UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

AREA TÉCNICA

TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Implementación de un protocolo de enrutamiento para dispositivos móviles en una herramienta de simulación de redes.

TRABAJO DE TITULACIÓN

Autora: Vargas Ríos, Priscila Yosselyn

Director: Dr. Torres Tandazo, Rommel Vicente

LOJA – ECUADOR

2017

Septiembre, 2017

APROBACIÒN DEL DIRECTOR DEL TRABAJO DE TITULACIÒN

Doctor.
Rommel Vicente Torres Tandazo
DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de titulación: **Implementación de un protocolo de enrutamiento para dispositivos móviles en una herramienta de simulación de redes**, realizado por **Vargas Ríos Priscila Yosselyn**, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, Septiembre del 2017.

f).....

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

"Yo, **Vargas Ríos Priscila Yosselyn**, declaro ser autora del presente trabajo de titulación: "Implementación de un protocolo de enrutamiento para dispositivos móviles en una herramienta de simulación de redes", de la Titulación de Sistemas Informáticos y Computación, siendo el Dr. Rommel Vicente Torres Tandazo director del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además, certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: "Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad".

f)……..
Autor: Priscila Yosselyn Vargas Ríos
Cédula: 1104183247

DEDICATORIA

A Dios, por darme la vida y la salud para alcanzar todos mis logros personales, especialmente éste que es el resultado de esfuerzo, dedicación y muchas noches de desvelo.

A mi madre Guicela, por apoyarme en cada momento de mi vida, por sus sabios consejos, su confianza depositada en mí, por la perseverancia que me infundía, por su motivación constante que me brindaba cada noche con una llamada cuando me quería dar por vencida, pero más que nada, por su infinito amor.

A mi hermana Verónica, por el apoyo incondicional, por sus consejos, por su compañía diaria en los mejores y peores días y por ser el mejor ejemplo de una hermana mayor y de la cual aprendí errores y aciertos y especialmente por su cariño y amor.

A mi padre Manuel, por su apoyo y por su gran amor.

A mis docentes por su apoyo y motivación para la culminación de este trabajo de titulación.

Finalmente a mis familiares y amigos, que marcaron cada fase de mi vida universitaria, y que me ayudaron en asesorías y dudas para la realización del presente trabajo de titulación.

Con cariño,

Priscila.

AGRADECIMIENTO

Quiero agradecer a la docencia de la Universidad Técnica Particular de Loja por su sabia enseñanza brindada en el transcurso de mi vida universitaria ya que sin ella no estaría donde estoy, especialmente al Ing. Rommel Torres por todo su apoyo, enseñanza, tiempo y confianza depositada en mí, en el transcurso de la realización de este trabajo de titulación y también a mis asesores Ing. Patricia e Ing. Manuel por todos los consejos brindados y la confianza depositada en mí.

También agradezco a mis padres por estar todos los días junto a mí, especialmente en los difíciles y apoyarme en todas mis decisiones personales y universitarias. Y agradezco a mi hermana por cuidarme, soportarme y consolarme cuando más lo necesitaba.

Finalmente, pero en primer lugar agradezco a Dios por darme salud y vida.

Con mucho cariño

Priscila

TABLA DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

RESUMEN

En la actualidad lo seres humanos cuentan con un sin número de dispositivos móviles que con el trascurso del tiempo van evolucionando en potencia, autonomía, etc. Estos dispositivos poseen diferentes formas de comunicación: redes inalámbricas, bluetooth, red de telefonía, etc. Lo que admite ser usados para configurar redes MANET.

Las redes MANET son una colección de nodos inalámbricos móviles que se comunican entre sí de manera espontánea sin necesidad de ninguna infraestructura establecida. Cada nodo puede trabajar como emisor, receptor o enrutador. De esta manera los protocolos de enrutamiento son necesarios ya que permiten la comunicación entre los nodos móviles.

En el presente trabajo de titulación se muestra la implementación de los protocolos Backup Clúster Head Protocol (BCHP) y Clúster Based Routing Protocol (CBRP) en la herramienta de simulación NS2. Estos protocolos son protocolos de enrutamiento jerárquico que dividen la red en subconjuntos de nodos llamados clúster, cada nodo Jefe Clúster es utilizado para concentrar y distribuir la información generada dentro del clúster. Además presenta la comparación de la implementación de los protocolos en distintas versiones de la herramienta NS2.

PALABRAS CLAVE: BCHP, CBRP, AODV, DSR, DSDV, NS2, Clúster, MANET, JC, SA, enrutador, protocolo.

ABSTRACT

At present human beings have a number of mobile devices that with the passage of time evolve in power, autonomy, etc. These devices have different forms of communication: wireless networks, bluetooth, telephone network, etc. What it admits to be used to configure MANET networks.

MANET networks are a collection of mobile wireless nodes that spontaneously communicate with each other without the need for any established infrastructure. Each node can work as an emitter, receiver, or router. In this way the routing protocols are necessary since they allow the communication between the mobile nodes.

In the present titling work we show the implementation of the Backup Cluster Head Protocol (BCHP) and Cluster Based Routing Protocol (CBRP) protocols in the NS2 simulation tool. These protocols are hierarchical routing protocols that divide the network into subsets of nodes called clusters, each cluster node is used to concentrate and distribute the information generated within the cluster. It also presents the comparison of the implementation of the protocols in different versions of the NS2 tool.

KEYWORDS: BCHP, CBRP, AODV, DSR, DSDV, NS2, Cluster, MANET, JC, SA, router, protocol.

CAPITULO 1

INTRODUCCION

INTRODUCCIÓN

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 Smartphone, tabletas y otros dispositivos móviles han permitido transformar la forma de comunicarse con las demás personas (Carrión & Delgado, 2015). Uno de los principales factores para el crecimiento de las redes inalámbricas es la movilidad de sus usuarios, que permite reducir significativamente los costes de mantenimiento.

Así, las redes sin infraestructura o Ad-Hoc (MANET) aprovechan el crecimiento de memoria y procesamiento de los dispositivos móviles para eliminar el punto de acceso centralizado (Valverde, 2012), y por tal motivo los dispositivos que se encuentren dentro de un rango establecido podrán comunicarse directamente sin necesidad de un punto de acceso. Para la comunicación entre los dispositivos es necesario tener un conjunto de reglas que permitan el intercambio de información, siendo necesarios los protocolos de enrutamiento en el presente trabajo de titulación.

Así mismo existen varios simuladores de redes: Opnet, Omnet ++, Network Simulator, Glomosim, etc. Sin embargo el simulador Network Simulator 2 o NS2 se ha convertido en un estándar debido a su amplia utilización entre la comunidad docente e investigadora del área de redes de computadores (Triviño, Casilari, & Ariza, 2006). Su estructura permite obtener una visión global de las redes, que facilita la relación de conceptos de distintas áreas como podría ser la propagación de señales en medios inalámbricos con el desarrollo de nuevos mecanismos de comunicación.

1.1. Definición del Problema

La manera habitual de utilizar redes inalámbricas es cuando los clientes móviles se conectan a estaciones base, estas estaciones poseen un área de cobertura en la que los usuarios pueden comunicarse entre sí sin necesidad de una estación base.

Las redes móviles MANET, están diseñadas para brindar rápidas y eficientes comunicaciones sin necesidad de que exista una infraestructura de red ya que se basa en una topología descentralizada. Esta topología elimina el área de cobertura de las redes habituales y se compone de propios nodos móviles autónomos comunicándose entre sí por enlaces inalámbricos. Debido a este nuevo reto de control de red, se debe utilizar protocolos de enrutamiento que permitan la comunicación entre nodos móviles.

1.2. Objetivos

1.2.1. Objetivo General

Evaluar protocolos de enrutamiento en redes móviles Ad-Hoc, mediante la herramienta de simulación NS-2.

1.2.2. Objetivos Específicos

- Analizar el funcionamiento de NS2 como herramienta de simulación.
- Determinar el proceso para la inclusión de un nuevo módulo o protocolo en NS2
- Implementar un protocolo de comunicación en NS2

1.3. Estructura de la tesis

Con la finalidad de que el presente trabajo de titulación posea la descripción de cada capítulo se realizó una distribución de los contenidos y desarrollo del mismo de la siguiente manera:

El capítulo 2 corresponde al Estado del Arte, en el que se indagan los conceptos de protocolos de enrutamiento, redes móviles y herramienta de simulación, los cuales son base fundamental para la elaboración de la presente investigación.

