado escuche un Heartbeat enviado por otro líder, enviará a leer su campo Time_state y dado que tal valor es menor a su valor almacenado de transición a líder, automáticamente cambiara a estado no líder [4].

También se puede dar el caso en que un nodo en estado NONLIDER no escuche el mensaje Heartbeat del líder provocando que el nodo pase a estado REQUEST. Para ello, el nodo de estado NONLEADER pasa a estado UNSTABLE y esperando a escuchar nuevamente un Heartbeat. Máximo se permite 2 Heartbeat perdidos antes de pasar a estado REQUEST.

Los mensajes pueden ser enviados de manera unicast, dando la posibilidad a la capa MAC del nodo destino confirmar su recepción, si el nodo origen no recibe una confirmación entonces reenvía el mensaje. Existe un mensaje del llamado LeaderElection enviado a madera de broadcast, por los nodos no líderes en ausencia de mensajes de HeartBeat.

Otro aspecto a considerar es el caso en el que el líder abandona la región lo cual puede causar latencias o perdida de paquetes. La región que tiene ausencia de líder debe ser capaz de asignar un nuevo líder en el menor tiempo procurando un buen rendimiento de la red. Por esto, cuando un líder abandona la región, envía un mensaje de LeaderLeft alertando a los nodos de la región para iniciar el proceso de elección de líder y los nodos no líderes cambian a estado REQUEST. Cuando un nodo cambia de región, cambia a estado REQUEST enviado un mensaje LeaderRequest, pero si la región ya está liderada el líder actual responderá con un mensaje LeaderReply de tal forma que el nodo cambiará de estado REQUEST a NONLEADER.

En la figura 3.4 se describe la máquina de estados del procedimiento gestión de líder dentro de la VNLayer.

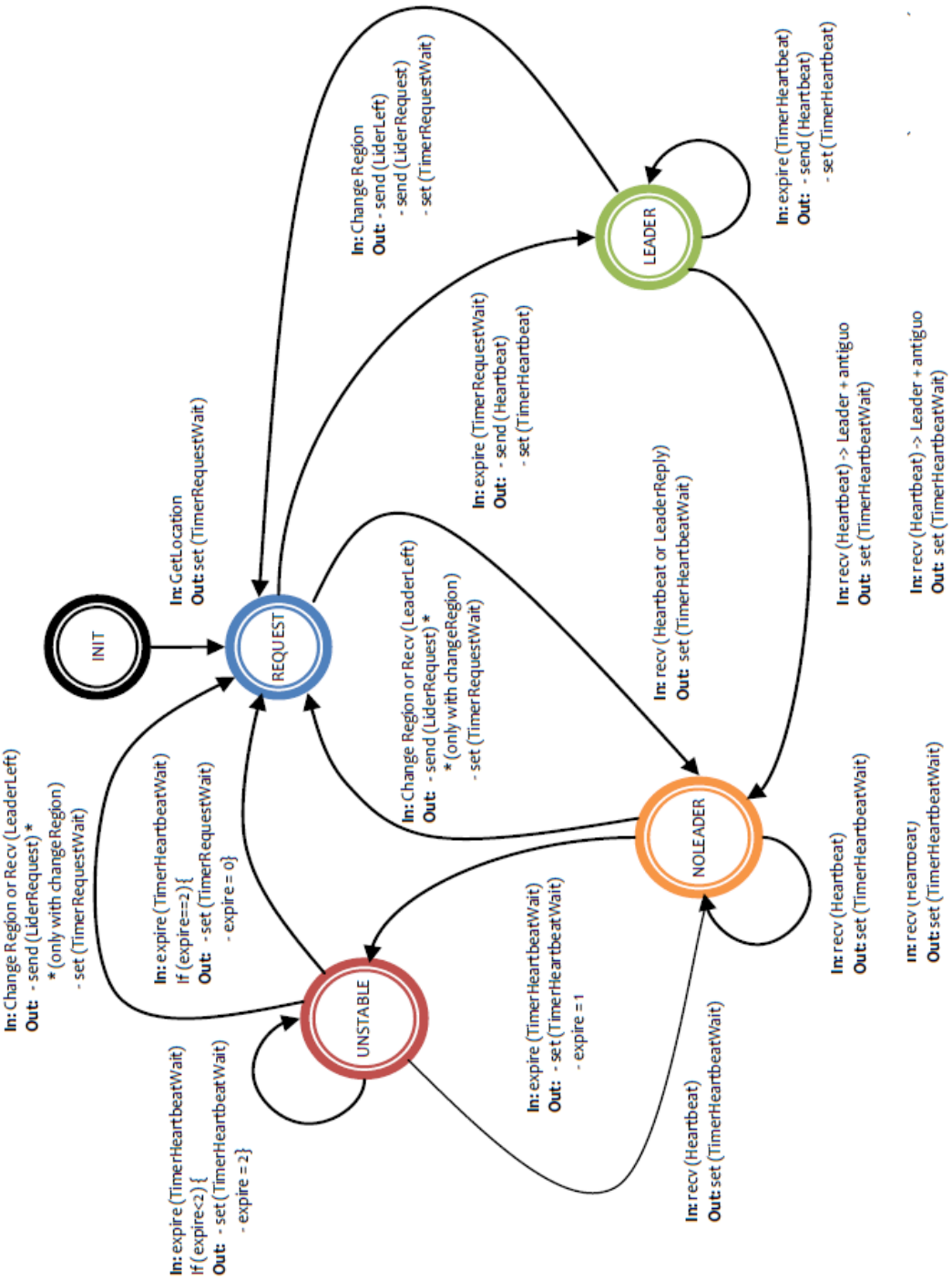


Figura 0.18 Máquina de estados del procedimiento gestión de líder dentro de la VNLayer [4]

A continuación se describe el pseudocódigo que sigue la VNLayer para la elección de líder de cada región.

```
INITIAL:
 [enter now region]
  set TimerRequestWait
  goto REQUEST
REQUEST:
 [expiere TimerRequestWait]
  send HeartBeat
  set TimerHeartBeat
  goto LEADER
 [recive HeartBeat or LeaderReply]
  set TimerHeartBeatWait
  goto NONLEADER
LEADER:
 [change region]
  send LiderLeft
  send LiederRequest
  set TimerRequestWait
  goto REQUEST
 [recive HeartBeat]
  set TimerHeartBeatWait
  goto NONLEADER
 [expire TimerHeartBeat]
  send HeartBeat
  set TimerHeartBeat
  goto LEADER
NONLEADER
 [change region or recive LeaderLeft]
  send LiderRequest only with change region
  set TimerRequestWait
  goto REQUEST
 [recive HeartBeat]
  set TimerHeartBeatWait
  goto NONLEADER
```

```
 [expire TimerHeartBeatWait]
  set TimerHeartBeatWait
  expire = 1
  goto UNSTBLE
UNSTABLE
 [change region or recive LeaderLeft]
  send LiderRequest only with change region
  set TimerRequestWait
  goto REQUEST
 [recive HeartBeat]
  set TimerHeartBeatWait
  goto NONLEADER
 [expire TimerHeartBeatWait  if expire < 2]
  set TimerHeartBeatWait
  expire = 2
  goto UNSTABLE
 [expire TimerHeartBeatWait  if expire == 2]
  set TimerRequestWait
  expire = 0
  goto REQUEST
```

3.1.2.2. Nodos de Backup y estado de sincronización

Entre los nodos no líderes, se eligen a los nodos de respaldo o Backup, el mismo que debe estar sincronizado con el nodo líder, los de respaldo escuchan y procesan los mismos paquetes como un líder en la región [4] [53]. Estos nodos sirven de apoyo inmediato en caso de que el líder actual abandone la región, de esta forma asumirá el roll de líder evitando que el nodo virtual caiga y se de una perdida de paquetes.

Tan pronto como suceda la elección de nodo líder, la elección de nodos de respaldo o Backups iniciará. Así, los mensajes del nodo líder llevan el número actual de nodos Backup de su región y el nodo no líder compara este con el máximo de backups de su región y si es menor inicia un TimerRequestBakup. Si el temporizador expira envía un mensaje SynRequest solicitando ser nodo de Backup. En este caso el líder

responde con un SynData (enviado de manera unicast) el cual lleva los datos de sincronización [4].

Para evitar problemas con el proceso de sincronización debido a colisiones de mensajes SynData, se utiliza una tabla de Backups y mensajes SynAck. Cuando el nodo líder recibe un SynRequest coloca en su tabla al nodo que envió la solicitud de sincronización y lo asigna como no sincronizado. Entonces, el nodo líder envía mensaje SynData al nodo solicitante para sincronizarlo. Este se responde con un SynAck y el nodo líder al recibir el SynAck establece al nodo como sincronizado [4] [53].

Para el proceso de sincronización de Backups, se asigna una prioridad entre los nodos Backups sincronizados. De esta forma cuando un nodo no líder sea el primero en asignarse como Backup tomará la posta entre los demás nodos Backups para asumir el roll de líder. Cuando el nodo líder salga de la región, el nodo de Backup con prioridad 0 asumirá el roll de nuevo líder. En la figura 3.5 se describe el efecto de las prioridades entre nodos Backups.

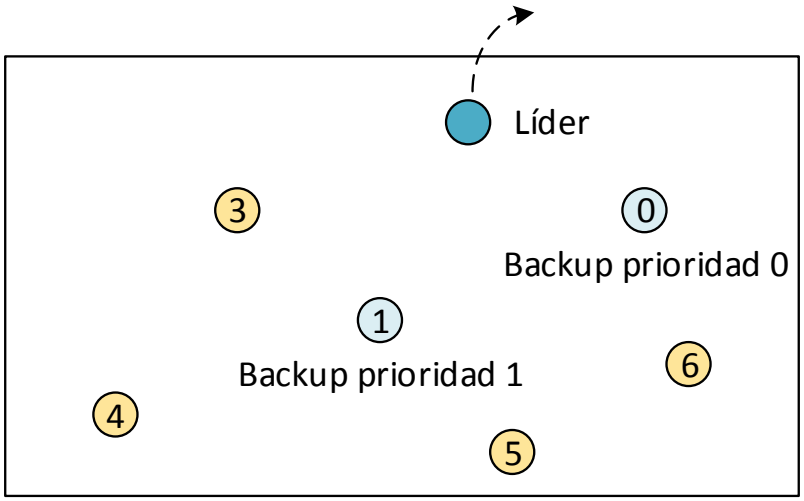


Figura 0.19 Descripción del proceso de elección de nodos Backups y el efecto de la asignación de prioridades

3.1.2.3.Alcance y limitaciones de VNLayer

VNLayer crea un sistema de agrupación de nodos físicos estableciendo un nivel de jerarquía, por lo tanto, se reduce el número de nodos a ser manejados lo que conlleva a una reducción de carga y de tráfico en la red.

Por otra parte, dado que la VNLayer establece nodos virtuales estáticos, se vuelve predecible y estable el movimiento de los nodos físicos en lugar de tratar con un conjunto de nodos cuyo comportamiento dentro de la red es impredecible.

La VNLayer, por sus características intrínsecas posee ciertas limitaciones, como un alto costo de implementación y establecimiento ya que los nodos o dispositivos físicos necesitan de un GPS para localizar su ubicación.

Es importante que la VNLayer sea diseñada de forma cuidadosa, manteniendo bajos niveles de tráfico de mensajes, ya que se requieren de mensajes de control y sincronización entre los nodos líderes y los no líderes.

3.2.Estructura de las trazas de movilidad

La herramienta de generación de trazas de movilidad (BonnMotion) está desarrollada de forma simple e intuitiva facilitando la ejecución de los modelos de movilidad a implementar.

De acuerdo con los modelos de movilidad analizados en el capítulo 2, el modelo RPGM se considera como el que más se adapta a la conducta de las personas dentro del grupo dado que los participantes del grupo se mueven de acuerdo a un centro lógico común.
A continuación se describen los parámetros de entrada que el modelo RPGM necesita para ser implementado dentro de BonnMotion.

3.2.1. Desarrollo del patrón de movilidad

La generación del escenario de movilidad (modelo RPGM) mediante BonnMotion se rige por ciertas características o variables de entrada. Permite modificar y establecer parámetros del escenario (área) en el cual se moverán los nodos, ajustar las variables de velocidad a las que se regirán los nodos y datos propios del modelo de movilidad aplicado.

A continuación se describe las variables de entrada que el modelo de movilidad necesita:

3.2.1.1.Escenario de movilidad

El escenario de movilidad es el área en la cual los nodos se van a desplazar, BonnMotion crea escenarios de prueba de forma cuadrada o rectangular. En la tabla 3.1 se describe los parámetros de entrada para recrear los escenarios de movilidad que BonnMotion presenta a sus usuarios.

Parámetro	Descripción
-d	scenario duration [s]
-i	number of seconds to skip [s]
-n	number of nodes
-x	width of simulation area [m]
-y	height of simulation area [m]
-R	random seed

Tabla 0.2 Parametros de entrada para la generacion del escenario

- El parámetro ‘d’ describe la duración (tiempo) del escenario de movilidad, se debe tomar en cuenta que el parámetro ingresado es en segundos.
- El parámetro ‘i’ estandarizado en 3600 segundos.
- El parámetro ‘n’ es el número de nodos que participaran en el modelo de movilidad.
- Los parámetros ‘x’ ‘y’ describen respectivamente el ancho y el alto del escenario de movilidad, dando con esto un área en la cual se mueven los nodos participantes.
- El parámetro ‘R’ describe una variable de tipo randomica para la distribución de los distintos parámetros aplicados a los nodos.

3.2.1.2.Velocidad

BonnMotion permite ajustar los valores de velocidad que los nodos pueden llegar a tener, incluso integra una variable de pausa máxima en el cual los nodos permanecen de forma estática. En la tabla 3.2 se describe los parámetros de entrada para recrear otorgarle velocidad máxima y mínima a los nodos.

Parámetro	Descripción
-h	max. speed [m/s]
-c	min. speed [m/s]
-d	max. pause time [s]

Tabla 0.3 Parametros de entrada para la velocidad de los nodos

- Los parámetros ‘h’ ‘l’ respectivamente describen la máxima y la mínima velocidad que pueden tener los nodos al desplazarse por el escenario de simulación.

- El parámetro ‘p’ describe la máxima pausa que un nodo puede llegar a tener para ir a la siguiente traza de movilidad.

3.2.1.3.RPGM

El modelo RPGM ya se encuentra compilado dentro de BonnMotion de forma que solo es necesario ingresar ciertos parámetros de entrada. En la tabla 3.3 se describe los parámetros de entrada necesarios para imprimirle el modelo RPGM a los nodos.

Parámetro	Descripción
-a	average no. of nodes per group
-c	group change probability
-r	max. distance to group center [m]
-s	group size standard deviation

Tabla 0.4 Parametros de entrada intrisecos al modelos RPGM

- El parámetro ‘a’ describe el promedio del número de nodos que formaran parte de los diferentes grupos formados por el modelo RPGM.
- El parámetro ‘c’ describe la probabilidad de que se produzca un cambio de grupo, en otras palabras, la probabilidad de que un nodo pase a integrar otro grupo distinto a su inicial.
- El parámetro ‘r’ describe la distancia máxima a la cual los nodos satélites se mueven de su punto de referencia o nodo líder. El parámetro es ingresado en metros.
- El parámetro ‘s’ describe una desviación estándar para el tamaño del grupo, tomando en cuenta el número de nodos participantes

Los análisis posteriores se realizaran utilizando el modelo RPGM considerando ciertas cualidades de la conducta humana como lo es la velocidad de desplazamiento y la distancia social [61].

CAPÍTULO 4. IMPLEMENTACIÓN

Las trazas de movilidad son importadas de BonnMotion hacia NS3 dándole a los nodos la movilidad necesaria y estableciendo el escenario de movilidad a analizar. Además NS3 es el encargado de añadirle las funciones de red a los nodos móviles.

En este capítulo se explicará la estructura de la implementación de las trazas de movilidad sobre NS3, la interfaz para la conjunción de estos dos simuladores y el roll de la VNLayer en los nodos o dispositivos móviles.

4.1.Componentes de la simulación

La generación de los patrones o trazas de movilidad son desarrollados por medio de BonnMotion como ya se mencionó. Esta herramienta de simulación posee diversos modelos de movilidad pero sus trazas son generadas para ser implementadas en NS2. Por lo tanto, NS3 recurre al módulo NS2MobilityHelper para convertirlos a eventos de movilidad NS3.

NS3 soporta protocolos tales como UDP, TCP, IP, además de varios protocolos de encaminamiento como AODV y OLSR, capaces de ofrecer el soporte necesario para las redes Ad-Hoc. Además, El mecanismo de virtualización (alojado en la capa de comunicación Ad-Hoc) requiere para incluir dos módulos, uno necesario para (VNLayer) y un protocolo de enrutamiento virtualizado (VNRouting), con el propósito de mejorar el rendimiento de la red Ad-Hoc.

En la figura 4.1 se describe mediante un diagrama a bloques el diseño de alto nivel de los componentes que intervienen en el proceso de simulación.

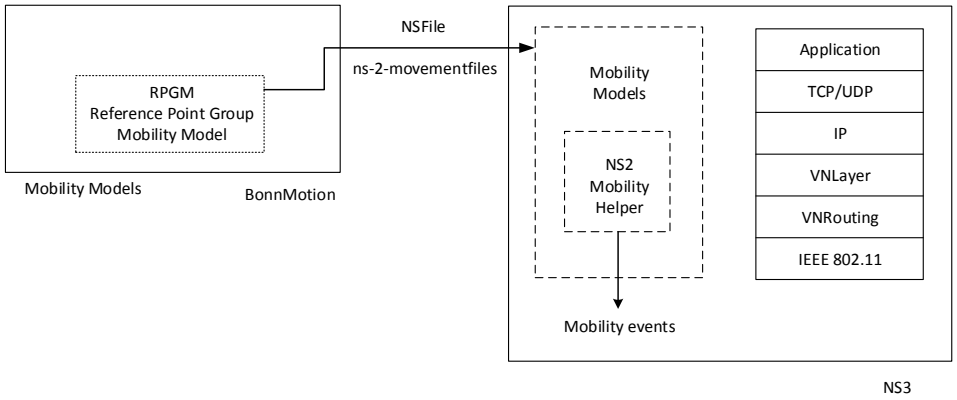


Figura 0.20 Diseño de alto nivel de los componentes del simulador

4.1.1. Bloques funcionales

De forma simplificada, para cada nodo se debe configurar:

- La capa de aplicación cuya función básicamente es realizar la tarea de un productor/consumidor de paquetes de datos.
- La pila de protocolos (UDP, TCP, IP, AODV, OLSR, etc.) necesaria para el encaminamiento.
- El dispositivo de red (NetDevice) que proporciona una interfaz tanto de entrada como de salida que da acceso a la capa de red.
- Un canal con la función de conectará una serie de dispositivos de red entre sí.

En la figura 4.2 (a) se muestra el modelo básico simplificado de una red NS3 en la cual se debe configurar aplicación que correrá sobre los nodos, la pila de protocolos, los dispotivos de la red y el canal de comunicación. Por otro lado, la introducción de un nuevo nivel a la pila de protocolos, y bajo el concepto de virtualización el nuevo modelo de red pasaría a ser como se muestra en la figura 4.2 (b).