El capítulo 3 presenta los protocolos de enrutamiento jerárquico CBRP y BCHP a implementar con sus respectivos paquetes, tablas y operaciones que utilizan para mejorar la disponibilidad de la red.

El capítulo 4 muestra la estructura del protocolo de enrutamiento jerárquico BCHP, que es una mejora del protocolo CBRP. El presente trabajo pretende hacer la comparación de estos protocolos, además se presenta los archivos modificados, la estructura del protocolo y la definición de algunas clases a utilizar en el protocolo BCHP.

El capítulo 5 presenta el análisis y discusión de resultados. En este apartado se muestra el proceso de simulación que se realiza, seguidamente se realiza la discusión de resultados en la herramienta de simulación NS2.35 con sus respectivos antecedentes de simulación, los cuales definen los escenarios propuestos para el trabajo de titulación. Además plasma la comparación de la implementación del protocolo en las versiones de NS2.35 y NS2.34 con sus respectivos antecedentes de simulación entre versiones de la herramienta.

Por último se presentan las conclusiones generales y los trabajos futuros del trabajo de titulación. Además de Anexos donde se plasman los manuales de instalación, configuración de la herramienta NS2 y código utilizados para generar los resultados.

CAPITULO 2

ESTADO DEL ARTE

2. Estado del Arte

2.1. Protocolos de enrutamiento

El enrutamiento o encaminamiento se refiere a la selección del camino en una red de computadoras para enviar paquetes a una red destino. El enrutador toma decisiones en función de la dirección IP de destino del computador donde se dirigen los paquetes enviados, todos los dispositivos intermedios utilizan esta dirección IP para poder guiar el paquete de manera confiable a su destino correcto (Garzón, 2013)**.**

Los protocolos de enrutamiento son algoritmos útiles que definen distintos mecanismos de comunicación entre enrutadores, permitiendo que compartan información entre ellos (Garzón, 2013)**.** Además de elaborar y mantener las tablas de enrutamiento (Medina & Macas, 2013) (Jiménez, Zerna, Chávez, & Basurto, 2011)**,** donde se encuentran las rutas de nodos vecinos para el envió de paquetes, por ende también determina la mejor ruta para llegar a un dispositivo elegido.

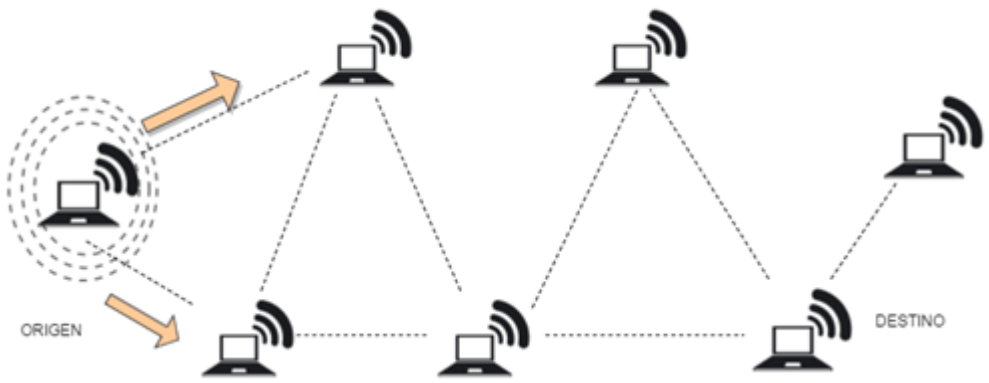


Figura 1: Transmisión de paquetes del origen al destino

Fuente: (Criollo & Ruilova, 2013)

Elaboración: La Autora

La figura 1 muestra la transmisión de paquetes de un origen a un destino, donde cada uno de los nodos debe poseer la información de las rutas, además de los mecanismos para poder descubrir las rutas de los otros nodos cercanos y así poderse enviar paquetes.

Los nodos actúan como enrutadores y debido a la movilidad no pueden disponer de las características de dispositivos que forman una infraestructura fija como Gateways, DHCP (Carrión & Delgado, 2015)**.** Por lo tanto algunos dispositivos deben incluir las funcionalidades del enrutador y el direccionamiento respectivo para una conectividad y envío exitoso.

2.2. Componentes de un protocolo de enrutamiento

- **Tablas de enrutamiento:** también conocidas como estructura de datos, algunos protocolos de enrutamiento usan tablas o base de datos para realizar sus operaciones y guardar información de los nodos.
- **Algoritmos/estrategias:** usan algoritmos para facilitar información de enrutamiento y determinar la mejor ruta.
- **Mensajes/paquetes:** usan varios tipos de mensajes para descubrir enrutadores vecinos, intercambiar información de enrutamiento y otras tareas para conservar información actualizada.

2.2.1. Tablas de enrutamiento o de encaminamiento

Todo enrutador posee una tabla de enrutamiento. Una tabla de enrutamiento es la constitución de una serie de entradas también conocidas como rutas que poseen información acerca de la ruta para el envío de paquetes.

Las entradas de la tabla de enrutamiento vienen de los siguientes orígenes:

- **Redes conectadas:** aquellas rutas de red establecidas por la conexión directa de nodos.
- **Rutas estáticas:** aquella ruta de red fija que indica por dónde se deberá enviar determinado paquete para que llegue a su destino.
- **Rutas dinámicas:** determinada a través de algún protocolo de enrutamiento para intercambiar la información de rutas con sus enrutadores adyacentes.

Las entradas de las tablas de enrutamiento según (Mendoza, 2011) posee la siguiente información:

- **Destino:** red de destino también es la computadora o host de destino. El destino para todas las rutas es 0.0.0.0
- **Mascara de red:** se utiliza en conjunto con el destino de red para delimitar el ámbito de una red.
- **Puerta de enlace:** es la dirección IP del siguiente enrutador al cual se le enviará un paquete.
- **Interfaz:** indica la interfaz LAN (conexión de enrutadores de redes públicas o privadas) que se va a utilizar para poder alcanzar el siguiente enrutador.

- **Métrica:** indica el coste relativo que utiliza la ruta para alcanzar el destino. Las métricas típicas son los saltos, es decir por cuántos enrutadores pasa para llegar al destino.
- **Protocolo:** indica la ruta aprendida en el transcurso del envío del paquete.

2.2.2. Algoritmos o estrategias clásicas

El algoritmo de encaminamiento es aquella parte del software situada en la capa de red, responsable de decidir por cuál línea de salida se comunicará un paquete entrante (Goitia, 2003b). Usualmente es usada en las redes cableadas.

Existen diversos algoritmos de enrutamiento, aquellos más utilizados se los puede clasificar en:

- **Algoritmo Vector Distancia.-** En los algoritmos de Vector Distancia cada enrutador envía periódicamente la tabla de enrutamiento a su enrutador vecino, actualizando así las direcciones en la topología. “La distancia es determinada por el número de saltos que existe desde el emisor hasta el destino y se considera salto a cada enrutador por los que tenga que pasar los paquetes antes de llegar al destino” (Garzón, 2013). Estos algoritmos también son conocidos como Algoritmo Bellman Ford. El algoritmo de Vector Distancia es iterativo, asíncrono y distribuido (Morató, n.d.)**.** Iterativo porque el proceso continúa hasta que la información se ha intercambiado entre nodos vecinos. Asíncrono porque no requiere que los nodos trabajen conjuntamente entre ellos. Y finalmente distribuido porque los nodos reciben información de los nodos vecinos directamente enlazados a él, los calcula y distribuye el resultado del cálculo de regreso a sus nodos vecinos.

  RIP es un protocolo de enrutamiento interno basado en el algoritmo vector distancia, existen dos versiones RIP versión 1 y RIP versión 2. RIP establece un número de 15 saltos máximo y sus actualizaciones son cada 30 segundos cuando la red esté cambiando. La segunda versión de RIP soporta subredes y las direcciones con CIDR (enrutamiento entre dominios sin clases) y además establece mecanismos de autentificación.
  Este protocolo tiene una distancia administrativa máxima de 120 saltos, es decir el grado de confiabilidad del protocolo de enrutamiento. Hay que tener en cuenta que mientras menor sea el valor mejor es el protocolo.