El script de simulación (virtual-layer-module) está compuesto por las siguientes clases:

- **virtual-layer-helper:** ésta clase básicamente se encargan de instalar la capa virtual en la pila de protocolos de un nodo, de modo que desde el script de simulación a la hora de configurar la pila de protocolos de cada nodo se

emplearía esta clase para añadir al NetDevice de cada nodo el virtual-layer-net-device.

- **virtual-layer-net-device:** ésta clase sobrescribe algunos métodos de un NetDevice de un determinado nodo de modo que se añade una capa en la pila de protocolos a través de la cual pasan los paquetes de datos antes de llegar al NetDevice, y la cual genera sus propios paquetes de protocolo. Es decir, esta clase es la más importante, y se encargan de ejecutar el protocolo de la capa virtual siguiendo el diagrama de estados previamente revisado en el capítulo 3, así como de tomar decisiones de envío, procesado y reenvío de paquetes en función del rol de cada nodo.

- **virtual-layer-header:** ésta clase definen el formato de los diferentes paquetes enviados en el protocolo de la capa virtual, añadiendo datos a estos paquetes en casos puntuales como la tabla de Backups.

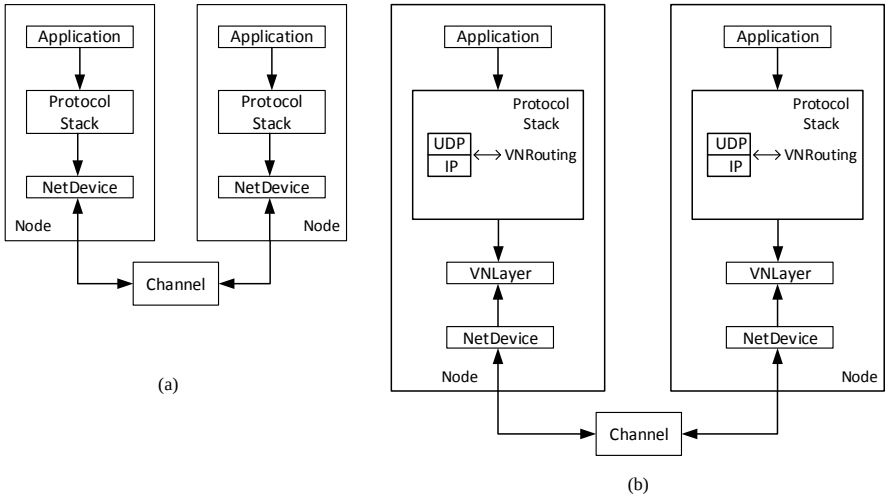


Figura 0.21 (a) Estructura de módulos de red básicos en NS3. (b) Estructura de módulos incluida la capa virtual

4.2.Escenario de movilidad RPGM

A pesar de que la simulación de los escenarios de movilidad guarda relación con el comportamiento humano, hemos de referirnos a nodos móviles. Para la realización de las trazas de movilidad se tomaron en cuenta los parámetros descritos en las tablas 4.1, 4.2 y 4.3.

- **Scenario:**

Descripción	Valor
Duración del escenario	60 [s]
Número de nodos	10 – 20 – 30 – 40
Ancho del escenario	40 [m]
Alto del escenario	40 [m]
Semilla randomica	0

Tabla 0.5 Parámetros utilizados para la generación de los escenarios

- **RandomSpeedBase:**

Descripción	Valor
Maxima velocidad	2 m/s
Minima velocidad	1 m/s
Pausa maxima	1 s

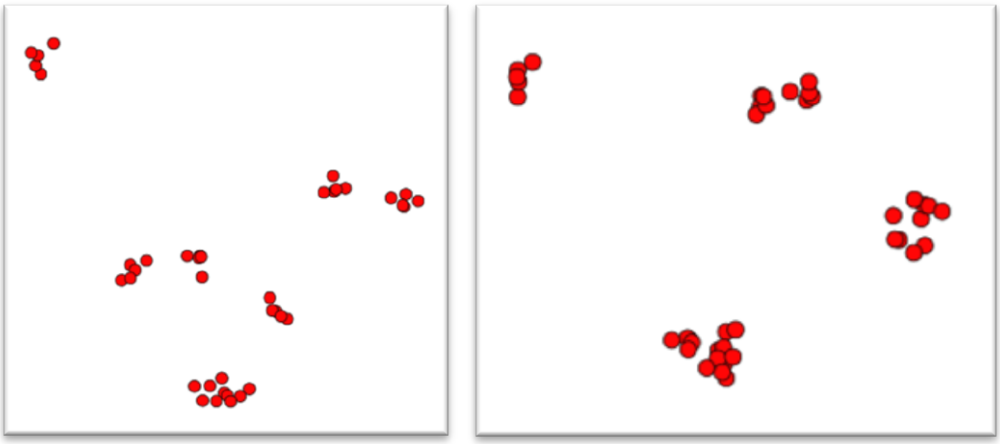
Tabla 0.6 Parámetros utilizados para la velocidad de los nodos

- **RPGM:**

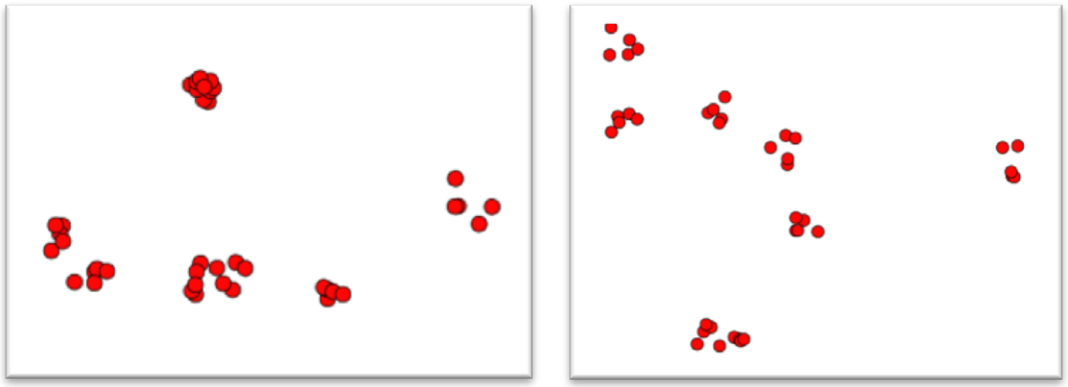
Descripción	Valor
Numero de nodos por grupo	5
Probabilidad de cambio de grupo	0
Máxima distancia al centro del grupo	2
Desviacion estandar	0

Tabla 0.7 Parámetros utilizados para la generación del modelo RPGM

El escenario generado cubre un área de 40 x 40 [m] por el cual los nodos móviles se desplazaran siguiendo los parámetros del modelo RPGM. En la figura 4.3 (a), (b) y (c) se muestra el escenario generado para 40 nodos móviles. Las observaciones fueron tomadas en un tiempo igual a 0, 5, 10, 20 segundos respectivamente.



(a) (b)



(c) (d)

Figura 0.22 Modelo RPGM para un tiempo (a) t=0, (b) t=5, (c) t=10, (d) t=20

4.3. VNLayer implementada.

Para explicar las características de la VNLayer implementada, se presenta el siguiente ejemplo, donde se establece un escenario de 10 nodos al cual se le ha impreso el modelo RPGM con los parámetros de entrada descrito en la sección 4.2. En la figura 4.4 se describe el escenario a simular, el cual posee un área de 50 x 50 m y está dividido en 4 regiones, inicialmente los nodos se encuentran en la región 2.

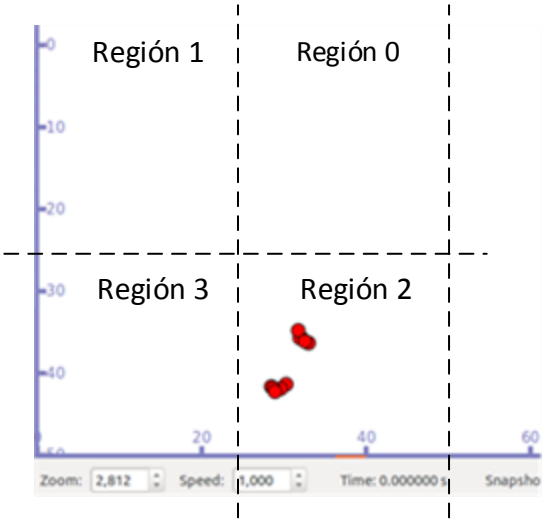


Figura 0.23 Modelo RPGM para 10 nodos sobre VNLayer

En primera instancia, se establecen las direcciones IP para cada nodo participante (10 nodos), tal como se muestra en la figura 4.5. Por lo tanto las direcciones IP van desde la 10.1.1.1 hasta la 10.1.1.10 para el nodo 0 hasta el nodo 9 respectivamente.

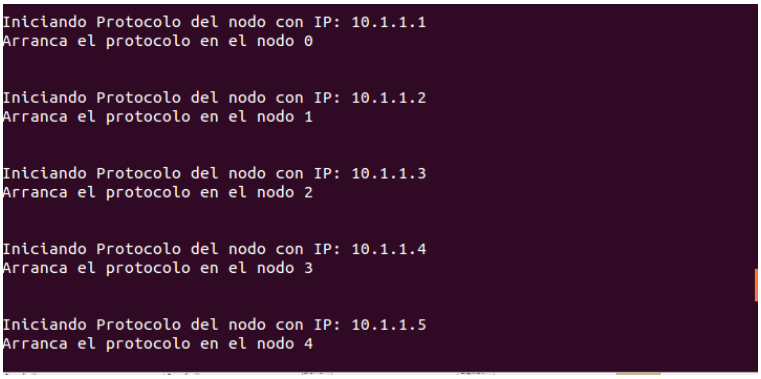


Figura 0.24 Establecimiento de direcciones IP para cada nodo

Posteriormente, se actualiza la localización de cada nodo, donde se detecta la región en la que se encuentra cada uno.

Los nodos pasan a estado REQUEST a la espera de la designación de liderato. Una vez expirado el TimerRequestWait, la elección de líder se designa a nodo que respondió primero (nodo 9). En esta etapa se generan los paquetes que utilizara la VNLayer para su funcionamiento. La figura 4.6 muestra la generación de paquetes dentro de la simulación.

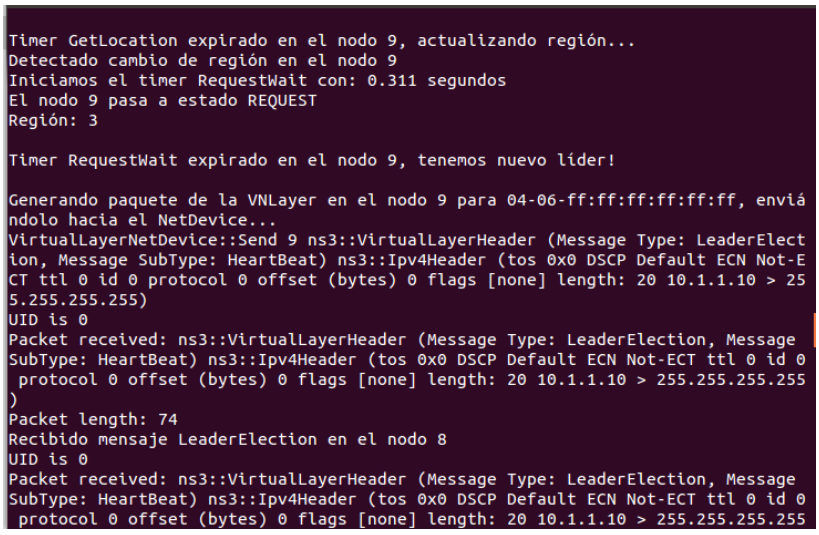


Figura 0.25 Inicialización de la región de cada nodo y elección de líder de región

El actual líder (nodo 9) anuncia su liderato al resto de nodos no líderes enviando un mensaje del tipo LeaderElection con su respectivo HeartBeat. En este caso, se envían paquetes desde la IP 10.1.1.10 a todos los nodos que forman parte de la región. En la figura 4.7 se muestra parte de la transmisión de los mensajes LeaderElection y HeartBeat.

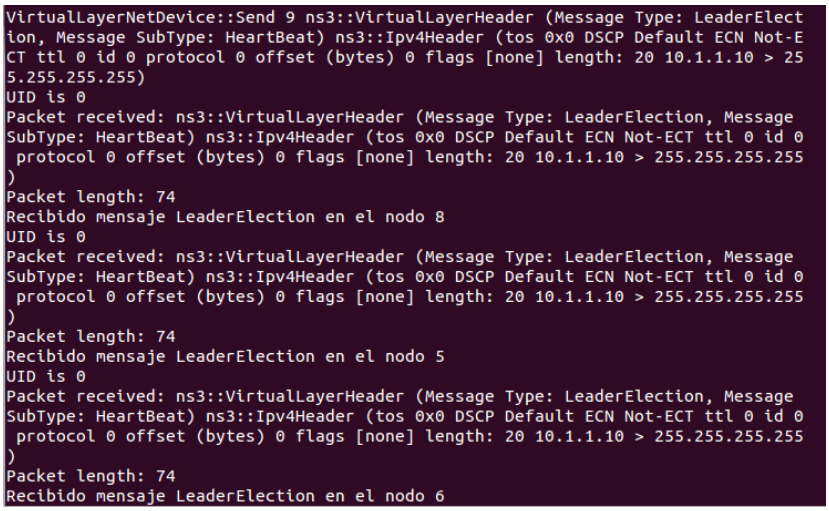


Figura 0.26 Trasmisión de paquetes del estado LeaderElection

Una vez terminado de enviar los mensajes LeaderElection a todos los nodos no líderes, se empieza el proceso para los nodos de BackUp. Todos los nodos no líderes envían mensajes de sincronización a la dirección IP del nodo líder actual.

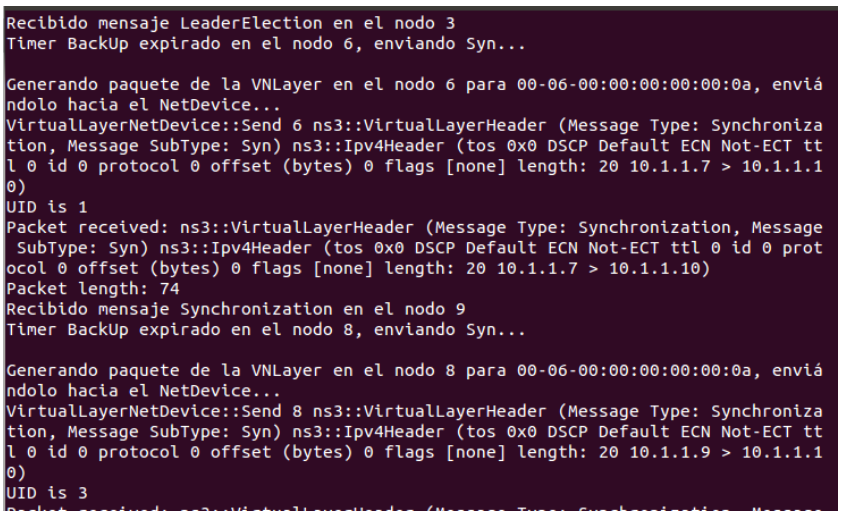


Figura 0.27 Mensajes de sincronización para los nodos de BackUp

Debido a la movilidad y a los cambios de región de los nodos se pueden dar situaciones en las que los nodos de BackUp asuman el roll de nuevo líder, de tal forma que el proceso liderato iniciara nuevamente. En la figura 4.9 se muestra que en un tiempo aproximado de 10 s un grupo de nodos ha pasado a la región 3.

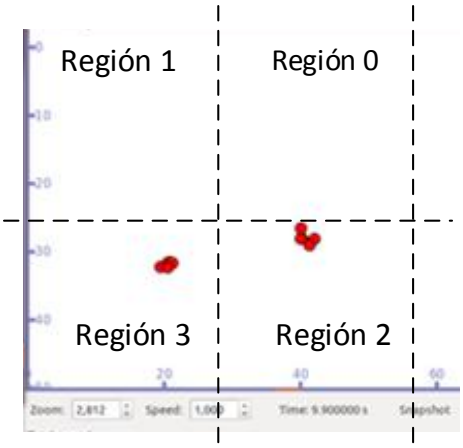


Figura 0.28 Cambio de región de un grupo de nodos

El desplazamiento de los nodos provoca cambios en la en el liderato de cada región, de tal modo que los nodos de BackUp están prestos a asumirlo. En la figura 4.10 se observa como el nodo 7 y el nodo 2 reclama el liderato de su respectiva región.

```
ssage SubType: Syn) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 0 id 0 protoco
l 0 offset (bytes) 0 flags [none] length: 20 10.1.1.1 > 10.1.1.3)
UID is 49
Packet received: ns3::VirtualLayerHeader (Message Type: Synchronization, Message SubType
: Syn) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 0 id 0 protocol 0 offset (b
ytes) 0 flags [none] length: 20 10.1.1.1 > 10.1.1.3)
Packet length: 74
Recibido mensaje Synchronization en el nodo 2
Timer BackUp expirado en el nodo 7 (líder), purgando BackUp Table...

Timer BackUp expirado en el nodo 2 (líder), purgando BackUp Table...

Timer Heartbeat expirado en el nodo 7 -> enviando Heartbeat...

Generando paquete de la VNLayer en el nodo 7 para 04-06-ff:ff:ff:ff:ff:ff, enviándolo ha
cia el NetDevice...
VirtualLayerNetDevice::Send 7 ns3::VirtualLayerHeader (Message Type: LeaderElection, Mes
sage SubType: HeartBeat) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 0 id 0 pr
otocol 0 offset (bytes) 0 flags [none] length: 20 10.1.1.8 > 255.255.255.255)
UID is 51
Packet received: ns3::VirtualLayerHeader (Message Type: LeaderElection, Message SubType:
 HeartBeat) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 0 id 0 protocol 0 offs
et (bytes) 0 flags [none] length: 20 10.1.1.8 > 255.255.255.255)
Packet length: 74
Recibido mensaje LeaderElection en el nodo 9
UID is 51
```

Figura 0.29 Los nodos de BackUp asumen el liderato de su respectiva región

La capa virtual tiene la capacidad de dividir el escenario en regiones en la cual se desplazan los nodos participantes de modo que cada región puede emular un nodo virtual facilitando la comunicación de los mismos, aquí se pudo constatar el funcionamiento de la capa virtual, donde se pudo verificar los estados por los que pasan los nodos que son leader o no-líder.

CAPÍTULO 5. PRUEBAS

Basados en el capítulo IV donde se presentó la capa virtual (VNLayer) y el desarrollo de las trazas de movilidad por medio de BonnMotion. Este capítulo contiene el resultado de las simulaciones (60 segundos de duración) de los escenarios de movilidad para 10 – 20 – 30 – 40 nodos móviles respectivamente.