  (Fernandez & Mena, 2004) manifestó:

  > “El algoritmo acumula las distancias de la red de tal manera que puede mantener una base de datos con la información de la topología de la red. Sin embargo, los algoritmos de vector distancia no permiten a un enrutador conocer la topología exacta de toda la red, dado que cada enrutador solamente se comunica con sus nodos vecinos. La tabla

de encaminamiento incluye información acerca del costo total de ruta definido por su métrica, y la dirección lógica del primer enrutador hacia un destino dado”

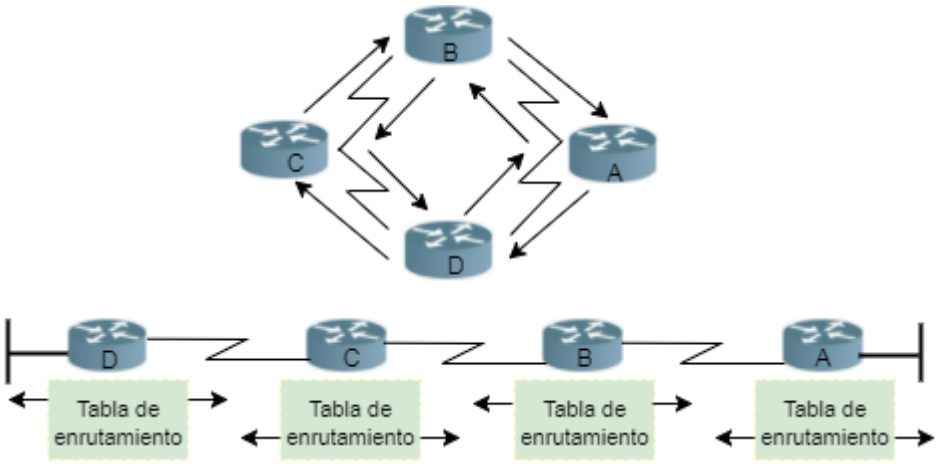


Figura 2: Algoritmo vector distancia

Fuente: (Fernandez & Mena, 2004)

Elaboración: La Autora

Tal como se muestra en la figura 2 cada enrutador envía la tabla de encaminamiento a sus enrutadores vecinos que se encuentran conectados a su alrededor. El enrutador A envía información al enrutador B, y añade un número de saltos, lo que incrementa el vector distancia. Entonces el enrutador B pasa su nueva tabla de encaminamiento hacia el enrutador C. El mismo proceso ocurre en todas las direcciones entre los nodos vecinos que existan.

- **Algoritmo de Estado de enlace.-** este algoritmo mantiene una base de datos compleja ya que posee toda la información de la topología de la red. Se basa en que cada nodo llegue a conocer la topología de la red y los costes de retardos asociados a los enlaces, para que a partir de estos datos se pueda obtener el árbol y la tabla de encaminamiento tras aplicar el algoritmo SPF (camino más corto) y calcular la ruta más corta al grafo de la red (Sevilla, 2011)**.** También se conocen como algoritmos Dijkstra o SPF ("primero la ruta más corta"). Su métrica se basa en el retardo, ancho de banda, carga y disponibilidad (Modesto & González, n.d.)**.**

  OSPF (Open Shortest Path First) es un protocolo de estado de enlace. Su funcionamiento se basa en que cada enrutador conoce los enrutadores vecinos y las direcciones que posee cada uno de estos enrutadores.

  Este protocolo “basa su selección de la mejor ruta en el costo de la misma” (Ulloa, 2007), que depende de los parámetros que posea como retardo, ancho de banda, etc. OSPF

separa las redes muy grandes por áreas más pequeñas para su mejor enrutamiento, luego estas áreas se enrutan entre ellas a través de una área de backbone (principales conexiones troncales de Internet).

"Cada enrutador de la red envía los paquetes "hello" de manera multicast para realizar un seguimiento del estado de los enrutadores vecinos" (Sevilla, 2011)**.** Con este mensaje el enrutador se comunica que existe un vecino.

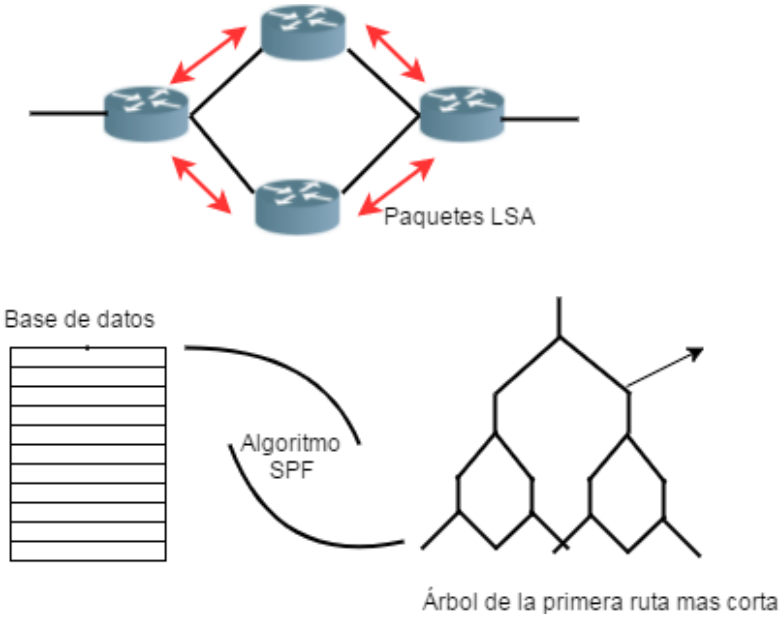


Figura 3: Algoritmo de Estado de Enlace

Fuente: (Fernandez & Mena, 2004)

Elaboración: La Autora

En la figura 3 se muestra como cada enrutador envía publicaciones de estado de enlace (LSA) a sus nodos vecinos, y así se construye la base topológica de datos de la red. Así mismo el algoritmo SPF es utilizado para calcular la ruta más corta en el cual el enrutador individual constituye la raíz. Seguidamente se crea la tabla de encaminamiento**.**

2.2.3. Paquetes

Un paquete es un grupo de información que consta de cadena de bits, es decir datos propios y/o de bits de control, es decir de información de control en la que se especifica la ruta a seguir por medio de la red hasta el destino del paquete. Se organiza según la longitud y el formato determinado dependiente del protocolo de la capa de red que se esté utilizando.

La estructura global de los paquetes en la que es dividida la información se compone a su vez de varias entidades individuales, llamados campos.

En los protocolos de enrutamiento existen muchos paquetes utilizados según el algoritmo elegido. A continuación se nombran los mensajes utilizados en los protocolos implementados: Hello, RREQ (Solicitud de Ruta), RREP (Respuesta de Ruta), Enrutamiento, Error.

A continuación se presenta el formato de paquetes de dos de los protocolos antes mencionados que engloban a los algoritmos de vector distancia y estado de enlace. Cabe mencionar que el formato y los campos de cada paquete varían según el algoritmo.

En la siguiente figura 4 se puede observar el formato del paquete RIP:

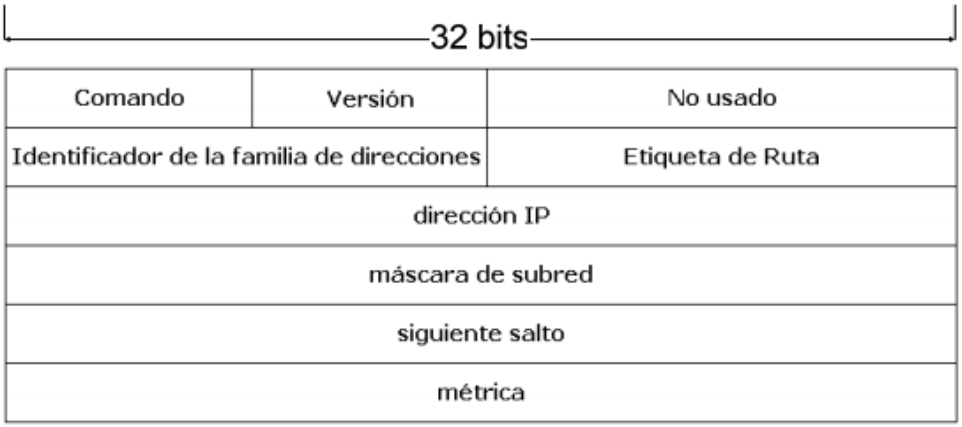


Figura 4: Formato del paquete RIPv2

Fuente: (Ulloa, 2007)

Elaboración: (Ulloa, 2007)

Se describe a continuación cada uno de los campos del paquete:

- **Comando:** este campo indica si el paquete es de solicitud o de respuesta.
- **Versión:** especifica la versión del protocolo.
- **Identificador de la familia de direcciones:** en este campo se especifica el tipo de direcciones que utiliza el protocolo.
- **Etiqueta de ruta:** este campo ayuda a distinguir entre rutas externas o internas.
- **Dirección IP:** este campo especifica la dirección IP de entrada.
- **Máscara de subred:** este campo indica la máscara de subred de la entrada, en caso de ser 0 quiere decir que no se ha especificado la máscara.
- **Siguiente salto:** contiene la dirección IP del próximo salto.