Se realizará un análisis sobre la respuesta de la capa virtual en cada uno de los escenarios propuestos en la sección 4.2 y una comparación final de los mismos.

5.1.Escenario RPGM con 10 nodos móviles

De forma simplificada, para este escenario se realizaron observaciones sobre el nodo 4 (escogido arbitrariamente) de forma que se pudieron obtener distintos datos entregados por la VNLayer. Este nodo nunca asume el roll de líder.

En el escenario formado por 10 nodos móviles, los movimientos de cada nodo guardan relación dependiendo al grupo al pertenecen. En la figura 5.1 se observa la traza de movilidad para el nodo 4.

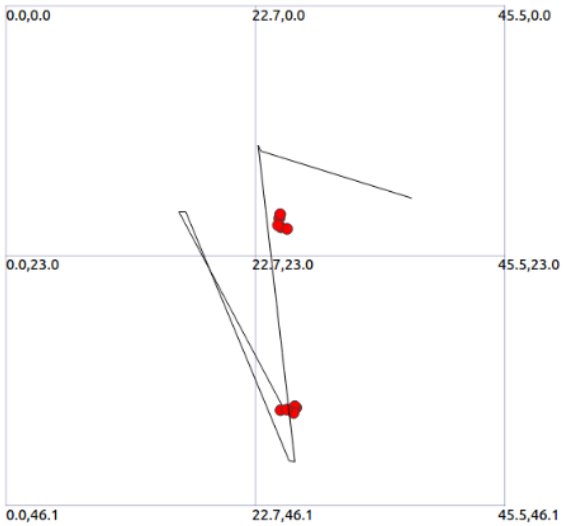


Figura 0.30 Traza de movilidad del nodo 4 del escenario RPGM con 10 nodos móviles

La VNLayer se encargará de coordinar los respectivos mensajes para la elección de líder en cada región, además de la sincronización y el asignamiento de nodos de Backup. Por lo tanto cada nodo debe recibir y transmitir los paquetes de la VNLayer para la coordinación respectiva. En la figura 5.2 se muestra que en efecto se da la transmisión y recepción de los paquetes de la VNLayer sobre el nodo 4.

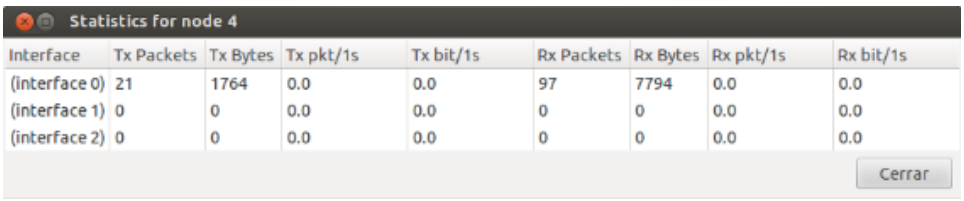


Figura 0.31 Transmisión y recepción de paquetes de la VNLayer sobre el nodo 4

Dado que el propio movimiento de cada nodo participante afecta a la red, la VNLayer debe esforzarse por mantener la comunicación entre todos los nodos participantes los paquetes y mensajes enviados y recibidos. Además, en cado nodo son diferentes dependiendo del momento y el lugar en el que se encuentre.

En la figura 5.3 se observa la carga de Bytes que envía el nodo 4 a la red de tal forma que en 60 segundos este envía un total 1764 Bytes en un total de 21 paquetes. Por otro lado en 60 s el nodo 4 recibe una carga de 97 paquetes con un peso de 7794 Bytes, que se muestra que se envían bytes de una forma un tanto lineal con respecto al tiempo, queriendo decir esto que no existen interferencias en la conexión.

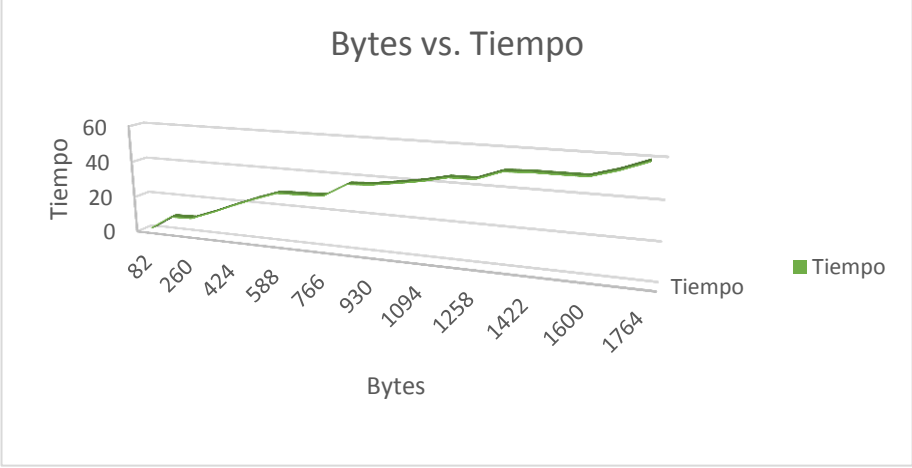


Figura 0.32 Cantidad de Bytes enviados a la red por el nodo 4 en 60 s

El nodo 4 necesita enviar mensajes por medio de la VNLayer a los demás dispositivos participantes dentro de la red para poder coordinarse. En la figura 5.4 se observa que los mensajes e sincronización son los mayormente enviados por el nodo 4, considerando un tiempo de 60 segundos a través de la VNLayer a la red. Esto significa que en este escenario los nodos tienden a mantener su liderato ya que los líderes de cada región van a permanecer más tiempo en su región, manteniendo de esta manera su liderato.

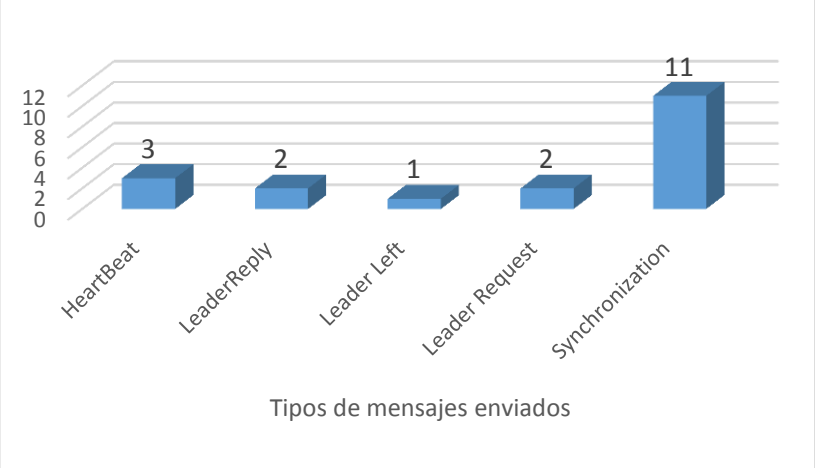


Figura 0.33 Tipos de mensajes enviados por el nodo 4 a la red

Con un escenario de 50 x 50 metros en el cual se han ubicado 10 nodos móviles que se desplaza con las características de descritas en la sección 4.2, la transmisión de paquetes alcanza un estimado de 258 paquetes con un peso de 21814 Bytes. Por otro lado la recepción de paquetes alcanza un estimado de 1008 paquetes con un peso de 80514 Bytes.

Entonces se puede ver que este nodo por el cual circula una mayor cantidad de paquetes corresponde al roll de nodo líder. Por lo tanto, los nodos que han asumido el roll de líder en el transcurso de la simulación son los nodos 1 y 7.

En la figura 5.5 se observa la relación entre los Bytes trasmitidos y los Bytes recibidos dentro del escenario RPGM con 10 nodos de movilidad en un tiempo de 60

segundos, donde se observa que los picos indican que esos nodos son los lideres por lo tanto tienen más procesamiento de paquetes.

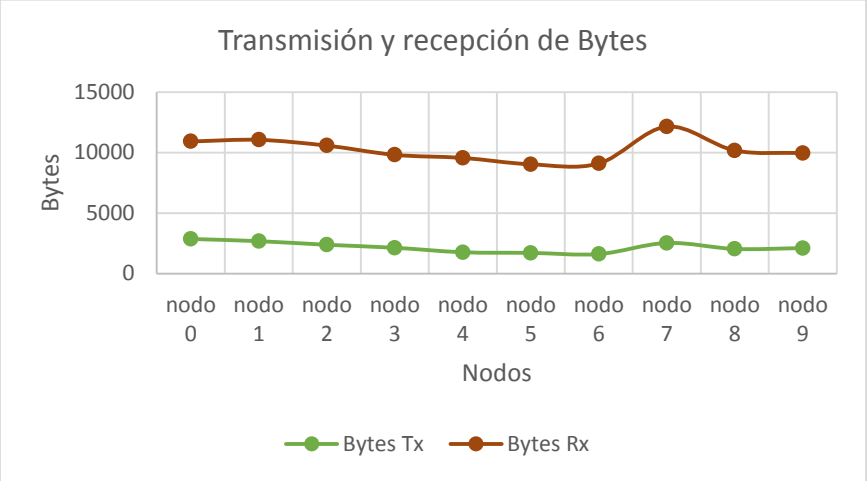


Figura 0.34 Transmisión y recepción de información de la VNLayer para 10 nodos móviles

5.2.Escenario RPGM con 20 nodos móviles

A continuación se presenta el análisis del nodo 19 (escogido arbitrariamente) del escenario RPGM con 20 nodos móviles. En la figura 5.6 se observa la traza de movilidad que este nodo posee en dicho escenario. Este nodo nunca asume el roll de líder.

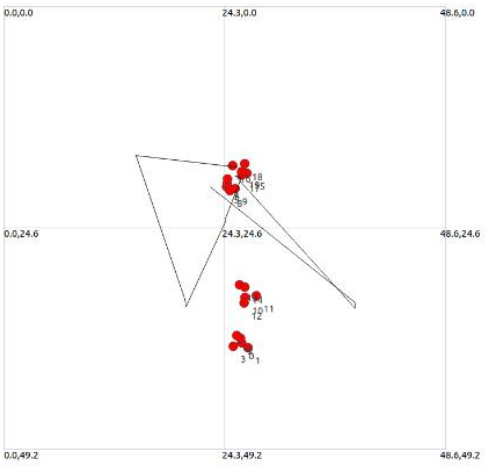


Figura 0.35 Traza de movilidad del nodo 19 del escenario RPGM con 20 nodos móviles

En un tiempo de 60 segundos el nodo 19 envía una carga de 26 paquetes con un peso de 2188 Bytes y recibe 162 paquetes con una peso de 13052 Bytes como se muestra en la figura 5.7.

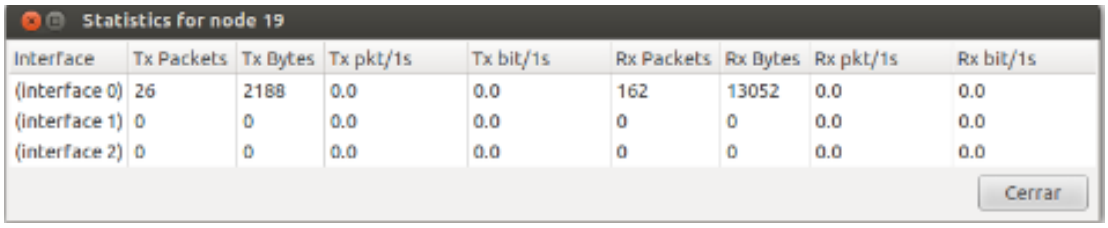


Figura 0.36 Transmisión y recepción de paquetes de la VNLayer sobre el nodo 19

La cantidad de Bytes enviados por el nodo 19 mediante la VNLayer para la coordinación de dicho nodo en un tiempo de 60 segundos puede observarse en la figura 5.8 en la cual se nota que el envió de paquetes de ejecuta de forma lineal y esto demuestra que no existe perdida de conexión.

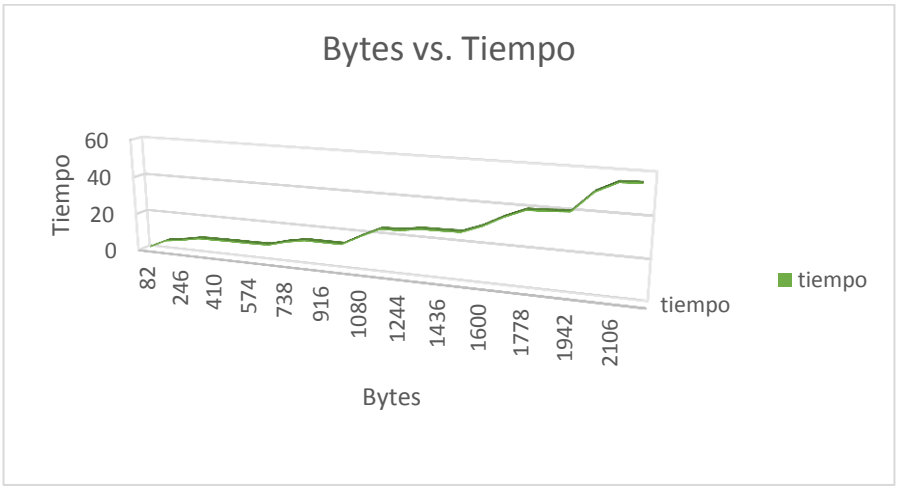


Figura 0.37 Cantidad de Bytes enviados a la red por el nodo 19 en 60 s

De igual forma, en el tiempo de simulación establecido, el nodo 19 debe enviar diferentes tipos de mensajes por medio de la VNLayer para su respectiva

coordinación en la red. En la figura 5.9 se muestran los tipos de mensajes enviados por el nodo 19.

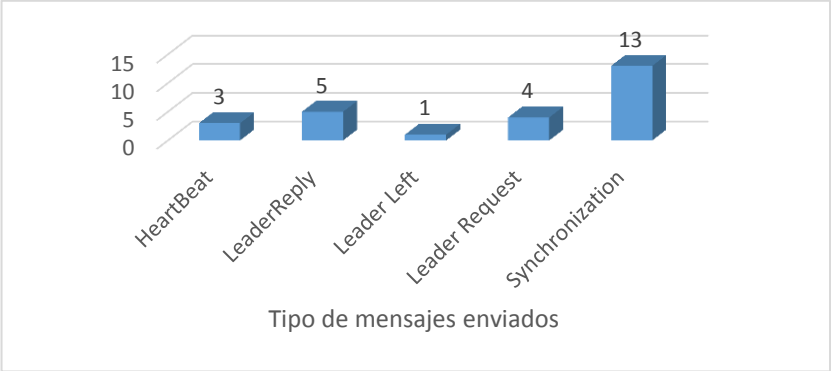


Figura 0.38 Tipos de mensajes enviados por el nodo 19 a la red

Para un escenario de 20 nodos móviles que se desplaza con las características descritas en la sección 4.2, la transmisión de paquetes alcanza un estimado de 476 paquetes con un peso de 40152 Bytes. Por otro lado la recepción de paquetes alcanza un estimado de 2870 paquetes con un peso de 244020 Bytes.

Los nodos que asumen el roll de líder en el transcurso de la simulación son los nodos 2, 12 y 18. Es decir que en cada caso la cantidad de nodos que asumen el roll de líder se incrementa según la cantidad de nodos estén en el escenario, pues la cantidad de trafico cursado por la red aumenta, además el área que van a cubrir es mayor en comparación al escenario de 10 nodos.

En la figura 5.10 se observa la relación entre los Bytes trasmitidos y los Bytes recibidos dentro del escenario RPGM con 20 nodos de movilidad en un tiempo de 60 segundos, donde se puede observar los picos en los nodos 2, 12 y 18, pero pequeños en el 5 y 19 esto se da debido a un cambio de región por pequeños instantes de tiempo.

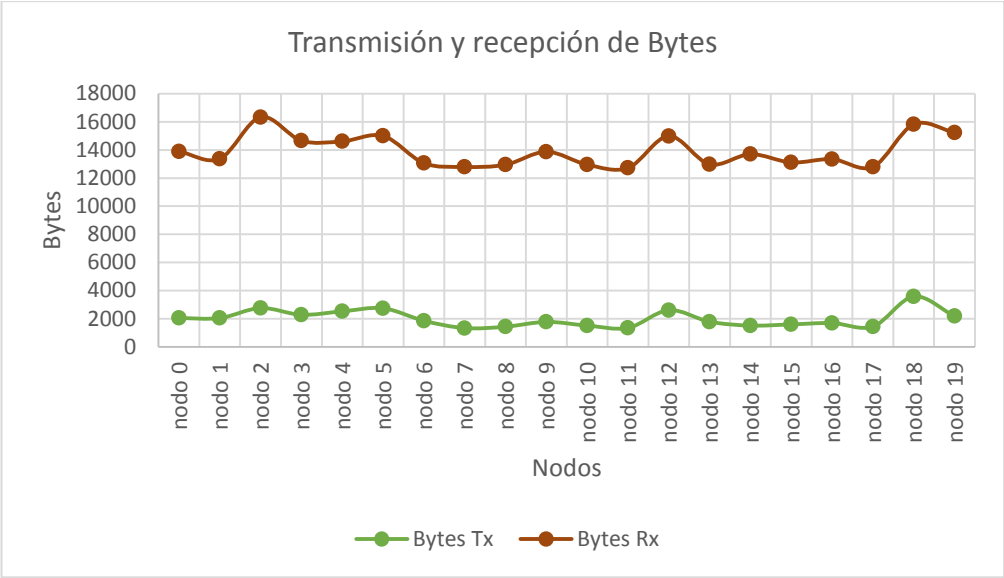


Figura 0.39 Transmisión y recepción de información de la VNLayer para 20 nodos móviles

5.3.Escenario RPGM con 30 nodos móviles

A continuación se presenta el análisis del nodo 24 (escogido arbitrariamente) del escenario RPGM con 30 nodos móviles. En la figura 5.11 se observa la traza de movilidad que posee dicho nodo. Este nodo nunca asume el roll de líder.

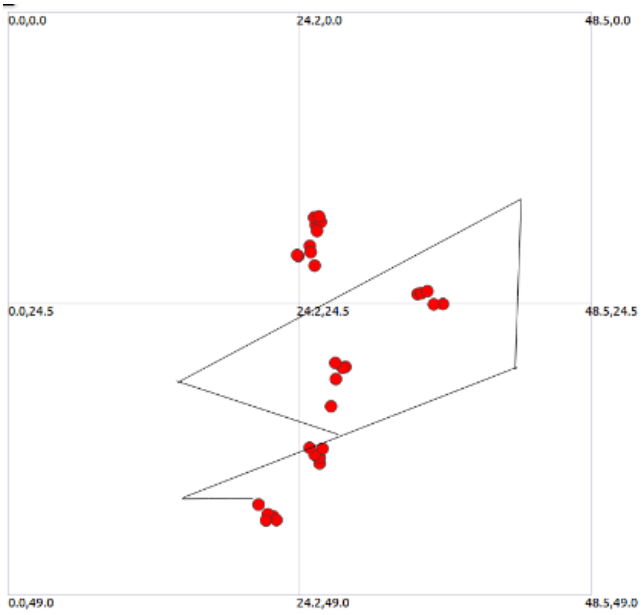


Figura 0.40 Traza de movilidad del nodo 24 del escenario RPGM con 30 nodos móviles

De igual forma el tiempo de simulación se establece en 60 segundos de modo que el nodo 24 envía una carga de 20 paquetes con un peso de 1710 Bytes y recibe 182 paquetes con una peso de 1550 Bytes como se muestra en la figura 5.12. De manera que a más número de nodos la cantidad de mensajes enviados o recibidos aumenta.

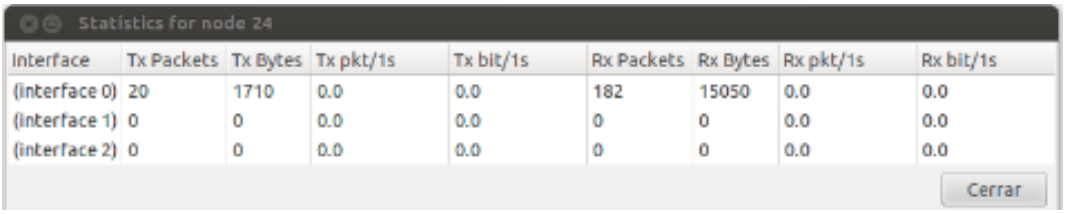


Figura 0.41 Transmisión y recepción de paquetes de la VNLayer sobre el nodo 24

Al igual que el resto de los nodos en la red, el nodo 24 envía una cantidad de 1710 Bytes mediante la VNLayer para la coordinación de dicho nodo en un tiempo de 60 segundos. En la figura 5.13 se observa la relación entre los Bytes enviado y el tiempo de simulación.

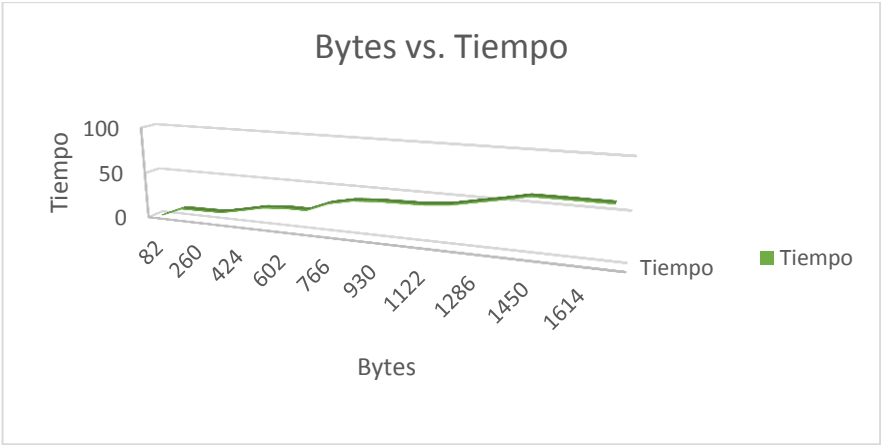


Figura 0.42 Cantidad de Bytes enviados a la red por el nodo 24 en 60 s

El nodo 24 debe enviar diferentes tipos de mensajes por medio de la VNLayer para su respectiva coordinación en la red. La figura 5.14 confirma lo mencionado en el escenario anterior, donde a mayor cantidad de nodos, mayor número de nodos líder y por lo tanto mayor cantidad de solicitudes de sincronización.

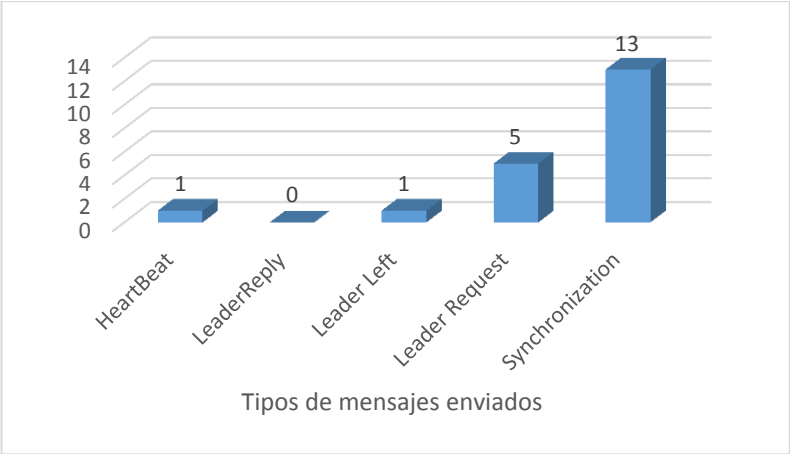


Figura 0.43 Tipos de mensajes enviados por el nodo 24 a la red

Para un escenario de 30 nodos la transmisión de paquetes alcanza un estimado de 627 paquetes con un peso de 52038 Bytes. Por otro lado la recepción de paquetes alcanza un estimado de 5560 paquetes con un peso de 458 Kbyte.

Los nodos que asumen el roll de líder en el transcurso de la simulación son los nodos 13, 17 y 26.

En la figura 5.15 se observa la relación entre los Bytes trasmitidos y los Bytes recibidos dentro del escenario RPGM con 30 nodos de movilidad en un tiempo de 60 segundos, en donde se observa 3 picos lo que implica 3 líderes, entonces al tener un mayor número de nodos y al estar estos dispersos, ocupan gran área, entonces se tendrá un líder por región ocupada.

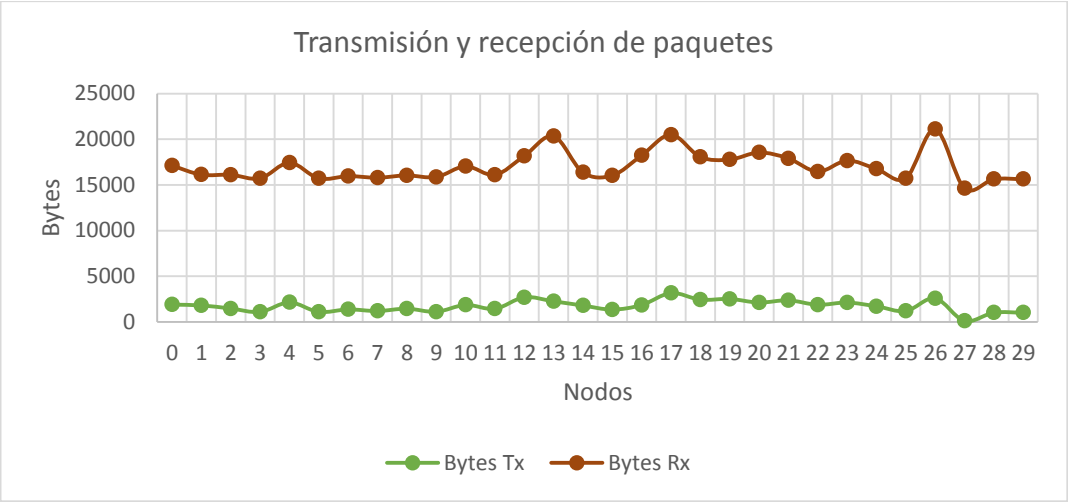


Figura 0.44 Transmisión y recepción de información de la VNLayer para 30 nodos móviles

5.4.Escenario RPGM con 40 nodos móviles

Para un tiempo de simulación de 60 segundos, a continuación se presenta el análisis del nodo 35 (escogido arbitrariamente) del escenario RPGM con 40 nodos móviles. En la figura 5.16 se observa la traza de movilidad que posee dicho nodo. Este nodo nunca asume el roll de líder.

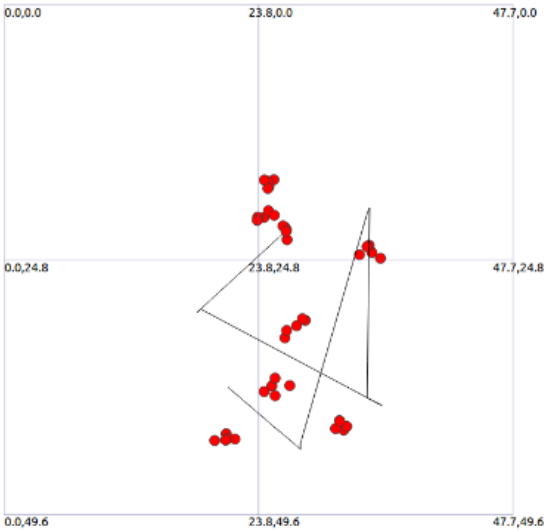


Figura 0.45 Traza de movilidad del nodo 35 del escenario RPGM con 40 nodos móviles

Al igual que los casos anteriores el tiempo de simulación se establece en 60 segundos de modo que el nodo 35 envía una carga de 19 paquetes con un peso de 1628 Bytes y recibe 239 paquetes con una peso de 20024 Bytes como se muestra en la figura 5.17

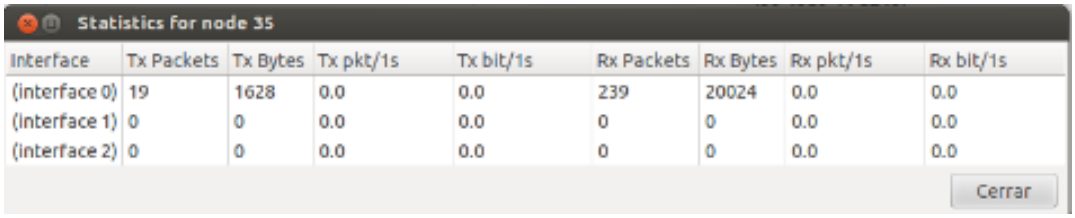


Figura 0.46 Transmisión y recepción de paquetes de la VNLayer sobre el nodo 35

La VNLayer se encarga de coordinar el proceso de comunicación entre los nodos participantes, de esta forma, el nodo 35envia hacia la rede una cantidad de 1628 Bytes por medio de la VNLayer la respectiva coordinación de dicho nodo en la red. En la figura 5.18 se observa la relación entre los Bytes enviados y el tiempo de simulación (60 segundos).

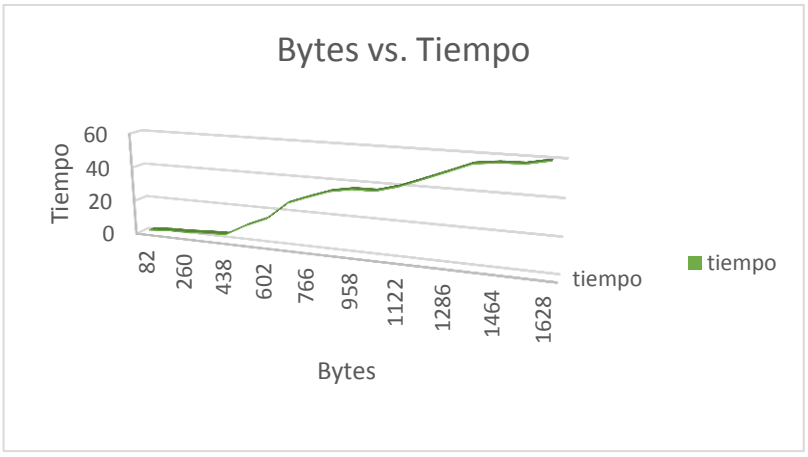


Figura 0.47 Cantidad de Bytes enviados a la red por el nodo 35 en 60 s

El nodo 35 debe enviar diferentes tipos de mensajes por medio de la VNLayer para su respectiva coordinación en la red. En la figura 5.19 se muestran los tipos de mensajes enviados por el nodo 35, el cual se observa no llega a ser líder en ningún momento ya que LeaderLeft tiene un valor de 0 pero quiere decir que no hubo la necesidad, ya que su prioridad como back up probablemente era muy baja.

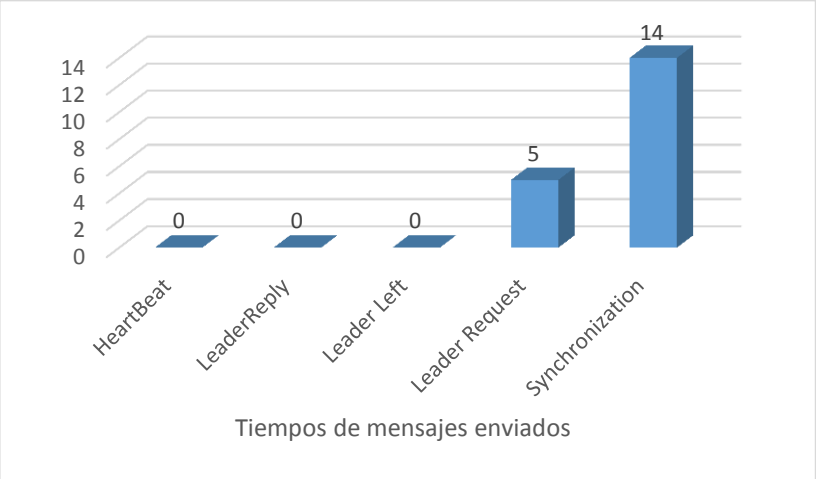


Figura 0.48 Tipos de mensajes enviados por el nodo 35 a la red

Para un escenario de 40 nodos la transmisión de paquetes alcanza un estimado de 857 paquetes con un peso de 71549 Bytes. Por otro lado la recepción de paquetes alcanza un estimado de 9673 paquetes con un peso de 808 Kbyte.

Los nodos que asumen el roll de líder en el transcurso de la simulación son los nodos 0, 8, 25 y 37.

En la figura 5.20 se observa la relación entre los Bytes trasmitidos y los Bytes recibidos dentro del escenario RPGM con 40 nodos de movilidad en un tiempo de 60 segundos, en donde podemos observar una gran cantidad de datos emitidos por el nodo 25 en particular el cual al ser líder de una región de varios nodos como en este caso enviara pues muchos más paquetes (sincronización, de liderato, etc.).

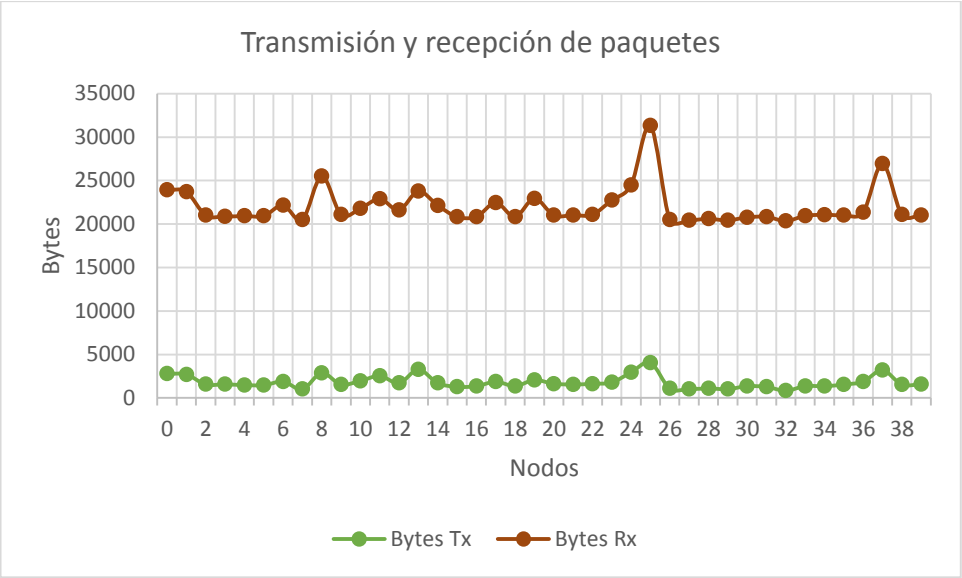


Figura 0.49 Transmisión y recepción de información de la VNLayer para 40 nodos móviles

5.5.Evaluación de la VNLayer

Con los resultados obtenidos anteriormente se observa que la VNLayer empieza a mejorar su rendimiento al momento que se incrementan los nodos móviles en una determinada área o escenario.

Dado que el escenario de simulación cubre un área de 50 x 50 metros al momento de simular el modelo RPGM con 10 nodos móviles la distancia de separación entre los grupos formados es mayor y por ende el envió de paquetes para establecer la comunicación se vuelve mayor. De este modo al tener una mayor densidad de nodos móviles en una misma área la coordinación de mensajes que realiza la VNLayer se vuelve más eficaz. Por lo tanto, en el caso del escenario con 10 nodos móviles la cantidad de paquetes enviados es de 258 pero para el escenario de 40 nodos el envío de paquetes es de 857. En la tabla 5.1 se muestran los valores de los paquetes y Bytes transmitidos y recibidos para cada escenario de movilidad propuesto.

A pesar de incrementarse 4 veces más la cantidad de nodos móviles en el escenario RPGM con 40 nodos el envío de paquetes se ha visto ligeramente reducido. De este modo se espera que para una mayor densidad de nodos móviles se de una transmisión de información más eficiente.

Escenario	Paquetes Tx	Paquetes Rx	Bytes Tx	Bytes Rx
RPGM 10 nodos	258	1008	21814	80514
RPGM 20 nodos	476	2870	40152	238020
RPGM 30 nodos	627	5560	52038	458524
RPGM 40 nodos	857	9673	71549	808175

Tabla 0.8 Información transmitida y recibida para cada escenario de movilidad.

Además, se adjunta el código desarrollado para la implementación de la VNLayer.

En Apéndice A. se muestra el código del módulo: Virtual-layer-header.cc

En Apéndice B. se muestra el código del módulo: Virtual-layer-header.cc

En Apéndice C. se muestra el código del módulo: Virtual-layer-net-device.cc

En Apéndice D. se muestra el código para la Implementación en NS3 de VNLayer y RPGM

CONCLUSIONES Y RECOMENDACIONES

Este proyecto de investigación propuso simular y analizar la creación de redes esporádicas móviles Ad-Hoc mediante la implementación de una capar virtual y un protocolo de enrutamiento basado en AODV, el cual pertenece a los protocolos reactivos que son exclusivos para redes MANET, y este protocolo se apega a nuestras necesidades, como es el caso de optimización de los recursos de la red para evitar el tráfico excesivo de datos para encaminamiento. Además se simulo el movimiento de los nodos móviles basados en el modelo de movilidad de grupo RPGM por medio de BonnMotion, permitiendo acercarse al comportamiento real de personas que se mueven en grupo en un área determinada. Mediante el simulador de redes de comunicación NS3 se implementó la capa virtual VNLayer, el protocolo AODV y las trazas de movilidad.

BonnMotion genera trazas de movilidad con extensión .ns_movements el cual establece los parámetros de movilidad tales como la velocidad máxima/minina, escenario de movilidad, numero de nodos participantes y otros dependiendo del modelo de movilidad.

Mediante los visualizadores NAM y Python se pudo analizar y observar los resultados de la capa virtual y las trazas de movilidad, los cuales se presenta los resultados a continuación.

Basados en el concepto de virtualización de redes el cual trata de emular un nodo virtual que proveerá de un mecanismo de encaminamiento para las REMs de forma que fusiona los conceptos de una topología plana y una topología jerárquica en donde los problemas de escalabilidad que presenta la topología plana pueden ser mitigados ya que esta topología posee cierto nivel de jerarquía. Por otro lado, en una topología clásica jerárquica existe el inconveniente que cuando se retira o se desconecta el cluster-head, los demás nodos que forman parte del mismo pueden quedar incomunicados.