- **Métrica:** este campo contiene el número de saltos del paquete, estableciendo como máximo número 15.

Seguidamente en la figura 5 se podrá observar el formato del paquete OSPF versión 2:

Figura 5: Formato del paquete OSPFv2

Fuente: (Ulloa, 2007)

Elaboración: (Ulloa, 2007)

A continuación se describen los campos del paquete OSPF:

- **Versión:** en este campo se indica la versión del protocolo que se está utilizando.
- **Tipo:** este campo se describe el tipo del paquete que se enviará.
- **Longitud del paquete:** este campo contiene la longitud del paquete incluyendo la cabecera.
- **ID del enrutador:** contiene el identificador del enrutador del origen del paquete (Ulloa, 2007)
- **ID del área:** contiene el identificador del área al que pertenece el paquete.
- **Checksum:** contiene una suma de comprobación que chequea toda la cabecera, excluyendo la autentificación.
- **Tipo de autentificación:** este campo contiene el tipo de autentificación a utilizarse en el paquete.
- **Autentificación:** es una campo de 64 bits que consta con la información para la autentificación

2.3. Redes Móviles

Las redes MANET (Mobile ad hoc network) pertenecen a un grupo de redes inalámbricas, son redes que no dependen de una infraestructura física para su funcionamiento, es decir, cada nodo que está dentro de la red tiene la capacidad de enviar y recibir información de un punto a otro sin necesidad de estar conectados a un punto de acceso (Garzón, 2013)**.** Estas redes son temporales y auto-configurables. "Tienen recursos finitos, es decir ancho de banda, batería, procesamiento que deben ser utilizados eficientemente con el objetivo de mejorar el desempeño de la red" (Perkins, 1997)**.**

El término Ad Hoc en el punto de vista de redes inalámbricas se refiere a redes flexibles (Garzón, 2013)**,** cuando la comunicación se establece entre terminales alejados es preciso que ciertos dispositivos intermedios, miembros de la propia red Ad hoc actúen como enrutadores para que el paquete pueda alcanzar el destino (Triviño et al., 2006)**,** caso contrario el paquete tendrá un pérdida no favorable para la comunicación.

2.3.1. Características de las redes inalámbricas

El propósito de las redes móviles ad hoc es el de apoyar operaciones robustas y eficientes, incorporando un enrutamiento funcional en los nodos móviles. Debido a que las MANET se pueden construir espontáneamente sin necesidad de una infraestructura existente, estas redes presentan restricciones especiales que se deben considerar para el diseño de soluciones (Jaramillo, 2004)**.**

Según Criollo & Ruilova (2013) se describen algunas de las características:

- **Topologías dinámicas**: En las topologías dinámicas, los nodos se mueven libremente en cualquier dirección independientemente de los demás, haciendo que la topología esté en constante cambio.
- **Operación en forma distribuida:** los nodos tienen conocimiento local sobre su entorno y cada nodo conoce la información sobre los nodos vecinos que se encuentran dentro de su radio de transmisión, pero no poseen un conocimiento global de la red (Jaramillo, 2004)
- **Limitaciones de energía:** Las baterías son su fuente de energía, las cuales tienen una reserva de energía limitada, esto hace que sea necesario el ahorro de energía.
- **Seguridad física limitada:** Son susceptibles de tener carencia de seguridad por su conexión inalámbrica y pueden ser atacadas fácilmente.
- **Ancho de banda restringido:** Restricciones impuestas por el canal inalámbrico, tal como acceso múltiple, interferencias, ruido, disponibilidad limitada del espectro, etc

2.4. Protocolos de enrutamiento para MANETS

2.4.1. Proactivo

"Tiene como objetivo mantener las tablas de los nodos permanentemente actualizadas, de forma que cuando un nodo quiera enviar datos a otro, ya disponga de la información necesaria para alcanzar a su destino" (Radicelli, 2013), es decir los terminales de red envían periódicamente información de la topología de red así no se utilice en el intercambio de información. Esto permite una respuesta rápida ante solicitudes de ruta, sin embargo se tiene una sobrecarga importante de red con los mensajes de control.

Los protocolos de enrutamiento en Internet están 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 la grande sobrecarga y a no poder entregar la información de manera más directa a los nodos involucrados (Carrión & Delgado, 2015)**.**

2.4.1.1. DSDV (Destination Sequenced Distance Vector)

Este protocolo DSDV es un ejemplo de protocolo proactivo. DSDV es una adaptación del protocolo convencional RIP (Routing Information Protocol) (Doumenc, 2008)**.** Se añade un nuevo atributo, denominado número de secuencia, el cual permite diferenciar las rutas nuevas de las antiguas, y así enviar la información por las rutas nuevas evitando bucles de enrutamiento (Garzón, 2013). La idea principal es el uso de números de secuencia de destino (Carrión & Delgado, 2015)**,** es decir saber cuántos salto son necesarios para llegar al nodo destino.

DSDV está basado en la actualización de las tablas de enrutamiento, cada nodo perteneciente a la red tiene una tabla de enrutamiento que indica los posibles destinos y el número de saltos que hacen falta atravesar para llevar el paquete desde su origen hasta su destino (Criollo & Ruilova, 2013).

Este protocolo implementa un algoritmo basado en el vector distancia, lo que permite mantener tablas con todos sus destinos accesibles junto con el siguiente salto, la métrica y un número de secuencia de la entrada en la tabla generado por el nodo destino (Doumenc, 2008)**.**

DSDV utiliza una tabla de enrutamiento para almacenar la información lo que puede provocar mayor trabajo para el nodo cuando la cantidad de nodos en la red sea mayor (Mohseni, Hassan, Patel, & Razali, 2010).

2.4.2. Reactivos

Según Carrión & Delgado (2015): “En este tipo de protocolos los nodos actualizan las tablas de encaminamiento solamente en caso de necesitarla, 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 lleva acabo el descubrimiento de la ruta necesaria para encontrar un camino hasta su destino.

Una desventaja de este protocolo es el retardo adicional para descubrir una ruta. Sin embargo, la sobrecarga de encaminamiento se puede reducir hasta cero si ningún nodo debe enviar información (Radicelli, 2013)(Criollo & Ruilova, 2013)**.** La característica principal de este protocolo es que depende del cambio en la topología o alguna movilidad, debido a que los dispositivos no actualizan las tablas de encaminamiento a menos que lo requieran.

Algunos de los ejemplos de protocolo reactivo, es Dynamic Source Routing (DSR) y el protocolo Ad-Hoc on Demand Distance Vector (AODV). A continuación se explica brevemente estos protocolos:

2.4.2.1. DSR (Dynamic Source Routing)

“Este protocolo fue diseñado para restringir el ancho de banda consumido por paquetes de control eliminando las actualizaciones periódicas” (Carrión & Delgado, 2015)**.**
Por otra parte se basa en tener el mejor rendimiento en entornos donde la velocidad móvil de los nodos es baja. Además, como pertenece al protocolo reactivo cada nodo almacena un cache de enrutamiento que contiene las rutas conocidas de los vecinos cercanos, así el momento de enviar un paquete utiliza alguna de ellas o en caso de no tener la ruta, buscarla. Así mismo no requiere ningún mensaje periódico, disminuyendo la sobrecarga por paquetes de control dentro de la red.

DSR tiene dos tareas principales en su funcionamiento: el de búsqueda y el de mantenimiento de rutas. Además “puede funcionar adecuadamente con una red de 200 nodos” (Chalmeta, 2009)**.**

Cuando un nodo desea enviar un paquete el proceso se inicia con un proceso llamado Routing Discovery que consiste en que el nodo envíe un mensaje RREQ sobre la red. En el caso en el que el nodo sea destinatario responde con un mensaje RREQ, transmitiendo al nodo solicitante un mensaje RREP. Además se utiliza un mensaje RERR en caso de que una conexión es interrumpida, de modo que los nodos actualicen su memoria de direccionamiento y no usen el enlace roto.