A pesar de que la movilidad de los usuarios móviles puede ser bastante impredecible; la recreación de los escenarios basados en el modelo RPGM se asemeja bastante a la conducta humana. Se ha tomado en cuenta la velocidad máxima y minia a la cual una persona se desplazaría en una determinada circunstancia y además de su distancia social.

Mediante el análisis de las simulaciones de los distintos escenarios, se puedo observar que el uso de la capa virtual mejora el desempeño de la red al aumentar el número de nodos móviles, ya que el al tener un menor número de nodos, las regiones tienen a perder nodos de Backup lo cual implica que se pierda la comunicación entre regiones distantes causando rutas inestables y la inexistencia de nodos virtuales necesarios para el encaminamiento, además al tener una mayor cantidad de nodos en una región es menos probable que un nodo Backup sea líder ya que la lista de prioridad dependerá del número de nodos que exista en un región.

En el caso de tener un líder de región con una sobrecarga de trabajo, es decir que la cola de salida de paquetes sobrepasa su umbral de desempeño, se recomienda que en ese momento se pueda solicitar que un nodo de Backup también asuma el roll de líder el cual debería disponer de su propio identificador que permitirá que el resto de nodos diferencien entre los mensaje de distintos líderes. Del mismo modo si un líder con un determinado identificador abandona la región el resto de nodos (sus BackUps con mayor prioridad) competirán por sustituirlo, obteniendo dicho identificador aquel que lo consiga.

Además se recomienda para posteriores análisis la creación de un escenario de simulación realista de modo que se incluyan obstáculos que afecten a la propagación de las señales de los nodos, con este fin NS-3 proporciona el Buildings Module. Este módulo permite situar edificios con dimensiones tridimensionales dentro del escenario de simulación, pudiendo definir el tipo de construcción (residencial, oficina o comercio), el material de sus muros (madera, hormigón con o sin ventanas o bloques de piedra) y el número de plantas y de habitaciones [3].

Para aplicaciones enfocadas a un ambiente real se considera la heterogeneidad de las redes inalámbricas de modo que puedan trabajar conjuntamente (redes Ad-Hoc, 3G,

4G, redes wifi) desarrollando nuevas aplicaciones mejorando la experiencia del usuario. Como por ejemplo cloud computing o NaaS (network as service).

APÉNDICE A

Código del Módulo: Virtual-layer helper.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
#include "virtual-layer-helper.h"
#include "ns3/log.h"
#include "ns3/virtual-layer-net-device.h"
#include "ns3/node.h"
#include "ns3/names.h"
NS_LOG_COMPONENT_DEFINE ("VirtualLayerHelper");
namespace ns3 {
VirtualLayerHelper::VirtualLayerHelper ()
{
  NS_LOG_FUNCTION (this);
  m_deviceFactory.SetTypeId ("ns3::VirtualLayerNetDevice");
}

void VirtualLayerHelper::SetDeviceAttribute (std::string n1,
                                const AttributeValue &v1)
{
  NS_LOG_FUNCTION (this);
  m_deviceFactory.Set (n1, v1);
}
NetDeviceContainer VirtualLayerHelper::Install (const NetDeviceContainer c)
{
  NS_LOG_FUNCTION (this);
  NetDeviceContainer devs;
  for (uint32_t i = 0; i < c.GetN (); ++i)
    {
      Ptr<NetDevice> device = c.Get (i);
      NS_ASSERT_MSG (device != 0, "No NetDevice found in the node " << int(i) );
      Ptr<Node> node = device->GetNode ();
      NS_LOG_LOGIC ("**** Install VirtualLayer on node " << node->GetId ());
      Ptr<VirtualLayerNetDevice> dev = m_deviceFactory.Create<VirtualLayerNetDevice> ();
      devs.Add (dev);
      node->AddDevice (dev);
      dev->SetNetDevice (device);
    }
  return devs;
}
int64_t VirtualLayerHelper::AssignStreams (NetDeviceContainer c, int64_t stream)
{
  int64_t currentStream = stream;
  Ptr<NetDevice> netDevice;
  for (NetDeviceContainer::Iterator i = c.Begin (); i != c.End (); ++i)
    {
      netDevice = (*i);
      Ptr<VirtualLayerNetDevice> dev = DynamicCast<VirtualLayerNetDevice> (netDevice);
      if (dev)
        {
          currentStream += dev->AssignStreams (currentStream);
        }
    }
  return (currentStream - stream);
}
}
```

APÉNDICE B

Código del Módulo: Virtual-layer-header

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
#include "ns3/assert.h"
#include "ns3/log.h"
#include "ns3/abort.h"
#include "ns3/address-utils.h"
#include "virtual-layer-header.h"
namespace ns3 {
   /*
    * VirtualLayerHeader
    */
   NS_OBJECT_ENSURE_REGISTERED(VirtualLayerHeader);
   VirtualLayerHeader::VirtualLayerHeader() {
   }
   TypeId VirtualLayerHeader::GetTypeId(void) {
      static TypeId tid = TypeId("ns3::VirtualLayerHeader").SetParent<Header>
().AddConstructor<VirtualLayerHeader> ();
      return tid;
   }
   TypeId VirtualLayerHeader::GetInstanceTypeId(void) const {
      return GetTypeId();
   }
   void VirtualLayerHeader::Print(std::ostream & os) const {
      switch (m_type) {
         case Aplication:
            os << "Message Type: Aplication";
            break;
         case LeaderElection:
            os << "Message Type: LeaderElection";
            break;
         case Synchronization:
            os << "Message Type: Synchronization";
            break;
         case Hello:
            os << "Message Type: Hello";
            break;
         case virtual_to_aodv:
            os << "Message Type: virtual_to_aodv";
            break;
         case aodv_hello:
            os << "Message Type: aodv_hello";
            break;
         case topology:
            os << "Message Type: topology";
            break;
         case Empty:
            os << "Message Type: Empty";
            break;
         case RtRepair:
            os << "Message Type: RtRepair";
            break;
         default:
            os << "Message Type: UNDEFINED";
            break;
```

```
    }
    switch (m_subType) {
        case Client:
            os << ", Message SubType: Client";
            break;
        case Server:
            os << ", Message SubType: Server";
            break;
        case ForwardedServed:
            os << ", Message SubType: ForwardedServed";
            break;
        case ForwardedClient:
            os << ", Message SubType: ForwardedClient";
            break;
        case LeaderRequest:
            os << ", Message SubType: LeaderRequest";
            break;
        case LeaderReply:
            os << ", Message SubType: LeaderReply";
            break;
        case HeartBeat:
            os << ", Message SubType: HeartBeat";
            break;
        case LeaderLeft:
            os << ", Message SubType: LeaderLeft";
            break;
        case BackUpLeft:
            os << ", Message SubType: BackUpLeft";
            break;
        case Syn:
            os << ", Message SubType: Syn";
            break;
        case SynAck:
            os << ", Message SubType: SynAck";
            break;
        case hello:
            os << ", Message SubType: hello";
            break;
        case aodv:
            os << ", Message SubType: aodv";
            break;
        case AckRepair:
            os << ", Message SubType: AckRepair";
            break;
        case ChangeRoutes:
            os << ", Message SubType: ChangeRoutes";
            break;
        case CorrectionRouteRequest:
            os << ", Message SubType: CorrectionRouteRequest";
            break;
        case CorrectionRouteReply:
            os << ", Message SubType: CorrectionRouteReply";
            break;
        default:
            os << ", Message SubType: UNDEFINED";
            break;
    }
}
uint32_t VirtualLayerHeader::GetSerializedSize() const {
    uint32_t serializedSize = 33;
```

```
    serializedSize += 3 + m_srcMAC.GetLength() + m_dstMAC.GetLength() +
m_leaderMAC.GetLength();
    return serializedSize;
  }

  void VirtualLayerHeader::Serialize(Buffer::Iterator start) const {
    Buffer::Iterator i = start;
    uint8_t mac[20];
    i.WriteU8(static_cast<uint8_t> (m_type));
    i.WriteU8(static_cast<uint8_t> (m_subType));
    i.WriteHtonU16(m_region);
    i.WriteHtonU32(m_srcAddress.Get());
    m_srcMAC.CopyTo(mac);
    i.WriteU8(m_srcMAC.GetLength());
    i.Write(mac, m_srcMAC.GetLength());
    i.WriteHtonU32(m_dstAddress.Get());
    m_dstMAC.CopyTo(mac);
    i.WriteU8(m_dstMAC.GetLength());
    i.Write(mac, m_dstMAC.GetLength());
    i.WriteHtonU64(m_timeInState);
    i.WriteHtonU32(m_numBackUp);
    i.WriteHtonU32(m_backUp);
    i.WriteU8(static_cast<uint8_t> (m_rol));
    i.WriteHtonU32(m_leaderAddress.Get());
    m_leaderMAC.CopyTo(mac);
    i.WriteU8(m_leaderMAC.GetLength());
    i.Write(mac, m_leaderMAC.GetLength());
  }
  uint32_t VirtualLayerHeader::Deserialize(Buffer::Iterator start) {
    Buffer::Iterator i = start;
    uint8_t mac[20];
    m_type = static_cast<type> (i.ReadU8());
    m_subType = static_cast<subType> (i.ReadU8());
    m_region = i.ReadNtohU16();
    m_srcAddress.Set(i.ReadNtohU32());
    uint8_t len = i.ReadU8();
    i.Read(mac, len);
    m_srcMAC.CopyFrom(mac, len);
    m_dstAddress.Set(i.ReadNtohU32());
    len = i.ReadU8();
    i.Read(mac, len);
    m_dstMAC.CopyFrom(mac, len);
    m_timeInState = i.ReadNtohU64();
    m_numBackUp = i.ReadNtohU32();
    m_backUp = i.ReadNtohU32();
    m_rol = static_cast<typeState> (i.ReadU8());
    m_leaderAddress.Set(i.ReadNtohU32());
    len = i.ReadU8();
    i.Read(mac, len);
    m_leaderMAC.CopyFrom(mac, len);
    return GetSerializedSize();
  }
  void VirtualLayerHeader::SetSrcAddress(Ipv4Address srcAddress) {
    m_srcAddress = srcAddress;
  }
  Ipv4Address VirtualLayerHeader::GetSrcAddress() const {
    return m_srcAddress;
  }
  void VirtualLayerHeader::SetDstAddress(Ipv4Address dstAddress) {
    m_dstAddress = dstAddress;
```

```
}
Ipv4Address VirtualLayerHeader::GetDstAddress() const {
   return m_dstAddress;
}

VirtualLayerHeader::type VirtualLayerHeader::GetType() const {
   return m_type;
}
void VirtualLayerHeader::SetType(VirtualLayerHeader::type msgType) {
   m_type = msgType;
}
VirtualLayerHeader::subType VirtualLayerHeader::GetSubType() const {
   return m_subType;
}
void VirtualLayerHeader::SetSubType(VirtualLayerHeader::subType msgSubType) {
   m_subType = msgSubType;
}
u_int16_t VirtualLayerHeader::GetRegion() const {
   return m_region;
}
void VirtualLayerHeader::SetRegion(uint16_t region) {
   m_region = region;
}
uint64_t VirtualLayerHeader::GetTimeInState() const {
   return m_timeInState;
}
void VirtualLayerHeader::SetTimeInState(uint64_t timeInState) {
   m_timeInState = timeInState;
}
uint32_t VirtualLayerHeader::GetNumBackUp() const {
   return m_numBackUp;
}
void VirtualLayerHeader::SetNumBackUp(uint32_t numBackUp) {
   m_numBackUp = numBackUp;
}
uint32_t VirtualLayerHeader::GetBackUp() const {
   return m_backUp;
}
void VirtualLayerHeader::SetBackUp(uint32_t backUp) {
   m_backUp = backUp;
}
VirtualLayerHeader::typeState VirtualLayerHeader::GetRol() const {
   return m_rol;
}
void VirtualLayerHeader::SetRol(VirtualLayerHeader::typeState rol) {
   m_rol = rol;
}
Ipv4Address VirtualLayerHeader::GetLeaderAddress() const {
   return m_leaderAddress;
}
void VirtualLayerHeader::SetLeaderAddress(Ipv4Address leaderAddress) {
   m_leaderAddress = leaderAddress;
}
Address VirtualLayerHeader::GetLeaderMAC() const {
   return m_leaderMAC;
}
void VirtualLayerHeader::SetLeaderMAC(Address leaderMAC) {
   m_leaderMAC = leaderMAC;
}
Address VirtualLayerHeader::GetSrcMAC() const {
```

```
    return m_srcMAC;
  }
  void VirtualLayerHeader::SetSrcMAC(Address srcMAC) {
    m_srcMAC = srcMAC;
  }
  Address VirtualLayerHeader::GetDstMAC() const {
    return m_dstMAC;
  }
  void VirtualLayerHeader::SetDstMAC(Address dstMAC) {
    m_dstMAC = dstMAC;
  }
  std::ostream & operator<<(std::ostream & os, const VirtualLayerHeader & h) {
    h.Print(os);
    return os;
  }
}
```

APÉNDICE C

Código del Módulo: virtual-layer-net-device

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
#include <float.h>
#include "ns3/channel.h"
#include "ns3/packet.h"
#include "ns3/log.h"
#include "ns3/boolean.h"
#include "ns3/abort.h"
#include "ns3/simulator.h"
#include "ns3/uinteger.h"
#include "ns3/ipv4-l3-protocol.h"
#include "virtual-layer-net-device.h"
#include "virtual-layer-header.h"
#include "ns3/mobility-model.h"
#include "ns3/flow-monitor-module.h"
#define TIME_HEARTBEAT Seconds(5)
#define WAIT_HEARTBEAT Seconds(8)
#define WAIT_LESS_HEARTBEAT Seconds(5)
#define CURRENT_TIME Simulator::Now()
NS_LOG_COMPONENT_DEFINE("VirtualLayerNetDevice");
namespace ns3 {
   NS_OBJECT_ENSURE_REGISTERED(VirtualLayerNetDevice);
   TypeId VirtualLayerNetDevice::GetTypeId(void) {
      static TypeId tid = TypeId("ns3::VirtualLayerNetDevice")
            .SetParent<NetDevice> ()
            .AddConstructor<VirtualLayerNetDevice> ()
            .AddAttribute("Long",
            "The long of the simulation scenario",
            UintegerValue(50),
            MakeUintegerAccessor(&VirtualLayerNetDevice::m_long),
            MakeUintegerChecker<uint32_t> ())
            .AddAttribute("Width",
            "The width of the simulation scenario",
            UintegerValue(50),
            MakeUintegerAccessor(&VirtualLayerNetDevice::m_width),
            MakeUintegerChecker<uint32_t> ())
            .AddAttribute("Rows",
            "The number of rows of the simulation scenario",
            UintegerValue(2),
            MakeUintegerAccessor(&VirtualLayerNetDevice::m_rows),
            MakeUintegerChecker<uint32_t> ())
            .AddAttribute("Columns",
            "The number of columns of the simulation scenario",
            UintegerValue(2),
            MakeUintegerAccessor(&VirtualLayerNetDevice::m_columns),
            MakeUintegerChecker<uint32_t> ())
            .AddTraceSource("Tx", "Send - packet (including VNLayer header),
VirtualLayerNetDevice Ptr, interface index.",
            MakeTraceSourceAccessor(&VirtualLayerNetDevice::m_txTrace))
            .AddTraceSource("Rx", "Receive - packet (including VNLayer header),
VirtualLayerNetDevice Ptr, interface index.",
            MakeTraceSourceAccessor(&VirtualLayerNetDevice::m_rxTrace))
            .AddTraceSource("Drop", "Drop - DropReason, packet (including VNLayer header),
VirtualLayerNetDevice Ptr, interface index.",
```

```
            MakeTraceSourceAccessor(&VirtualLayerNetDevice::m_dropTrace))
            ;
      return tid;
   }
   VirtualLayerNetDevice::VirtualLayerNetDevice()
   : m_node(0),
   m_netDevice(0),
   m_ifIndex(0) {
      NS_LOG_FUNCTION(this);
      m_netDevice = 0;
      m_rng = CreateObject<UniformRandomVariable> ();
      srand(time(NULL));
      m_timerRequestWait.SetFunction(&VirtualLayerNetDevice::TimerRequestWaitExpire, this);
      m_timerHeartbeat.SetFunction(&VirtualLayerNetDevice::TimerHeartbeatExpire, this);
      m_timerHeartbeatWait.SetFunction(&VirtualLayerNetDevice::TimerHeartbeatWaitExpire, this);
      m_timerGetLocation.SetFunction(&VirtualLayerNetDevice::TimerGetLocationExpire, this);
      m_timerBackUp.SetFunction(&VirtualLayerNetDevice::TimerBackUpExpire, this);
      m_region = unknown;
      m_leaderAddress.Set(none);
      m_state = VirtualLayerHeader::INITIAL;
      m_timeInState = 0;
      m_expires = 0;
      ClearBackUp();
   }
   Ptr<NetDevice> VirtualLayerNetDevice::GetNetDevice() const {
      NS_LOG_FUNCTION(this);
      return m_netDevice;
   }
   void VirtualLayerNetDevice::SetNetDevice(Ptr<NetDevice> device) {
      NS_LOG_FUNCTION(this << device);
      m_netDevice = device;
      NS_LOG_DEBUG("RegisterProtocolHandler for " << device-
>GetInstanceTypeId().GetName());
      uint16_t protocolType = 0;
      m_node-
>RegisterProtocolHandler(MakeCallback(&VirtualLayerNetDevice::ReceiveFromDevice,
            this),
            protocolType, device, false);
   }
   int64_t VirtualLayerNetDevice::AssignStreams(int64_t stream) {
      NS_LOG_FUNCTION(this << stream);
      m_rng->SetStream(stream);
      return 1;
   }
   void VirtualLayerNetDevice::DoDispose() {
      NS_LOG_FUNCTION(this);
      m_netDevice = 0;
      m_node = 0;
      NetDevice::DoDispose();
   }
   void VirtualLayerNetDevice::ReceiveFromDevice(Ptr<NetDevice> incomingPort,
         Ptr<const Packet> packet,
         uint16_t protocol,
         Address const &src,
         Address const &dst,
         PacketType packetType) {
      NS_LOG_FUNCTION(this << incomingPort << packet << protocol << src << dst);
      NS_LOG_DEBUG("UID is " << packet->GetUid());
      Ptr<Packet> copyPkt = packet->Copy();
      m_rxTrace(copyPkt, m_node->GetObject<VirtualLayerNetDevice> (), GetIfIndex());
```

```
      NS_LOG_DEBUG("Packet received: " << *copyPkt);
      NS_LOG_DEBUG("Packet length: " << copyPkt->GetSize());
      VirtualLayerHeader virtualLayerHdr;
      copyPkt->RemoveHeader(virtualLayerHdr);
      switch (virtualLayerHdr.GetType()) {
         case VirtualLayerHeader::LeaderElection:
            RecvLeaderElection(packet->Copy());
            SendAODV(VirtualLayerHeader::virtual_to_aodv, m_ip);
            break;
         case VirtualLayerHeader::Synchronization:
            RecvSynchronization(packet->Copy());
            SendAODV(VirtualLayerHeader::virtual_to_aodv, m_ip);
            break;
         default:
            NS_LOG_INFO("Paquete recibido por la VNLayer! --> Enviándolo hacia
Ipv4L3Protocol...");
            if (!m_promiscRxCallback.IsNull()) {
               m_promiscRxCallback(this, copyPkt, Ipv4L3Protocol::PROT_NUMBER, src, dst,
packetType);
            }
            m_rxCallback(this, copyPkt, Ipv4L3Protocol::PROT_NUMBER, src);
      }
      return;
   }
   void VirtualLayerNetDevice::SetIfIndex(const uint32_t index) {
      NS_LOG_FUNCTION(this << index);
      m_ifIndex = index;
   }
   uint32_t VirtualLayerNetDevice::GetIfIndex(void) const {
      NS_LOG_FUNCTION(this);
      return m_ifIndex;
   }
   Ptr<Channel> VirtualLayerNetDevice::GetChannel(void) const {
      NS_LOG_FUNCTION(this);
      NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
      return m_netDevice->GetChannel();
   }
   void VirtualLayerNetDevice::SetAddress(Address address) {
      NS_LOG_FUNCTION(this << address);
      NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
      m_netDevice->SetAddress(address);
   }
   Address VirtualLayerNetDevice::GetAddress(void) const {
      NS_LOG_FUNCTION(this);
      NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
      return m_netDevice->GetAddress();
   }
   bool VirtualLayerNetDevice::SetMtu(const uint16_t mtu) {
      NS_LOG_FUNCTION(this << mtu);
      NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
      return m_netDevice->SetMtu(mtu);
   }
   uint16_t VirtualLayerNetDevice::GetMtu(void) const {
      NS_LOG_FUNCTION(this);
      uint16_t mtu = m_netDevice->GetMtu();
      // Minimum MTU for Ipv4
```

```
    if (mtu < 576) {
      mtu = 576;
    }
    return mtu;
  }
  bool VirtualLayerNetDevice::IsLinkUp(void) const {
    NS_LOG_FUNCTION(this);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->IsLinkUp();
  }
  void VirtualLayerNetDevice::AddLinkChangeCallback(Callback<void> callback) {
    NS_LOG_FUNCTION(this);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->AddLinkChangeCallback(callback);
  }
  bool VirtualLayerNetDevice::IsBroadcast(void) const {
    NS_LOG_FUNCTION(this);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->IsBroadcast();
  }
  Address VirtualLayerNetDevice::GetBroadcast(void) const {
    NS_LOG_FUNCTION(this);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->GetBroadcast();
  }
  bool VirtualLayerNetDevice::IsMulticast(void) const {
    NS_LOG_FUNCTION(this);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->IsMulticast();
  }
  Address VirtualLayerNetDevice::GetMulticast(Ipv4Address multicastGroup) const {
    NS_LOG_FUNCTION(this << multicastGroup);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->GetMulticast(multicastGroup);
  }
  Address VirtualLayerNetDevice::GetMulticast(Ipv6Address addr) const {
    NS_LOG_FUNCTION(this << addr);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->GetMulticast(addr);
  }
  bool VirtualLayerNetDevice::IsPointToPoint(void) const {
    NS_LOG_FUNCTION(this);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->IsPointToPoint();
  }
  bool VirtualLayerNetDevice::IsBridge(void) const {
    NS_LOG_FUNCTION(this);
    NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
    return m_netDevice->IsBridge();
  }
  bool VirtualLayerNetDevice::Send(Ptr<Packet> packet,
```

```
            const Address& dest,
            uint16_t protocolNumber) {
        NS_LOG_FUNCTION(this << *packet << dest << protocolNumber);
        bool ret = false;
        Address src;
        ret = DoSend(packet, src, dest, protocolNumber, false);
        return ret;
    }
    bool VirtualLayerNetDevice::SendFrom(Ptr<Packet> packet,
            const Address& src,
            const Address& dest,
            uint16_t protocolNumber) {
        NS_LOG_FUNCTION(this << *packet << src << dest << protocolNumber);
        bool ret = false;
        ret = DoSend(packet, src, dest, protocolNumber, true);
        return ret;
    }
    bool VirtualLayerNetDevice::DoSend(Ptr<Packet> packet,
            const Address& src,
            const Address& dest,
            uint16_t protocolNumber,
            bool doSendFrom) {
        NS_LOG_FUNCTION(this << *packet << src << dest << protocolNumber << doSendFrom);
        NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
        Ptr<Packet> origPacket = packet->Copy();
        bool ret = false;
        VirtualLayerHeader virtualLayerHdr = VirtualLayerHeader();
        virtualLayerHdr.SetType(VirtualLayerHeader::Aplication);
        virtualLayerHdr.SetSubType(VirtualLayerHeader::Client);
        virtualLayerHdr.SetRegion(m_region);
        virtualLayerHdr.SetSrcAddress(m_ip);
        virtualLayerHdr.SetSrcMAC(src);
        virtualLayerHdr.SetDstMAC(dest);
        virtualLayerHdr.SetRol(m_state);
        virtualLayerHdr.SetLeaderAddress(m_leaderAddress);
        virtualLayerHdr.SetTimeInState(m_timeInState);
        virtualLayerHdr.SetNumBackUp(m_numBackUp);
        virtualLayerHdr.SetBackUp(m_backUp);
        packet->AddHeader(virtualLayerHdr);
        NS_LOG_INFO("Paquete recibido por la VNLayer en el nodo " << m_node->GetId() << " para
" << dest << ", enviándolo hacia el NetDevice...");
        m_txTrace(packet, m_node->GetObject<VirtualLayerNetDevice> (), GetIfIndex());
        if (doSendFrom) {
            NS_LOG_DEBUG("VirtualLayerNetDevice::SendFrom " << m_node->GetId() << " " <<
*packet);
            ret = m_netDevice->SendFrom(packet, src, dest, protocolNumber);
        } else {
            NS_LOG_DEBUG("VirtualLayerNetDevice::Send " << m_node->GetId() << " " << *packet);
            ret = m_netDevice->Send(packet, dest, protocolNumber);
        }
        return ret;
    }
    Ptr<Node> VirtualLayerNetDevice::GetNode(void) const {
        NS_LOG_FUNCTION(this);
        return m_node;
    }
    void VirtualLayerNetDevice::SetNode(Ptr<Node> node) {
        NS_LOG_FUNCTION(this << node);
        m_node = node;
```

```
}
bool VirtualLayerNetDevice::NeedsArp(void) const {
  NS_LOG_FUNCTION(this);
  NS_ASSERT_MSG(m_netDevice != 0, "VirtualLayer: can't find any lower-layer protocol " <<
m_netDevice);
  return m_netDevice->NeedsArp();
}
void VirtualLayerNetDevice::SetReceiveCallback(NetDevice::ReceiveCallback cb) {
  NS_LOG_FUNCTION(this << &cb);
  m_rxCallback = cb;
}
void VirtualLayerNetDevice::SetPromiscReceiveCallback(NetDevice::PromiscReceiveCallback cb)
{
  NS_LOG_FUNCTION(this << &cb);
  m_promiscRxCallback = cb;
}
bool VirtualLayerNetDevice::SupportsSendFrom() const {
  NS_LOG_FUNCTION(this);
  return true;
}
void VirtualLayerNetDevice::Start(Ipv4Address ip, Vector2D scn1, Vector2D scn2) {
  NS_LOG_INFO("Arranca el protocolo en el nodo " << m_node->GetId() << "\n");
  m_ip = ip;
  m_mac = this->GetAddress();
  m_long = static_cast<uint32_t> (scn1.x);
  m_width = static_cast<uint32_t> (scn1.y);
  m_rows = static_cast<uint32_t> (scn2.x);
  m_columns = static_cast<uint32_t> (scn2.y);
  m_ifIndex = m_node->GetObject<Ipv4L3Protocol>()->GetInterfaceForDevice(m_netDevice);
  m_timerGetLocation.Schedule(Seconds(0.01));
}
void VirtualLayerNetDevice::TimerRequestWaitExpire() {
  NS_LOG_INFO("Timer RequestWait expirado en el nodo " << m_node->GetId() << ", tenemos
nuevo líder!\n");
  m_state = VirtualLayerHeader::LEADER;
  m_timeInState = CURRENT_TIME.GetDouble();
  m_leaderAddress = m_ip;
  m_leaderMAC = m_mac;
  if (m_numBackUp) SendPkt(VirtualLayerHeader::LeaderElection,
VirtualLayerHeader::HeartBeat, Ipv4Address(m_tableBackUp[0][0]), m_backUpMACs[0]);
  else SendPkt(VirtualLayerHeader::LeaderElection, VirtualLayerHeader::HeartBeat,
Ipv4Address::GetBroadcast(), m_netDevice->GetBroadcast());
  m_timerHeartbeat.Cancel();
  m_timerHeartbeat.Schedule(TIME_HEARTBEAT);
  SendAODV(VirtualLayerHeader::virtual_to_aodv, m_ip); //envia al agente de enrutamiento su
region, su estado y su lider
}
void VirtualLayerNetDevice::TimerHeartbeatExpire() {
  NS_LOG_INFO("Timer Heartbeat expirado en el nodo " << m_node->GetId() << " -> enviando
Heartbeat...\n");
  if (m_numBackUp) SendPkt(VirtualLayerHeader::LeaderElection,
VirtualLayerHeader::HeartBeat, Ipv4Address(m_tableBackUp[0][0]), m_backUpMACs[0]);
  else SendPkt(VirtualLayerHeader::LeaderElection, VirtualLayerHeader::HeartBeat,
Ipv4Address::GetBroadcast(), m_netDevice->GetBroadcast());
  m_timerHeartbeat.Cancel();
  m_timerHeartbeat.Schedule(TIME_HEARTBEAT);
  SendAODV(VirtualLayerHeader::virtual_to_aodv, m_ip); //envia al agente de enrutamiento su
region, su estado y su lider
}
void VirtualLayerNetDevice::TimerHeartbeatWaitExpire() {
```

```
      switch (m_state) {
         case VirtualLayerHeader::NONLEADER:
            m_timerHeartbeatWait.Cancel();
            m_timerHeartbeatWait.Schedule(WAIT_LESS_HEARTBEAT);
            m_state = VirtualLayerHeader::UNSTABLE;
            m_expires++;
            NS_LOG_INFO("Timer HeartbeatWait expirado en el nodo " << m_node->GetId() << " ->
pasa al estado UNSTABLE\n");
            break;
         case VirtualLayerHeader::UNSTABLE:
            m_expires++;
            if (m_expires > 2) {
               m_state = VirtualLayerHeader::REQUEST;
               CancelTimers();
               double Rand = (double(rand() % (200 + 1))) / 100;
               m_timerRequestWait.Schedule(Seconds(1 + Rand));
               m_expires = 0;
               NS_LOG_INFO("Timer HeartbeatWait expirado en el nodo " << m_node->GetId() << " -
> pasa al estado REQUEST\n");
            } else {
               m_timerHeartbeatWait.Cancel();
               m_timerHeartbeatWait.Schedule(WAIT_LESS_HEARTBEAT);
               NS_LOG_INFO("Timer HeartbeatWait expirado en el nodo " << m_node->GetId() << " -
> reiniciamos timer...\n");
            }
            break;
         default:;
      }
      SendAODV(VirtualLayerHeader::virtual_to_aodv, m_ip); //envia al agente de enrutamiento su
region, su estado y su lider
   }
   void VirtualLayerNetDevice::TimerGetLocationExpire() {
      NS_LOG_INFO("Timer GetLocation expirado en el nodo " << m_node->GetId() << ",
actualizando región...");
      int region = GetLocation();
      if (region != m_region) {
         NS_LOG_INFO("Detectado cambio de región en el nodo " << m_node->GetId());
         CancelTimers();
         if (m_leaderAddress.IsEqual(m_ip)) { // Si es el lider envia un LeaderLeft
            if (m_numBackUp) //Si hay BackUp se lo envía a él para que le sustituya
               SendPkt(VirtualLayerHeader::LeaderElection, VirtualLayerHeader::LeaderLeft,
Ipv4Address(m_tableBackUp[0][0]), m_backUpMACs[0]);
            else //Si no, lo envía por broadcast para iniciar una nueva contienda
               SendPkt(VirtualLayerHeader::LeaderElection, VirtualLayerHeader::LeaderLeft,
Ipv4Address::GetBroadcast(), m_netDevice->GetBroadcast());
         } else if (m_backUp > 0) { // Si es BackUp envía BackUpLeft al líder
            SendPkt(VirtualLayerHeader::Synchronization, VirtualLayerHeader::BackUpLeft,
m_leaderAddress, m_leaderMAC);
         }
         m_region = region;
         ClearBackUp();
         m_leaderAddress.Set(none)
         uint8_t mac[20];
         for (int i = 0; i < 20; i++) {
            mac[i] = 0x00;
         }
         m_leaderMAC.CopyFrom(mac, 20);
         if (m_state != VirtualLayerHeader::INITIAL) SendPkt(VirtualLayerHeader::LeaderElection,
VirtualLayerHeader::LeaderRequest, Ipv4Address::GetBroadcast(), m_netDevice->GetBroadcast());
```

```
        double Rand = (double(rand() % (200 + 1))) / 1000; // valor randomico entre 0 a 3s ejm
2.33seg
        NS_LOG_INFO("Iniciamos el timer RequestWait con: " << Rand + 0.3 << " segundos");
        m_timerRequestWait.Cancel();
        m_timerRequestWait.Schedule(Seconds(0.3 + Rand));
        NS_LOG_INFO("El nodo " << m_node->GetId() << " pasa a estado REQUEST");
        m_state = VirtualLayerHeader::REQUEST;
      }
      m_timerGetLocation.Schedule(GetExitTime());
      NS_LOG_INFO("Región: " << m_region << "\n");
      SendAODV(VirtualLayerHeader::virtual_to_aodv, m_ip); //envia al agente de enrutamiento su
region, su estado y su lider
   }
   void VirtualLayerNetDevice::TimerBackUpExpire() {
      switch (m_state) {
         case VirtualLayerHeader::LEADER:
            PurgeBackUp();
            NS_LOG_INFO("Timer BackUp expirado en el nodo " << m_node->GetId() << " (líder),
purgando BackUp Table...\n");
            break;
         case VirtualLayerHeader::NONLEADER:
            if (m_backUp != 0) {
               NS_LOG_INFO("Timer BackUp expirado en el nodo " << m_node->GetId() << ",
enviando SynAck (es BackUp)...\n");
               SendPkt(VirtualLayerHeader::Synchronization, VirtualLayerHeader::SynAck,
Ipv4Address::GetBroadcast(), m_netDevice->GetBroadcast());
            } else {
               NS_LOG_INFO("Timer BackUp expirado en el nodo " << m_node->GetId() << ",
enviando Syn...\n");
               SendPkt(VirtualLayerHeader::Synchronization, VirtualLayerHeader::Syn,
m_leaderAddress, m_leaderMAC);
            }
            break;
         default:;
      }
      SendAODV(VirtualLayerHeader::virtual_to_aodv, m_ip); //envia al agente de enrutamiento su
region, su estado y su lider
   }
   void VirtualLayerNetDevice::CancelTimers(void) {
      if (m_timerRequestWait.IsRunning())
         m_timerRequestWait.Cancel();
      if (m_timerHeartbeat.IsRunning())
         m_timerHeartbeat.Cancel();
      if (m_timerHeartbeatWait.IsRunning())
         m_timerHeartbeatWait.Cancel();
   }
   int VirtualLayerNetDevice::GetLocation(void) {
      int xRegion;
      int yRegion;
      Vector3D pos = m_node->GetObject<MobilityModel>()->GetPosition();
      xRegion = int (pos.x / (double (m_long) / double (m_columns)));
      yRegion = int (pos.y / (double (m_width) / double (m_rows)));
      return (uint8_t) (m_columns * yRegion + xRegion);
   }
   /*
    * Funcion: estima el tiempo de salida
    *  de un nodo de su actual region
    */
   Time VirtualLayerNetDevice::GetExitTime(void) {
      // verificamos que no haya cambiado de region
```

```
int region = GetLocation();
if (region != m_region)
  return Seconds(0.0);
Vector3D pos = m_node->GetObject<MobilityModel>()->GetPosition();
Vector3D vel = m_node->GetObject<MobilityModel>()->GetVelocity();
double speed = sqrt(pow(vel.x, 2) + pow(vel.y, 2));
double dist = DBL_MAX;
// Si la velocidad es cero, retorna por defecto un valor para volver a consultar region
if (speed == 0)
  return Seconds(2.0);
double xR = int (pos.x / (double (m_long) / double (m_columns))) * (double (m_long) / double
(m_columns));
double yR = int (pos.y / (double (m_width) / double(m_rows))) * (double (m_width) /
double(m_rows));
// Cut points
Vector2D cut1, cut2, cut3, cut4;
// Vectors from the position to the cut points
Vector2D v1, v2, v3, v4;
// The node is moving along the vertical axis --> There are 2 cut points (Straight 1 and 3)
if (vel.x == 0) {
  // Cut with the straight 1
  cut1.x = (vel.x * yR + (vel.y * pos.x - vel.x * pos.y)) / vel.y;
  cut1.y = yR;
  // Cut with the straight 3
  cut3.x = (vel.x * (yR + (m_width / m_rows)) + (vel.y * pos.x - vel.x * pos.y)) / vel.y;
  cut3.y = yR + (m_width / m_rows);
  v1.x = cut1.x - pos.x;
  v1.y = cut1.y - pos.y;
  v3.x = cut3.x - pos.x;
  v3.y = cut3.y - pos.y;
  // If the direction of the v1 vector and the velocity vector is the same --> The node is going to
the cut1 point
  if ((v1.x * vel.x >= 0) && (v1.y * vel.y >= 0)) {
    dist = sqrt(pow(v1.x, 2) + pow(v1.y, 2));
    // retorna el tiempo + un delta t para asegurar la
    // salida de la region actual
    return (Seconds(dist / speed) + dt);
  } else {
    dist = sqrt(pow(v3.x, 2) + pow(v3.y, 2));
    // retorna el tiempo + un delta t para asegurar la
    // salida de la region actual
    return (Seconds(dist / speed) + dt);
  }
}// The node is moving along the horizontal axis --> There are 2 cut points (Straight 2 and 4)
else if (vel.y == 0) {
  // Cut with the straight 2
  cut2.x = xR;
  cut2.y = (vel.y * xR - (vel.y * pos.x - vel.x * pos.y)) / vel.x;
  // Cut with the straight 4
  cut4.x = xR + (m_long / m_columns);
  cut4.y = (vel.y * (xR + (m_long / m_columns)) - (vel.y * pos.x - vel.x * pos.y)) / vel.x;
  v2.x = cut2.x - pos.x;
  v2.y = cut2.y - pos.y;

  v4.x = cut4.x - pos.x;
  v4.y = cut4.y - pos.y;
  // If the direction of the v2 vector and the velocity vector is the same --> The node is going to
the cut2 point
  if ((v2.x * vel.x >= 0) && (v2.y * vel.y >= 0)) {
    dist = sqrt(pow(v2.x, 2) + pow(v2.y, 2));
```