2.4.2.2. AODV (Ad-Hoc on Demand Distance Vector)

Según Garzón (2013)**:**

“AODV es un protocolo que inicia el proceso de búsqueda de rutas hacia otro nodo en el momento que lo necesita, pero emite mensajes para alertar sobre su presencia dentro de la red por medio de una técnica llamada *Link Layer Feedback* la cual se encarga de que los nodos tengan conocimiento de sus nodos vecinos y de mantener tablas de enrutamiento actualizadas teniendo en cuenta los cambios de topología.”

Este protocolo está basado en el algoritmo Vector-Distancia. La idea principal es mantener y encontrar solo rutas que se necesiten. Utiliza tablas de enrutamiento tradicionales, una entrada por cada destino (Carrión & Delgado, 2015) y además soporta enrutamiento multidifusión.

“La idea es mejorar el protocolo DSDV minimizando el número de paquetes broadcast requeridos para crear rutas” (Doumenc, 2008), el protocolo AODV al ser bajo demanda los nodos que se encuentran en el camino no tienen que participar en el intercambio de tablas ni mantener la ruta.

Cada nodo en una red Ad Hoc con AODV genera tres tipos de mensajes para generar y mantener las rutas, solicitud de ruta (RREQ), respuesta de ruta (RREP) y errores de ruta (RERR) (Torres, 2011) mediante el protocolo UDP.

Como se puede ver, hay una gran similitud entre estos dos protocolos. Ambos son reactivos y proponen una red Ad Hoc. Sin embargo, existen diferencias en la forma de cómo almacenan sus rutas en base a los mecanismos propuestos por los dos protocolos. En el protocolo DSR, cada nodo de la red tiene en su tabla de enrutamiento todas las rutas "salto a salto" desde el origen hasta el nodo de destino, es decir, almacena en su memoria caché todo el camino que pueden ser más de uno, con todos los nodos a través del cual el paquete debe ser enviado. Por el otro lado, AODV usa en su caché, la entrada del nodo máximo para cada destino, es decir, no contiene toda la ruta a un nodo dado, pero si el siguiente salto para la cual el paquete debe pasar.

2.4.3. Jerárquico

“Los protocolos de enrutamiento jerárquico dividen la red en subconjuntos de nodos llamados clúster, en donde un nodo jefe de clúster es utilizado para concentrar y distribuir la información generada dentro del clúster**”** (Criollo & Ruilova, 2013)**.** Está basado en un direccionamiento jerarquizado, lo que reduce los requisitos de memoria de las simulaciones muy grandes.

Cuando las redes crecen también aumentan el número de enrutadores y como consecuencia las tablas de encaminamiento aumentan, y los enrutadores no pueden controlar el aumento de tráfico de red de manera eficiente, lo que provocan cálculos óptimos en mucho más tiempos y mayor tráfico de control para difundir información. Por lo que los protocolos de enrutamiento jerárquico se dividen en varios niveles reduciendo así las tablas de enrutamiento, donde cada enrutador conoce los detalles que enrutan paquetes a destinos de su propio nivel, pero no sabe de la estructura interna de los otros niveles (Gomez, n.d.) (Goitia, 2003)

Según Medina & Macas (2013) se debe considerar como ventajas de los protocolos de enrutamiento:

- La baja demanda para el descubrimiento de rutas.
- Reparación de enlaces rotos a nivel local.
- Optimación de enrutamiento con el acortado de rutas.

Clúster Based Routing Protocol (CBRP) y Backup Clúster Head Protocol (BCHP) son ejemplos de este tipo de protocolos, los cuales son tomados como base para la implementación del trabajo de fin de titulación.

2.4.3.1. Clúster Based Routing Protocol

CBRP “divide los nodos en subconjuntos liderados y mantenidos por un Jefe de Clúster” (Torres, 2011)**.**

Cada uno de los nodos miembros de un clúster conoce todos los nodos vecinos a una distancia de 2 saltos en el caso del protocolo CBRP y cada uno de los nodos pude enviar información a más de un nodo Jefe de Clúster, lo que permite mejorar la selección del mejor camino (Torres, 2011).

Una cabecera de clúster “clústerhead” es elegida por los nodos para mantener la información de los miembros del clúster. Al dividir la red en clúster, el protocolo minimiza las inundaciones de tráfico al momento de descubrir las rutas. Más aún el protocolo toma en consideración la existencia de enlaces unidireccionales y usa estos enlaces tanto para comunicación dentro de los clústers como fuera de ellos (Fernandez & Mena, 2004).

Según Torres (2011) el protocolo CBRP utiliza las siguientes tablas para el enrutamiento respectivo:

- Tablas de Vecinos: esta tabla guarda la información de los enrutadores vecinos, en ese caso si el vecino es un Jefe Clúster o Miembro del Clúster.
- Tabla de Adyacentes: es utilizada por el Jefe Clúster y mantiene la información de clúster vecino y toda su información.

- Base de Datos de la topología de dos saltos: mediante la utilización del paquete HELLO cada nodo envía información de su tabla de enrutamiento, ayudando así la generación de una visión completa de la topología de diámetro de dos saltos del nodo.

2.4.3.2. Backup Clúster Head Protocol

"BCHP es un protocolo de enrutamiento jerárquico que divide los nodos en subconjuntos liderados y mantenidos por un Jefe de Clúster y mejora la disponibilidad de la red por la inclusión de nodos Jefes de Clúster de Respaldos" (Medina & Macas, 2013). Además BCHP es una mejora del protocolo CBRP, debido a que posee nodos llamados Jefe Clúster de Respaldo el cual asume las funciones del nodo principal cuando este tiene algún problema.

BCHP reduce la sobrecarga de la red usando los jefes de clúster que son nodos seleccionados por sus métricas para el envío de los mensajes de encaminamiento a la red. Esta técnica minimiza la cantidad de información que debe transmitirse a todos los nodos de la red y ayuda a controlar los mensajes de broadcast (Torres, 2011). El protocolo utiliza las mismas estrategias que los protocolos que utilizan clúster como CBRP, en el intercambio de información. BCHP no cambia el formato de los paquetes IP. Debido a que solo interactúa con la tabla de enrutamiento de los nodos.

Criollo & Ruilova (2013) menciona:

"Los protocolos deben buscar un compromiso entre el tiempo de la búsqueda de ruta y la sobrecarga de la búsqueda/mantenimiento. Los proactivos suelen tener mejor latencia ya que las rutas se mantienen todo el tiempo, pero puede ser que sufran mayor sobrecarga debido a continuas actualizaciones de caminos, operaciones que consumen tiempo, ancho de banda y energía. Por lo contrario los reactivos suelen tener mayor latencia, porque se intentará buscar caminos entre dos nodos sólo cuando el origen desee comunicarse con el destino, mientras que la sobrecarga es menor, pues las rutas son buscadas sólo si se necesitan. La combinación de las dos aproximaciones anteriores da lugar a los protocolos híbridos, que minimizan la desventaja de los mismos."

2.5. Herramientas de simulación

Las herramientas de simulación son un componente fundamental para el diseño, la implementación y el monitoreo de redes de comunicación ya que permiten predecir el comportamiento de diferentes eventos que pueden afectar el desempeño de la red (Morales, Calle, Tovar, & Cuéllar, 2013)**.** Además de permitir la recreación de diferentes escenarios

reales con el fin de analizar el desempeño sin necesidad de implementar una infraestructura física, evitando así un consumo una enorme cantidad de gastos.

Existen varias consideraciones para la selección correcta de una herramienta de simulación, teniendo en cuenta la disponibilidad de licencias de software, la compatibilidad con el sistema operativo, la disponibilidad de una interfaz gráfica, la capacidad de modelar los escenarios, entre otros aspectos importantes. A continuación se describen algunas de las herramientas más usadas:

2.5.1. Network Simulator 2

NS2 es un simulador de eventos discretos destinado a la investigación de redes. Proporciona soporte para simular protocolos de la capa de enlace como CSMA/CD, protocolos y algoritmos de encaminamiento, protocolos de transporte como TCP y RTP, protocolos multicast, protocolos de aplicación como HTTP, TELNET y FTP. Consiste en dos lenguajes, por una parte el lenguaje C++ que sirve para modelar el comportamiento de los nodos y por otro lado el lenguaje oTcl (Object-oriented Tool Command Languaje) que permite al usuario interactuar con el simulador mediante *script* en el que detalla los parámetros de la topología a simular (Carrión & Delgado, 2015).La unión de estos dos lenguaje forman TclCL que es el lenguaje que interpreta esta herramienta. Además es una herramienta multiplataforma, es decir se encuentra disponible para los sistemas operativos Windows, Unix y Sistemas Mac. Es catalogado con un nivel alto en su curva de aprendizaje y bajo en nivel de características gráficas.

2.5.2. Network Simulator 3

Al igual que su predecesora, NS3 es un simulador de eventos discretos y de código abierto, de la misma manera se basa en el lenguaje C++ para la implementación de los modelos de simulación, “y de forma opcional, partes de la simulación se puede realizar utilizando el lenguaje Python” (Carrión & Delgado, 2015)**.** Tiene como principales objetivos lograr un mayor énfasis en los niveles 2 y 4 del modelo OSI y que su uso sea principalmente educativo (Morales et al., 2013)**.** Sin embargo NS3 no utiliza comandos oTcl para controlar la simulación.

NS3 tiene la capacidad de simular redes cableadas o inalámbricas, modelando los elementos que conforman una red, así como dispositivos finales o nodos centrales. Permite también, la simulación de canales WiFi y demás basados en el estándar IEEE 802.11 (Carrión & Delgado, 2015).

2.5.3. OMNeT++ (Objective Modular Network Testbed in C++)

OMNeT++ es un entorno de simulación de eventos discretos. Su área principal de aplicación es la simulación de redes de comunicaciones y el análisis y evaluación de éstas. Proporciona además un conjunto de herramientas y componentes programados en C++ y cuya interfaz gráfica está basada en la plataforma Eclipse. Puede ser implementada en diferentes plataformas como: Windows, Unix y MAC, soportando diversos compiladores de C++ al igual que ns2 y ns3. “Dedicada principalmente a la creación de simulaciones de red incluyendo no solo redes cableadas e inalámbricas, sino redes de cola, redes de sensores, redes inalámbricas Ad-Hoc, redes fotónicas, etc” (Carrión & Delgado, 2015)**.**

La interfaz gráfica que ofrece la herramienta presenta características de depuración superiores al de otras herramientas (Morales et al., 2013)**.** La configuración de la simulación de la red se lleva a cabo en NED (NEtwork Description), el lenguaje de descripción de la red de OMNeT++ (Weingartner, Vom Lehn, & Wehrle, 2009)**.**

2.5.4. OPNET Modeler

Es una herramienta de simulación orientada a las comunicaciones que permite flexibilidad y escalabilidad en modelos jerárquicos, estos modelos están divididos en tres dominios denominados Red, Nodo y Procesos, los cuales están escritos en C++ y poseen su propio editor (Morales et al., 2013). Ofrece múltiples capacidades de simulación, animación y análisis (Sevilla, 2011)**,** además permite diseñar y analizar los resultados para una posterior comparación por medio de grafos presentados dentro de paneles de análisis.

Posee una curva de aprendizaje alta, debido a que se necesita conocimientos previos en redes y programación. Su licencia es comercial y puede ser implementada en los sistemas operativos de Unix o Windows.

2.5.5. GNS3

“Software gráfico de simulación de red que permite la emulación de redes complejas” (Morales et al., 2013) y pone en marcha simulaciones. GNS3 está vinculada con Dydnamips que permite emular imágenes IOS de Cisco Systems. De una manera mejor explicada permite configurar unidades virtuales como enrutador, switch, host utilizando imágenes de IOS de Cisco, para observar el comportamiento de envío y recepción de información a través de la red.

Además posee una interfaz gráfica muy amigable para el usuario, posee un ambiente libre para descargas y puede estar disponible para sistemas Linux, Windows y Mac.

Tabla 1: Comparación de las herramientas de simulaciones

Nombre	Protocolos	Plataforma	Gráfica de resultados	Licencia	Uso investigativo	Flexibilidad
NS2	TCP/IP, UDP, FTP, RTP, SRN, GPRS, mobile IPv6, RSRV, MPLS, redes Ad Hoc, WLAN, Mobile-IP, UMTS y Wireless	Windows, Linux, Mac	No posee	Libre	Alto	Baja
NS3	IPv4 e IPv6, Wireless y algoritmos de enrutamiento	Windows, Linux, Mac	Aceptable	Libre	Medio	Baja
OMNeT++	Redes cableadas e inalámbricas, redes de cola, redes de sensores, redes inalámbricas Ad-Hoc, redes fotónicas	Windows, Linux	Aceptable	Libre	Alto	Alta
OPNET Modeler	Aloha, TCP/ip y UDP/IP, RIP, UDP, TCP, Ethernet, Fast Ethernet, gigabit, Ethernet y OSPF	Windows, Linux	Buena	Comercial	Alto	Media
GNS3		Windows, Linux, Mac	Limitada	Libre, Comercial	Bajo	Media

Fuente y Elaboración: La Autora

En la tabla 1 se presenta un cuadro comparativo de las principales y más usadas herramientas de simulación para las redes de comunicaciones, de las cuales para la implementación de este trabajo de titulación se ha seleccionado a la herramienta NS2 debido a que posee algunos protocolos ya implementados, además se cuenta con el conocimiento necesario para la implementación de nuevos protocolos y es ampliamente aceptado por la comunidad

mundial. Por otra parte, la herramienta es multiplaforma y se la encuentra de una manera libre y con mucha información en la web para su respectiva instalación.

2.6. Network Simulator 2

Se empezó a desarrollar en 1989 como una variante del ya existente simulador REAL Network Simulator, creado por la Universidad de California. En 1995, bajo la supervisión del proyecto VINT (virtual Inter Network Testbed), acabó en manos de uno de los investigadores y desarrolladores de la Universidad de California en Berkeley (Sevilla, 2011).

Network Simulator 2 (NS2) es un software simulador de eventos libre diseñado para la ayuda en las redes telemáticas y disponible en diversas plataformas como Unix, Windows y sistemas Mac.

Sevilla (2011) afirma:

> "Network simulator permite entre otras cosas trabajar tanto en redes cableadas como redes inalámbricas (simulando wifi etc..), redes vía satélite con una enorme cantidad de protocolos a distintos niveles, la capa de transporte (tcp, udp, etc…), la capa aplicación (ftp, cbr, http, etc…) o la capa de enlace de datos (como el CSMA/CA). Además permite trabajar en los modos Unicast o Multicast y utilizar diversos algoritmos para la planificación de colas (como el DRR (Deficit Round Robin), FIFO (First In First Out), FQ (Encolamiento justo) o SFQ (Encolamiento Estocástico Justo) en caso de que se produzca el cuello de botella en algún nodo"

Además maneja diversos mecanismos de colas que se generan en los enrutadores, tales como DropTail, RED, CQB, algoritmo de Dijkstra, etc (Herrera, 2004).

NS2 consiste en dos lenguajes principales para su correcto funcionamiento. Por un lado C++ define el mecanismo interno, es decir, un *backend* de los objetos de simulación. Por otro lado, el lenguaje orientado a *Objetos Tool Command Language* (OTcl), es decir una interfaz de simulación (Issariyakul & Hossain, 2012). Estos dos idiomas están unidos entre sí mediante TclCL (TCL con clases), mismo que es el lenguaje propio de la NS2.

Una parte importante de NS2 es que cuenta con varios protocolos ya definidos, lo que facilita simulación y lo hace más interesante (Garzón, 2013).

Para realizar una simulación primero debemos indicar al simulador la topología de la red a simular, en la cual se debe especificar: Nodos, Enlaces, Agentes, Aplicaciones y Paquetes.