```
            // retorna el tiempo + un delta t para asegurar la
            // salida de la region actual
            return (Seconds(dist / speed) + dt);
          } else {
            dist = sqrt(pow(v4.x, 2) + pow(v4.y, 2));
            // retorna el tiempo + un delta t para asegurar la salida de la region actual
            return (Seconds(dist / speed) + dt);
          }
        } else {
          // Cut with the straight 1
          cut1.x = (vel.x * yR + (vel.y * pos.x - vel.x * pos.y)) / vel.y;
          cut1.y = yR;
          // Cut with the straight 2
          cut2.x = xR;
          cut2.y = (vel.y * xR - (vel.y * pos.x - vel.x * pos.y)) / vel.x;
          // Cut with the straight 3
          cut3.x = (vel.x * (yR + (m_width / m_rows)) + (vel.y * pos.x - vel.x * pos.y)) / vel.y;
          cut3.y = yR + (m_width / m_rows);
          // Cut with the straight 4
          cut4.x = xR + (m_long / m_columns);
          cut4.y = (vel.y * (xR + (m_long / m_columns)) - (vel.y * pos.x - vel.x * pos.y)) / vel.x;
          v1.x = cut1.x - pos.x;
          v1.y = cut1.y - pos.y;
          v2.x = cut2.x - pos.x;
          v2.y = cut2.y - pos.y;
          v3.x = cut3.x - pos.x;
          v3.y = cut3.y - pos.y;
          v4.x = cut4.x - pos.x;
          v4.y = cut4.y - pos.y;
          if ((v1.x * vel.x >= 0) && (v1.y * vel.y >= 0) && (sqrt(pow(v1.x, 2) + pow(v1.y, 2)) < dist)) {
            dist = sqrt(pow(v1.x, 2) + pow(v1.y, 2));
          }
          if ((v2.x * vel.x >= 0) && (v2.y * vel.y >= 0) && (sqrt(pow(v2.x, 2) + pow(v2.y, 2)) < dist)) {
            dist = sqrt(pow(v2.x, 2) + pow(v2.y, 2));
          }
          if ((v3.x * vel.x >= 0) && (v3.y * vel.y >= 0) && (sqrt(pow(v3.x, 2) + pow(v3.y, 2)) < dist)) {
            dist = sqrt(pow(v3.x, 2) + pow(v3.y, 2));
          }
          if ((v4.x * vel.x >= 0) && (v4.y * vel.y >= 0) && (sqrt(pow(v4.x, 2) + pow(v4.y, 2)) < dist)) {
            dist = sqrt(pow(v4.x, 2) + pow(v4.y, 2));
          }
          // retorna el tiempo + un delta t para asegurar la salida de la region actual
          return (Seconds(dist / speed) + dt);
        }
      }
      void VirtualLayerNetDevice::PurgeBackUp(void) {
        m_numBackUp = 0;
        uint8_t mac[20];
        for (int i = 0; i < 20; i++) {
          mac[i] = 0x00;
        }
        for (int i = 0; i < NUM_BACKUP; i++) {
          if (m_tableBackUp[i][1] == non_syn) { //Borramos los no sincronizados
            m_tableBackUp[i][0] = empty;
            m_backUpMACs[i].CopyFrom(mac, 20);
          } else if (m_tableBackUp[i][0] == m_leaderAddress.Get()) { //Borramos al lider
            m_tableBackUp[i][0] = empty;
            m_tableBackUp[i][1] = non_syn;
            m_backUpMACs[i].CopyFrom(mac, 20);
```

```
        } else if (m_tableBackUp[i][0] != empty) //Si hay un backup sincronizado incrementamos
numbackup
            m_numBackUp++;
      }
      for (int i = 0; i < NUM_BACKUP; i++) { //Reordena
         if (m_tableBackUp[i][0] == empty) {
            for (int j = i; j < NUM_BACKUP - 1; j++) {
               m_tableBackUp[j][0] = m_tableBackUp[j + 1][0];
               m_tableBackUp[j][1] = m_tableBackUp[j + 1][1];
               m_backUpMACs[j] = m_backUpMACs[j + 1];
            }
         }
      }

   }
   void VirtualLayerNetDevice::PutBackUp(Ipv4Address ipBackUp, Address macBackUp) {
      for (int i = 0; i < NUM_BACKUP; i++) {
         if (m_tableBackUp[i][0] == empty) {
            m_tableBackUp[i][0] = ipBackUp.Get();
            m_tableBackUp[i][1] = non_syn;
            m_backUpMACs[i] = macBackUp;
            break;
         }
      }
   }
   void VirtualLayerNetDevice::QuitBackUp(Ipv4Address ipBackUp) {
      uint8_t mac[20];
      for (int i = 0; i < 20; i++) {
         mac[i] = 0x00;
      }
      for (int i = 0; i < NUM_BACKUP; i++) {
         if (m_tableBackUp[i][0] == ipBackUp.Get()) {
            m_tableBackUp[i][0] = empty;
            m_tableBackUp[i][1] = non_syn;
            m_backUpMACs[i].CopyFrom(mac, 20);
         }
      }
   }
   void VirtualLayerNetDevice::SynBackUp(Ipv4Address ipBackUp) {
      for (int i = 0; i < NUM_BACKUP; i++) {
         if (m_tableBackUp[i][0] == ipBackUp.Get())
            m_tableBackUp[i][1] = syn;
      }
   }
   void VirtualLayerNetDevice::ClearBackUp(void) {
      m_backUp = 0;
      m_numBackUp = 0;
      uint8_t mac[20];
      for (int i = 0; i < 20; i++) {
         mac[i] = 0x00;
      }
      for (int i = 0; i < NUM_BACKUP; i++) {
         m_tableBackUp[i][0] = empty;
         m_tableBackUp[i][1] = non_syn;
      }
      for (int i = 0; i < NUM_BACKUP; i++) {
         m_backUpMACs[i].CopyFrom(mac, 20);
      }
   }
   void VirtualLayerNetDevice::SetBackUp(void) {
```

```
      m_numBackUp = 0;
      uint8_t mac[20];
      for (int i = 0; i < 20; i++) {
         mac[i] = 0x00;
      }
      for (int i = 0; i < NUM_BACKUP; i++) {
         if (m_tableBackUp[i][0] == m_leaderAddress.Get()) {
            m_tableBackUp[i][0] = empty;
            m_tableBackUp[i][1] = non_syn;
            m_backUpMACs[i].CopyFrom(mac, 20);
         } else if (m_tableBackUp[i][0] != empty) {
            m_numBackUp++;
         }
      }
      for (int i = 0; i < NUM_BACKUP; i++) {
         if (m_tableBackUp[i][0] == empty) {
            for (int j = i; j < NUM_BACKUP - 1; j++) {
               m_tableBackUp[j][0] = m_tableBackUp[j + 1][0];
               m_tableBackUp[j][1] = m_tableBackUp[j + 1][1];
               m_backUpMACs[j] = m_backUpMACs[j + 1];
               m_tableBackUp[j + 1][0] = empty;
               m_tableBackUp[j + 1][1] = non_syn;
               m_backUpMACs[j + 1].CopyFrom(mac, 20);
            }
         }
      }
   }
   void VirtualLayerNetDevice::RecvLeaderElection(Ptr<Packet> pkt) {
      NS_LOG_INFO("Recibido mensaje LeaderElection en el nodo " << m_node->GetId());
      VirtualLayerHeader virtualLayerHdr;
      pkt->RemoveHeader(virtualLayerHdr);
      Ipv4Header ipv4Hdr;
      pkt->RemoveHeader(ipv4Hdr);
      double Rand;
      if (virtualLayerHdr.GetRegion() == m_region) {
         switch (virtualLayerHdr.GetSubType()) {
            case VirtualLayerHeader::LeaderRequest:
               switch (m_state) {
                  case VirtualLayerHeader::LEADER:
                     SendPkt(VirtualLayerHeader::LeaderElection, VirtualLayerHeader::LeaderReply,
virtualLayerHdr.GetSrcAddress(), virtualLayerHdr.GetSrcMAC());
                     break;
                  default:;
               }
               break;
            case VirtualLayerHeader::LeaderReply:
               CancelTimers();
               m_timerHeartbeatWait.Schedule(WAIT_HEARTBEAT);
               m_leaderAddress = ipv4Hdr.GetSource();
               m_leaderMAC = virtualLayerHdr.GetSrcMAC();
               m_state = VirtualLayerHeader::NONLEADER;
               break;
            case VirtualLayerHeader::HeartBeat:
               switch (m_state) {
                  case VirtualLayerHeader::LEADER:
                     ClearBackUp();
                     if (virtualLayerHdr.GetTimeInState() < m_timeInState) { //No soy el lider
                        CancelTimers();
                        m_timerHeartbeatWait.Schedule(WAIT_HEARTBEAT);
                        m_leaderAddress = virtualLayerHdr.GetLeaderAddress();
```

```
                        m_leaderMAC = virtualLayerHdr.GetLeaderMAC();
                        m_state = VirtualLayerHeader::NONLEADER;
                        m_timeInState = 0;
                     } else if (virtualLayerHdr.GetTimeInState() == m_timeInState) { //Solicitud de lider
nuevamente
                        CancelTimers();
                        m_state = VirtualLayerHeader::REQUEST;
                        m_leaderAddress = m_ip;
                        m_leaderMAC = m_mac;
                        Rand = (double(rand() % (300 + 1))) / 1000;
                        //Rand = (rand() / (double) RAND_MAX)*4;
                        m_timerRequestWait.Cancel();
                        m_timerRequestWait.Schedule(Seconds(0.5 + Rand));
                     } else { //Soy el lider
                        CancelTimers();
                        if (m_numBackUp) SendPkt(VirtualLayerHeader::LeaderElection,
VirtualLayerHeader::HeartBeat, Ipv4Address(m_tableBackUp[0][0]), m_backUpMACs[0]);
                        else SendPkt(VirtualLayerHeader::LeaderElection,
VirtualLayerHeader::HeartBeat, Ipv4Address::GetBroadcast(), m_netDevice->GetBroadcast());
                        m_timerHeartbeat.Schedule(TIME_HEARTBEAT);
                     }
                     break;
                  default:
                     CancelTimers();
                     int a = 0;
                     m_timerHeartbeatWait.Schedule(WAIT_HEARTBEAT);
                     if (!m_leaderAddress.IsEqual(ipv4Hdr.GetSource())) a = 1;
                     m_leaderAddress = ipv4Hdr.GetSource();
                     m_leaderMAC = virtualLayerHdr.GetLeaderMAC();
                     if (a) SetBackUp();
                     SetBackUp(); // ¿¿??
                     m_state = VirtualLayerHeader::NONLEADER;
                     m_timeInState = 0;
                     m_expires = 0;
                  }
                  break;
               case VirtualLayerHeader::LeaderLeft:
                  if (ipv4Hdr.GetSource().IsEqual(m_leaderAddress)) {
                     if (m_state == VirtualLayerHeader::NONLEADER) {
                        if (ipv4Hdr.GetDestination().IsEqual(m_ip)) { // Soy el Backup, me convierto en
Lider
                           CancelTimers();
                           m_timerRequestWait.Schedule(Seconds(0.));
                           m_backUp = 0;
                           SetBackUp();
                        } else {
                           CancelTimers();
                           Rand = (double(rand() % (400 + 1))) / 1000;
                           //Rand = (rand() / (double) RAND_MAX)*5;
                           m_state = VirtualLayerHeader::REQUEST;
                           m_leaderAddress.Set(none);
                           uint8_t mac[20];
                           for (int i = 0; i < 20; i++) {
                              mac[i] = 0x00;
                           }
                           m_leaderMAC.CopyFrom(mac, 20);

                           if (m_backUp) {
                              m_timerRequestWait.Schedule(Seconds(0.2 - double((3 - m_backUp)*0.1)));
                           } else {
```

```
                        m_timerRequestWait.Schedule(Seconds(0.3 + Rand));
                    }
                    if (m_backUp) m_backUp--;
                }
            }
        }
        break;
      default:;
    }
    // por cada mensaje escuchado de su lider re-establece el tiempo de su timerHeartbeatWait
    if ((m_state == VirtualLayerHeader::NONLEADER) &&
(virtualLayerHdr.GetSrcAddress().IsEqual(m_leaderAddress)) &&
        (virtualLayerHdr.GetSubType() != VirtualLayerHeader::LeaderLeft)) {
      CancelTimers();
      m_timerHeartbeatWait.Schedule(WAIT_HEARTBEAT);
      if ((virtualLayerHdr.GetNumBackUp() < NUM_BACKUP) &&
          ((m_backUp == 0) || (virtualLayerHdr.GetNumBackUp() < m_backUp))) {
        m_backUp = 0;
        Rand = (double(rand() % (50 + 1))) / 100;
        //Rand = (rand() / (double) RAND_MAX) * 0.5;
        m_timerBackUp.Cancel();
        m_timerBackUp.Schedule(Seconds(Rand));
      }
    }
  }
}
void VirtualLayerNetDevice::RecvSynchronization(Ptr<Packet> pkt) {
  NS_LOG_INFO("Recibido mensaje Synchronization en el nodo " << m_node->GetId());
  VirtualLayerHeader virtualLayerHdr;
  pkt->RemoveHeader(virtualLayerHdr);
  Ipv4Header ipv4Hdr;
  pkt->RemoveHeader(ipv4Hdr);
  if (virtualLayerHdr.GetRegion() != m_region) {
    // Drop
    return;
  }
  switch (virtualLayerHdr.GetSubType()) {
    case VirtualLayerHeader::Syn:
      if (virtualLayerHdr.GetSrcAddress().IsEqual(m_leaderAddress)) {
        CancelTimers();
        m_timerHeartbeatWait.Schedule(WAIT_HEARTBEAT);
      }
      if (m_state == VirtualLayerHeader::LEADER) {
        if (m_numBackUp < NUM_BACKUP) {
          PutBackUp(virtualLayerHdr.GetSrcAddress(), virtualLayerHdr.GetSrcMAC());
          m_numBackUp++;
          SendAODV(VirtualLayerHeader::Synchronization, virtualLayerHdr.GetSrcAddress());
          m_timerBackUp.Cancel();
          m_timerBackUp.Schedule(Seconds(1));
        }
      }
      if ((m_state == VirtualLayerHeader::NONLEADER) &&
(virtualLayerHdr.GetSrcAddress().IsEqual(m_leaderAddress)) &&
(ipv4Hdr.GetDestination().IsEqual(m_ip))) {
        CancelTimers();
        m_timerHeartbeatWait.Schedule(WAIT_HEARTBEAT);
        // envia el mensaje de sincronizacion al agente VAODV para
        // sincronizar su R_table
        m_backUp = virtualLayerHdr.GetNumBackUp();
        m_timerBackUp.Cancel();
```

```
                m_timerBackUp.Schedule(Seconds(0.5));
              }
              break;
            case VirtualLayerHeader::SynAck:
              if (m_state == VirtualLayerHeader::NONLEADER)
PutBackUp(virtualLayerHdr.GetSrcAddress(), virtualLayerHdr.GetSrcMAC());
              SynBackUp(virtualLayerHdr.GetSrcAddress());
              break;
            case VirtualLayerHeader::BackUpLeft:
              QuitBackUp(virtualLayerHdr.GetSrcAddress());
              SetBackUp();
              if ((m_numBackUp == 0) && (m_state == VirtualLayerHeader::LEADER)) {
                CancelTimers();
                m_timerHeartbeat.Schedule(Seconds(0.0));
              }
              if (m_state == VirtualLayerHeader::NONLEADER) {
                if (m_backUp > 0) {
                  if (virtualLayerHdr.GetBackUp() == NUM_BACKUP);
                  else if (m_backUp != 1) m_backUp--;
                }
              }
              break;
            default:;
          }
        }
        /*
         * Funcion: Envia mensajes de tipo Virtual hacia otros nodos de forma Unicast y Broadcast
         */
        bool VirtualLayerNetDevice::SendPkt(VirtualLayerHeader::type msgType,
VirtualLayerHeader::subType msgSubType, Ipv4Address ipDst, Address macDst) {
          Ptr<Packet> pkt = Create<Packet>();
          NS_LOG_INFO("Generando paquete de la VNLayer en el nodo " << m_node->GetId() << "
para " << macDst << ", enviándolo hacia el NetDevice...");
          Ipv4Header ipv4Hdr = Ipv4Header();
          ipv4Hdr.SetSource(m_ip);
          ipv4Hdr.SetDestination(ipDst);
          pkt->AddHeader(ipv4Hdr);
          VirtualLayerHeader virtualLayerHdr = VirtualLayerHeader();
          virtualLayerHdr.SetType(msgType);
          virtualLayerHdr.SetSubType(msgSubType);
          virtualLayerHdr.SetRegion(m_region);
          virtualLayerHdr.SetSrcAddress(m_ip);
          virtualLayerHdr.SetSrcMAC(m_mac);
          virtualLayerHdr.SetDstAddress(ipDst);
          virtualLayerHdr.SetDstMAC(macDst);
          virtualLayerHdr.SetRol(m_state);
          virtualLayerHdr.SetLeaderAddress(m_leaderAddress);
          virtualLayerHdr.SetLeaderMAC(m_leaderMAC);
          virtualLayerHdr.SetTimeInState(m_timeInState);
          virtualLayerHdr.SetNumBackUp(m_numBackUp);
          virtualLayerHdr.SetBackUp(m_backUp);
          pkt->AddHeader(virtualLayerHdr);
          m_txTrace(pkt, m_node->GetObject<VirtualLayerNetDevice> (), GetIfIndex());
          NS_LOG_DEBUG("VirtualLayerNetDevice::Send " << m_node->GetId() << " " << *pkt);
          bool ret = m_netDevice->Send(pkt, macDst, 0);
          return ret;
        }
        void VirtualLayerNetDevice::SendAODV(VirtualLayerHeader::type, Ipv4Address ip) {
        }
}
```

APÉNDICE D

Implementación en NS3 de VNLayer y RPGM

```
#include <fstream>
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/aodv-module.h"
#include <iostream>
#include "ns3/ocb-wifi-mac.h"
#include "ns3/wifi-80211p-helper.h"
#include "ns3/wave-mac-helper.h"
#include "ns3/virtual-layer-module.h"
#include "ns3/flow-monitor-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("VNLAYER");
 void showPosition (Ptr<Node> node, double deltaTime)
 {
  uint32_t nodeId = node->GetId ();
  Ptr<MobilityModel> mobModel = node->GetObject<MobilityModel> ();
  Vector3D pos = mobModel->GetPosition ();
  Vector3D speed = mobModel->GetVelocity ();
  std::cout << "At " << Simulator::Now ().GetSeconds () << " node " << nodeId
  << ": Position(" << pos.x << ", " << pos.y << ", " << pos.z
  << "); Speed(" << speed.x << ", " << speed.y << ", " << speed.z
  << ")" << std::endl;
  Simulator::Schedule (Seconds (deltaTime), &showPosition, node, deltaTime);
 }
void ReceivePacket(Ptr<Socket> socket) {
    NS_LOG_UNCOND("\n****Received one packet!****\n");
}
static void GenerateTraffic(Ptr<Socket> socket, uint32_t pktSize,
        uint32_t pktCount, Time pktInterval) {
    if (pktCount > 0) {
        socket->Send(Create<Packet> (pktSize));
        Simulator::Schedule(pktInterval, &GenerateTraffic,
                socket, pktSize, pktCount - 1, pktInterval);
    } else {
        socket->Close();
    }
}
void StartVLProtocol(Ptr<VirtualLayerNetDevice> vlNetDev, Ipv4Address ip, Vector2D scn1,
Vector2D scn2) {
    NS_LOG_UNCOND("\nIniciando Protocolo del nodo con IP: " << ip);
    vlNetDev->Start(ip, scn1, scn2);
}
// Prints actual position and velocity when a course change event occurs
static void
CourseChange(std::ostream *os, std::string foo, Ptr<const MobilityModel> mobility) {
    Vector pos = mobility->GetPosition(); // Get position
```

```
    Vector vel = mobility->GetVelocity(); // Get velocity
    // Prints position and velocities
    *os << Simulator::Now() << " POS: x=" << pos.x << ", y=" << pos.y
          << ", z=" << pos.z << "; VEL:" << vel.x << ", y=" << vel.y
          << ", z=" << vel.z << std::endl;
}
int main(int argc, char *argv[]) {
    std::string phyMode("OfdmRate6MbpsBW10MHz");
    uint32_t packetSize = 1000; // bytes
    uint32_t numPackets = 100;
    double interval = 1.0; // seconds
    bool verbose = false;
    std::string traceFile = "/home/crc/repos/ns-3-allinone/ns-3-
dev/Tesis/MOV10nod.ns_movements";//home/victor/Escritorio/Implementacion_ns2/Escenarios_sim
ulacion/Centro1/mobility.tcl";
    std::string logFile = "/home/crc/repos/ns-3-allinone/ns-3-
dev/Tesis/centro1.log";//home/victor/Escritorio/Implementacion_ns2/Escenarios_simulacion/Centro1/
centro1.log";
    uint32_t scnLong = 50; // The long of the simulation scenario
    uint32_t scnWidth = 50; // The width of the simulation scenario
    uint32_t scnRows = 2; // The number of rows of the simulation scenario
    uint32_t scnColumns = 2; // The number of columns of the simulation scenario
    double duration = 1000;
    double deltaTime = 100
    CommandLine cmd;
    cmd.AddValue("phyMode", "Wifi Phy mode", phyMode);
    cmd.AddValue("packetSize", "size of application packet sent", packetSize);
    cmd.AddValue("numPackets", "number of packets generated", numPackets);
    cmd.AddValue("interval", "interval (seconds) between packets", interval);
    cmd.AddValue("verbose", "turn on all WifiNetDevice log components", verbose);
    cmd.AddValue("traceFile", "Ns2 movement trace file", traceFile);
    cmd.AddValue ("deltaTime", "time interval (s) between updates (default 100)", deltaTime);
    cmd.AddValue("duration", "Duration of Simulation", duration);
    cmd.AddValue("logFile", "Log file", logFile);
    cmd.AddValue("scnLong", "The long of the simulation scenario", scnLong);
    cmd.AddValue("scnWidth", "The width of the simulation scenario", scnWidth);
    cmd.AddValue("scnRows", "The number of rows of the simulation scenario", scnRows);
    cmd.AddValue("scnColumns", "The number of columns of the simulation scenario", scnColumns);
    cmd.Parse(argc, argv);
    // Convert to time object
    Time interPacketInterval = Seconds(interval);
NodeContainer c;
    c.Create(10); //NUmero de Nodos
    // open log file for output
    std::ofstream os;
    os.open(logFile.c_str());
    // Configure callback for logging
    Config::Connect("/NodeList/*/$ns3::MobilityModel/CourseChange",
          MakeBoundCallback(&CourseChange, &os));
    // The below set of helpers will help us to put together the wifi NICs we want
    YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
    YansWifiChannelHelper wifiChannel;
    // 802.11p 5.9 GHz, ITU R-1441 Loss Model
    wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
    wifiChannel.AddPropagationLoss("ns3::ItuR1411LosPropagationLossModel", "Frequency",
DoubleValue(5.9e9));
    Ptr<YansWifiChannel> channel = wifiChannel.Create();
    wifiPhy.SetChannel(channel);
    // ns-3 supports generate a pcap trace
    wifiPhy.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11);
```

```
   // Set Tx Power and Detection Th
   wifiPhy.Set("EnergyDetectionThreshold", DoubleValue(-96.0));
   wifiPhy.Set("TxPowerStart", DoubleValue(10.0206));
   wifiPhy.Set("TxPowerEnd", DoubleValue(10.0206));
   NqosWaveMacHelper wifi80211pMac = NqosWaveMacHelper::Default();
   Wifi80211pHelper wifi80211p = Wifi80211pHelper::Default();
   LogComponentEnable("VirtualLayerNetDevice", LOG_LEVEL_INFO);
   if (verbose) {
      wifi80211p.EnableLogComponents(); // Turn on all Wifi 802.11p logging
   }
   wifi80211p.SetRemoteStationManager("ns3::ConstantRateWifiManager",
         "DataMode", StringValue(phyMode),
         "ControlMode", StringValue(phyMode));
   NetDeviceContainer devices = wifi80211p.Install(wifiPhy, wifi80211pMac, c);
   MobilityHelper mobility;
   mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
   mobility.SetMobilityModel("ns3::ConstantVelocityMobilityModel");
   mobility.Install(c);
   InternetStackHelper internet;
   AodvHelper aodv;
   Ipv4ListRoutingHelper list;
   //Routing protocol AODV --> Priority 100
   list.Add(aodv, 100);
   //internet.SetRoutingHelper(list);
   internet.Install(c);
   // Adding VNLayer to the NetDevices
   VirtualLayerHelper vnlayer;
   NetDeviceContainer vn1 = vnlayer.Install(devices);
   Ipv4AddressHelper ipv4;
   NS_LOG_INFO("Assign IP Addresses.");
   ipv4.SetBase("10.1.1.0", "255.255.255.0");
   Ipv4InterfaceContainer ifCont = ipv4.Assign(vn1);
   TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
   Ptr<Socket> recvSink = Socket::CreateSocket(c.Get(0), tid);
   InetSocketAddress local = InetSocketAddress(Ipv4Address::GetAny(), 80);
   recvSink->Bind(local);
   recvSink->SetRecvCallback(MakeCallback(&ReceivePacket));
   Ptr<Socket> source = Socket::CreateSocket(c.Get(1), tid);
   InetSocketAddress remote = InetSocketAddress(Ipv4Address("255.255.255.255"), 80);
   source->SetAllowBroadcast(true);
   source->Connect(remote);
   Vector2D scn1 = Vector2D(scnLong, scnWidth); //Long, Width
   Vector2D scn2 = Vector2D(scnRows, scnColumns); //Rows, Columns
// Calculate Throughput using Flowmonitor
      bool enableFlowMonitor = true;
      FlowMonitorHelper flowmon;
      Ptr<FlowMonitor> monitor = flowmon.InstallAll();
      if (enableFlowMonitor)
      {
       monitor = flowmon.InstallAll ();
      }
   for (int i = 0; i < 10; i++) {
      c.Get(i)->GetObject<MobilityModel>()->SetPosition(Vector(0.0, 0.0, 1.5));//Creamos una altura
de 1.5
      Simulator::ScheduleWithContext(c.Get(i)->GetId(), Seconds(1.0), &(StartVLProtocol),
dynamic_cast<VirtualLayerNetDevice*> (PeekPointer(vn1.Get(i))), ifCont.GetAddress(i, 0), scn1,
scn2);
   }
   // Create Ns2MobilityHelper with the specified trace log file as parameter
   Ns2MobilityHelper ns2 = Ns2MobilityHelper(traceFile);
```

```
   ns2.Install ();
   Simulator::Stop (Seconds (120));
   Simulator::Run();
if (enableFlowMonitor)
{
monitor->CheckForLostPackets ();
monitor->SerializeToXmlFile("VNLAYER.flowmon", true, true);
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);
std::cout << "Flow " << i->first << " (" << t.sourceAddress << " -> " << t.destinationAddress <<
")\n";
std::cout << " Tx Bytes: " << i->second.txBytes << "\n";
std::cout << " Rx Bytes: " << i->second.rxBytes << "\n";
std::cout << " Throughput: " << i->second.rxBytes * 8.0 / (i->second.timeLastRxPacket.GetSeconds()
- i->second.timeFirstTxPacket.GetSeconds())/1024/1024 << " Mbps\n";
}
}
Simulator::Destroy();
NS_LOG_INFO ("Done.");
}
```

BIBLIOGRAFÍA

[1] Koukoumidis, E., Lymberopoulos, D., Strauss, K., Liu, J., & Burger, D. (2012). Pocket cloudlets. *ACM SIGPLAN Notices*, *47*(4), 171-184.

[2] Valls Hermida, Eduardo. "Estudio de parámetros de conectividad y topología en redes Ad-Hoc con distintos patrones de movilidad." (2006).

[3] Caldas Calle, Luis Alberto, and Juan Carlos Zaruma Villamarín. "Implementación de un ambiente de simulación basado en software libre para el estudio de la provisión de servicios de comunicaciones en redes vehiculares Ad-Hoc mediante el uso de nodos móviles virtuales." (2013).

[4] Hurtado, Juan Pablo, and Edgar Patricio Siguenza. "Análisis del uso de nodos móviles virtuales para procesos de encaminamiento en redes vehiculares Ad-Hoc." (2013).

[5] González-Ripoll Cerezo, Pablo. "Estudio del simulador de redes vehiculares veins." (2012).

[6] The Network Simulator ns2, http://isi.edu/nsnam/ns, Septiembre 2014.

[7] The Network Simulator NS-3, http://www.nsnam.org/, Septiembre 2014.

[8] Bonnmotion: A mobility scenario generation and analysis tool, University of Osnabrück, http://sys.cs.uos.de/bonnmotion/ , Julio 2013.

[9] Simulation of Urban Mobility, http://sumo-sim.org/userdoc, Septiembre 2014.

[10] Objective Modular Network Testbed in C++, http://www.omnetpp.org/, Septiembre 2014.

[11] Cano, Juan-Carlos, et al. "Redes Inalámbricas Ad Hoc como Tecnología de Soporte para la Computacion Ubicua." (2008).

[12] Barriga Vázquez, W. E., & Zúñiga Calle, F. P. (2006). Tecnologías inalámbricas de corto alcance: zigbee y bluetooth.

[13] Manuel, Oscar Josué Calderón Cortés-Víctor, and Quintero Florez–Guefry Leider Agredo Mendez. "BLUETOOTH Y LAS REDES AD-HOC MOVILES."

[14] Camps-Mur, Daniel, Andres Garcia-Saavedra, and Pablo Serrano. "Device-to-device communications with Wi-Fi Direct: overview and experimentation."*Wireless Communications, IEEE* 20.3 (2013).

[15] Alliance, Wi-Fi. "Wi-Fi CERTIFIED Wi-Fi Direct®: Personal, portable Wi-Fi® to connect devices anywhere, anytime" http://www.wi-fi.org/certification, September 2014

[16] Doumenc, Hélène. "Estudios comparativo de protocolos de encaminamiento en redes VANET." (2008).

[17] Bali, Samer, and Ing Klaus Jobmann. "Routing Protocols for Ultra-Wideband Mobile Ad Hoc Networks." *IEEE Sarnoff Symposium 2008 (student paper/poster session proceeding)*. 2008.

[18] Aiello, G. Roberto, and Gerald D. Rogerson. "Ultra-wideband wireless systems." *Microwave Magazine, IEEE* 4.2 (2003).

[19] López-Nores, M., Blanco-Fernández, Y., Pazos-Arias, J., Bravo-Torres, J., & Ordóñez-Morales, E. (2014, February). Leveraging ad-hoc networking and mobile cloud computing to exploit short-lived relationships among users on the move. In *Proceedings of International Conference on Intelligent Cloud Computing (ICC 2014)*.

[20] Redes de comunicaciones de la telefonía móvil a internet, Jordi Casademont, coord.. Paginas 88-90, 2010.

[21] Introduction to Wireless & Mobile Systems, Third Edition, Dharma Prakash Agrawal, Quing-An Zeng. Paginas 317-335, 2011.

[22] Mobile Ad Hoc Networks Current Status of Future Trends, Jonathan Loo, Jaime Lloret Mauri, Jesus Hamilton Ortiz. paginas 8- 22, 2012.

[23] Mohseni, S., Hassan, R., Patel, A., & Razali, R. (2010, April). Comparative review study of reactive and proactive routing protocols in MANETs. In Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on (pp. 304-309). IEEE

[24] Ad Hoc Networks Technologies and Protocols, Prasant Mohopatra, Srikanth Krishnamurthy. Paginas 68-75, 2005.

[25] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In Proceedings of ACM SIGCOMM, paginas 234–244, 1994.

[26] C. Hedrick. Routing Information Protocol. RFC 1058, 1988.

[27] J. Moy. OSPF version 2. RFC 2328, 1998.

[28] D. Bertsekas and R. Gallager. Data Networks. Prentice Hall, 1992.

[29] S. Murthy and J. J. Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. ACM/Baltzer Mobile Networks and Applications, 1(2): 183–197, 1996.

[30]T. Clausen et al. Optimized Link State Routing Protocol.http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-11.txt.pdf, July 2003.

[31] C. E. Perkins, E. Belding-Royer, and S. R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing, http://www.ietf.org/rfc/rfc3561.txt.pdf, July 2003.

[32] C. E. Perkins and E. M. Royer. Ad Hoc On-Demand Distance Vector Routing. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), pages 90–100, 1999.

[33] D. Johnson and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In T. Imielinski and H. Korth, editors, Mobile computing, chapter 5. Kluwer Academic, 1996.

[34] D. B. Johnson, D. A. Maltz, Y. Hu, and J. G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR).

[35] Z. Haas and M. Pearlman. The Performance of Query Control Schemes for the Zone Routing Protocol. IEEE/ACM Transactions on Networking, 9(4):427–38, 2001

[36] M. Pearlman and Z. Haas. Determining the Optimal Configuration for the Zone Routing Protocol. IEEE Journal on Selected Areas in Communications, 17(8): 1395–1414, 1999.

[37] C. Santivanez, R. Ramanathan, and I. Stavrakakis. Making Link-State Routing Scale for Ad Hoc Networks. In Proceedings of ACM MobiHoc, pages 22–32, 2001.

[38] Estigarribia, H. (2012). Mobile cloud computing y su relación con aplicaciones móviles y aplicaciones sensibles al contexto (Doctoral dissertation, Facultad de Informática).

[39]RTX4100_Application_Note_SoftAP_AN8,http://www.rtx.dk/WiFi_Documentation-4028.aspx, Septiembre 2014.

[40]VINT Project: Virtual InterNetwork Testbed, http://www.isi.edu/nsnam/vint/index.html , septiembre de 2014.

[41] Pérez Hueso, José Manuel "Protocolos de encaminamiento ip en dispositivos móviles para crear redes Manets"

[42] SPORANGIUM, Grupo SSI, disponible en: http://gssi.det.uvigo.es/menu-sporangium-retos

[43] Camp, T., Boleng, J., & Davies, V. (2002). A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, *2*(5), 483-502.

[44] Sánchez Landin, J. C. (2012). *Implementación de modelos de movilidad para redes móviles Ad hoc* .

[45] Jabbari, B., Zhou, Y., & Hillier, F. (1998, May). Random walk modeling of mobility in wireless networks. In *Vehicular Technology Conference, 1998. VTC 98. 48th IEEE* (Vol. 1, pp. 639-643). IEEE.

[46] Biomo, J. D. M. M., Kunz, T., & St-Hilaire, M. (2014, May). An enhanced Gauss-Markov mobility model for simulations of unmanned aerial ad hoc networks. In *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP* (pp. 1-8). IEEE.

[47] Sánchez, M., & Manzoni, P. (2001). ANEJOS: a java based simulator for ad hoc networks. *Future generation computer systems*, *17*(5), 573-583.

[48] Hong, X., Gerla, M., Pei, G., & Chiang, C. C. (1999, August). A group mobility model for ad hoc wireless networks. In *Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems* (pp. 53-60). ACM.

[49] Zhao, C., & Sichitiu, M. L. (2010, June). N-body: Social based mobility model for wireless ad hoc network research. In *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on* (pp. 1-9). IEEE.

[50] Bai, F., & Helmy, A. (2004). A survey of mobility models. *Wireless Adhoc Networks. University of Southern California, USA*, *206*.

[51] Helgason, O., Kouyoumdjieva, S., & Karlsson, G. (2013). Opportunistic communication and human mobility.

[52] Einstein, A. (1956). *Investigations on the Theory of the Brownian Movement*. Courier Dover Publications.

[53] Bravo-Torres, J. F., López-Nores, M., Blanco-Fernández, Y., Pazos-Arias, J. J., & Ordóñez-Morales, E. F. (2014). VaNetLayer: A virtualization layer supporting access to web contents from within vehicular networks. *Journal of Computational Science*.

[54] Bravo-Torres, J. F., Lopez-Nores, M., & Bianco-Fernandez, Y. (2012, June). On the use of virtual mobile nodes with real-world considerations in vehicular ad hoc networks. In *Communications (COMM), 2012 9th International Conference on* (pp. 193-196). IEEE.

[55] Wu, J. (2011). *A simulation study on using the virtual node layer to implement efficient and reliable manet protocols* (Doctoral dissertation, The City University of New York).

[56] TraNS (Traffic and Network Simulator environment). Disponible en: http://lca.epfl.ch/projects/trans , Enero de 2015.

[57] Martín Bermejo, D. (2007). Comparación de tiempos de trayectos metro-a pie-bici en la zona urbana de Barcelona.

[58] Martinez, D. M. A., & Kelsey, R. J. R. Redes Inalámbricas Enmalladas Metropolitanas.

[59] Arellano, C. E. C., & Ramos, V. M. R. Análisis de protocolos de encaminamiento para redes inalámbricas tipo malla en modo infraestructura.

[60] A SHORT SURVEY OF SYNTHETIC MOBILITY MODELS, Athanasios Bamis.http://www.eng.yale.edu/enalab/courses/2006f/eeng460a/homeworks/Mobility_Models.htm#_Toc146577914 , Enero 2015.

[61] Prochet, T. C., & Silva, M. J. P. D. (2008). Proxêmica: as situações reconhecidas pelo idoso hospitalizado que caracterizam sua invasão do espaço pessoal e territorial; Proxémica: las situaciones reconocidas por el anciano hospitalizado que caracterizan la invasión de su espacio personal y territorial. *Texto & contexto enferm*, *17*(2), 321-326.