2.6.1. Estructura de directorio de NS2

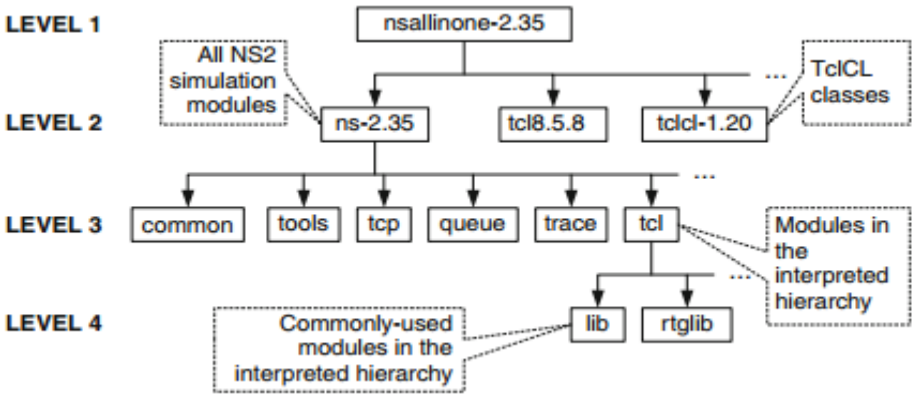


Figura 6: Directorio de NS2

Fuente: (Issariyakul & Hossain, 2012)

Elaboración: (Issariyakul & Hossain, 2012)

Issariyakul & Hossain (2012) describe el directorio de NS2:

> La figura 6 muestra la estructura de directorios en el directorio nsallinone-2.35. El directorio nsallinone-2.35 está en el nivel 1. En el nivel 2, el directorio tclcl-1.20 contiene clases en TclCL (por ejemplo, Tcl, TclObject, TclClass). Todos los módulos de simulación NS2 están en el directorio ns-2.35 en el nivel 2. En adelante, nos referiremos a los directorios ns-2.35 y tclcl-1.20 como ~ns / y ~tclcl /, respectivamente. En el nivel 3, los módulos en la jerarquía interpretados están bajo TCL directorio. Entre estos módulos, los de uso más frecuente en concreto (ej, ns-lib.tcl, ns-node.tcl, ns-link.tcl) se almacenan en lib directorio en el Nivel 4. Los módulos de simulación en la jerarquía compilada se clasifican en los directorios de nivel 2. Por ejemplo, las herramientas de directorio contienen diversas clases de ayuda, tales como generadores variables aleatorias. Directorio common contiene módulos básicos relacionados con el reenvío de paquetes, como el simulador, el planificador, el conector de paquetes. Directorios de cola, TCP, y trazas contienen módulos para la cola, TCP (Transmission Control Protocol), y el rastreo, respectivamente"

Es importante conocer la estructura del directorio de la herramienta NS2 para la correcta implementación de los protocolos. Los protocolos a implementar deberán estar situados en el Nivel 3 dentro del directorio ns-2.35, así estos protocolos podrán interactuar con el nivel 2 y el nivel 4 de NS2. Los cambios a realizarse para la correcta implementación y funcionamiento de los protocolos se deben realizar en el nivel 3 y 4.

2.6.2. Componentes de NS2

Los componentes que conforman la herramienta de simulación NS2 se describen en la tabla 2.

Tabla 2: Componente de NS2

Componentes	Requerido	Opcional	Descripción
Tcl	X		Tool Command language, es un lenguaje interpretado que contiene las ordenes a ejecutar
Tk	X		Es una extensión de TCL que forma el paquete de distribución Tcl/Tk
Otcl	X		Se trata de una extensión de Tcl/Tk con soporte de programación orientada a objetos
TclCL	X		Permite variables de C++ con OTCL
Ns	X		Componentes principal para la ejecución del simulador
Nam	X		Permite visualizar simulaciones animadas
Xgraph		X	Permite visualizar gráficas de parámetros de las simulaciones
Gt-itm & sgb2ns		X	Generador de topologías de la red
CWeb		X	Componente opcional pero requerido para el uso de Gt-itm
SGB		X	Stanford Graphics Base, formato de salida de las topologias de red generadas con Gt-itm
Zlib	X		Componente opcional, pero requerido para el uso de Nam

Fuente: (Martinez & Palau, 2011)

Elaboración: La Autora

Como se puede observar en la tabla 2, algunos componentes son necesarios en la herramienta NS2 para su total funcionamiento, para la realización de este trabajo de titulación será requerido el componente NAM ya que nos permitirá visualizar las simulaciones realizadas y por ende el componente Zlib que es necesario para el funcionamiento de NAM. Cabe mencionar que la versión de cada uno de los componentes antes mencionados, depende siempre de la versión de la herramienta NS2 que se esté utilizando.

2.6.3. Proceso de simulación de NS2

Debido a que uno de sus lenguajes es C++, se puede invocar a la herramienta mediante consola con solo poner ns, siempre y cuando este correctamente instalada. Otra manera de interactuar con NS2 es definir un script en Otcl es decir un lenguaje orientado a objetos que es una versión de TCL 2 orientado a objetos, donde se declara los protocolos de comunicaciones utilizados, la topología, el tipo de tráfico, entre otros (Capella, n.d.).

El TCL es el único INPUT que da el usuario, el resto del procesamiento lo realiza internamente NS2. Luego de llamar el TCL con todo lo que se desea simular, se procede a generar un conjunto de datos de salida que se guarda en un fichero de traza, estas trazas pueden servir para realizar un procesamiento de datos es decir para obtener datos específicos que se deseen evaluar, además permite realizar un análisis visual del envío, recepción de los paquetes de datos y control de toda la simulación del script mediante *Network Animator* también conocido con NAM. Todo el proceso antes mencionado se lo puede observar en la figura 7.

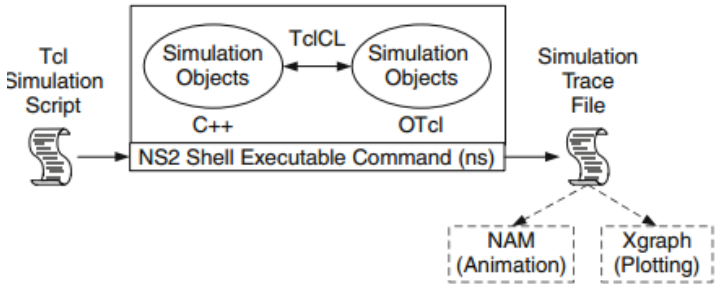


Figura 7: Arquitectura básica de NS2

Fuente: (Issariyakul & Hossain, 2012)

Elaboración: (Issariyakul & Hossain, 2012)

Como se ha mencionado anteriormente NAM es una herramienta que permite realizar la simulación de los *Script*, a continuación se menciona sus principales características.

2.6.3.1. NAM (Network Animator)

Es una aplicación que sirve para representar la simulación que hemos programado en NS2, podemos visualizar la topología de la red diseñada y el tránsito de los paquetes de un nodo hacia otro con las colas que se generan en cada nodo e incluso la pérdida de dichos paquetes (Sevilla, 2011). Además se puede pausar adelantar o retrasar la simulación.

Una de las aplicaciones que posee NAM es el llamado NAM editor, como su nombre lo indica es una herramienta para crear simulaciones con extensiones .nam. Esto ayudará en lugar de programar en Otcl un script, con NAM lo único que hará es dibujar la topología añadiendo nodos y enlaces gráficamente, para posteriormente indicar todos los parámetros de la simulación, como fuentes de tráfico, protocolo tcp, ftp, entre otros. Una vez dibujada la topología el editor nos da la opción de seleccionar el tipo de protocolo a nivel de aplicación o transporte que asignaremos a cada nodo (Sevilla, 2011)**.**

Sevilla (2011) manifiesta algunas opciones que el editor NAM posee:

- **Retroceso Rápido:** se retrocede multiplicando su paso del tiempo por 25
- **Retroceso Normal:** se retrocede según el paso del tiempo normal
- **Stop:** detiene la simulación
- **Avance normal:** se inicia la animación
- **Avance rápido:** se inicia la animación y avanza multiplicando el paso del tiempo por 25
- **Tiempo:** indica el instante en que se encuentra la simulación
- **Zoom:** aumenta y disminuye la simulación
- **Indicador de tiempo:** muestra el tiempo transcurrido en la simulación mediante una barra de tiempo.
- **Flujo de enlace:** en la opción *Graph* podrá visualizar el tiempo que viajara la información por el enlace en ambas direcciones y la información que podrá perderse.

En la figura 8 se puede observar la pantalla principal de la herramienta NAM, la cual muestra sus opciones principales.

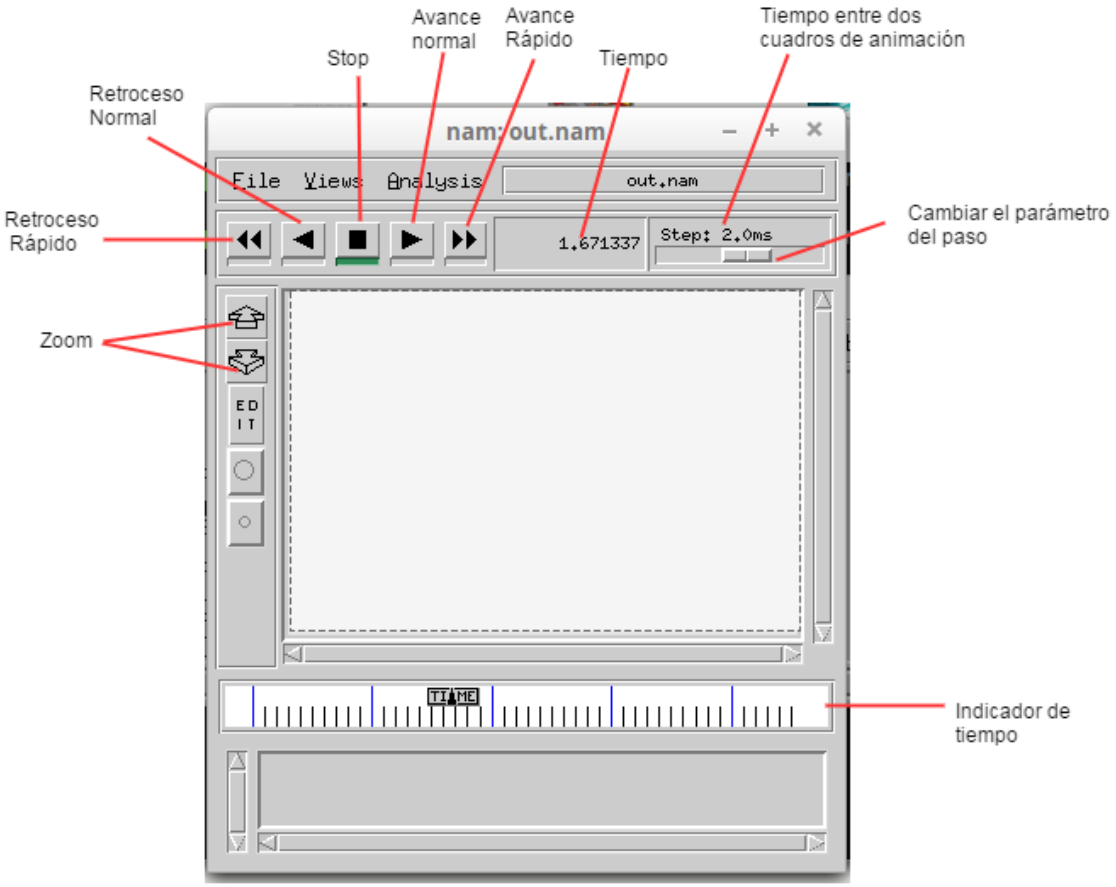


Figura 8: *Pantalla principal de NAM*

Fuente: La Autora

Elaboración: La Autora

2.7. Selección del protocolo a implementar

Conforme al trabajo de titulación se elige los protocolos CBRP y BCHP para ser implementados en NS2 debido a que son protocolos de enrutamiento jerárquico para redes móviles Ad Hoc, los mismos que serán analizados y comparados en diferentes versiones de la herramienta de simulación. Además debido a que existen protocolos ya implementados se ha seleccionado también el protocolo AODV para su respectivo análisis y comparación.

CAPITULO 3

ANÁLISIS DE PROTOCOLOS CBRP Y BCHP

3. Análisis de protocolos CBRP y BCHP

A continuación se describirán los protocolos de enrutamiento seleccionados para la implementación del trabajo de titulación. Los protocolos establecidos son CBRP y BCHP los mismos que se les realizará una revisión teórica y el análisis del funcionamiento de cada uno de estos protocolos.

3.1. Protocolo CBRP

En el apartado 2.4.3.1 se menciona principalmente el protocolo de enrutamiento jerárquico CBRP y a continuación se describe breves rasgos de este protocolo. A continuación se profundiza para conocer sus paquetes, tablas de enrutamiento y estrategias utilizadas para la implementación de este protocolo.

3.1.1. Características de CBRP

CBRP es un protocolo de enrutamiento que pertenece al grupo de los protocolos jerárquicos. Debido a que divide los nodos en clúster, cada grupo elige una cabeza de grupo, llamada también “clústerhead” que mantiene información de pertenencia de los nodos de grupo y del proceso de enrutamiento (Hollerung, 2004).

CBRP tiene las siguientes características:

- Operación completamente distribuida.
- Menos tráfico de inundación durante el proceso de descubrimiento de rutas dinámicas.
- Explotación explícita de enlaces unidireccionales que de otro modo no se utilizarían.
- Las rutas rotas podrían ser reparadas localmente sin redescubrimiento.
- Las rutas sub-óptimas podrían acortarse a medida que se utilicen.

Según Torres (2011) el enrutamiento se realiza en dos niveles:

- **Intra-clúster:** La entrega es directa y el nodo utiliza su tabla de vecinos, ya que los nodos conocen sus vecinos a una distancia de dos saltos. La comunicación entre los nodos dura más tiempo y apoya al mantenimiento de los clúster en la red.
- **Extra-clúster:** se realiza entre los Jefe de clúster para intercambiar información entre clúster. Cuando un nodo se encuentra fuera del rango de 2 saltos, la información se le entrega al Jefe Clúster para realizar la consulta a la tabla adyacente.

3.1.2. Paquetes de CBRP

Por otra parte CBRP utiliza los siguientes tipos de mensajes para cumplir con su funcionamiento:

- **HELLO:** mensaje de saludo, es usado para mantener y establecer comunicación con nodos vecinos, mediante HELLO_INTERVAL. Se clasifican en dos tipos de mensajes, HELLO vecino y HELLO clúster. En algunos casos el mensaje HELLO es activado cuanto algún evento requiere una acción rápida.

  Los mensajes HELLO vecino "envían tablas de rutas a todos los nodos a través de broadcast" (Medina & Macas, 2013), también son utilizados para el mantenimiento del clúster y determinar el estado de los nodos vecinos. En cambio los mensajes HELLO clúster "son utilizados para generar una red de malla entre los diferentes clúster" (Torres, 2011) y además se utiliza para el descubrimiento de clúster adyacentes.
- **Solicitud de Ruta (RREQ):** este paquete contiene la dirección del destino, del nodo fuente y un número único de identificación del nodo origen (Carrión & Delgado, 2015) y es utilizado para descubrir una ruta a un destino que se encuentra fuera de la distancia de dos saltos.

  Cada vez que un nodo recibe el mensaje RREQ, si conoce la ruta se añade su identificador a la ruta contenida en el paquete y se trasmite a sus vecinos el nuevo mensaje RREQ. En caso que no se conozca la ruta este proceso se repite hasta que el nodo destino recibe uno de los paquetes.
- **Respuesta de Ruta (RREP):** contiene una copia de las direcciones acumuladas en el paquete RREQ recibido y es utilizado en el descubrimiento de las rutas, es decir para dar respuesta a un mensaje de solicitud de ruta

- **Enrutamiento:** ayuda en el proceso de enrutamiento, es decir informa la mejor ruta al origen, además ayuda al acortamiento de las rutas, estas rutas pueden acortarse debido al movimiento de los nodos. También informa a los nodos que la ruta que está en uso, ha cambiado (Torres, 2011).

- **Errores de Ruta (RERR): "**paquete para informar errores de enrutamiento" (Medina & Macas, 2013), estos errores se producen cuando la capa de enlace encuentra problemas graves de transmisión. Este paquete contiene las dos direcciones de los nodos que estaban unidos por el enlace que falló. Cuando el nodo de origen recibe un mensaje de error debe eliminar la ruta de su caché (Chalmeta, 2009).La tabla de adyacencias sólo funciona cuando existe un mensaje de error (Enciso, 2013).

3.1.3. Tablas de enrutamiento de CBRP

Según (Torres, 2011) el protocolo CBRP utiliza las siguientes tablas para el enrutamiento respectivo:

- **Tablas de Vecinos:** es una estructura conceptual de datos que sirve para detectar el estado del enlace y la formación de agrupaciones. Se mantiene actualizada mediante la difusión del mensaje HELLO. Esta tabla contiene los campos para la identificación del vecino; el estado del vecino, es decir si es Jefe Clúster o Miembro del clúster; y el estado del enlace, es decir si es uni-direccional o bi-direccional (Torres, 2011).

  En el protocolo CBRP cada nodo conoce los enlaces uni/bidireccionales hacia sus vecinos. Cada uno de los nodos mantiene una tabla de vecinos, tal como se muestra en la tabla 3.

  Tabla 3: Tabla de vecinos en el protocolo CBRP

Vecino ID	**Estado del Enlace**	**Función**
Vecino 1	Enlace uni/bidireccional?	Es el vecino 1 el clústerhead
Vecino 2	Enlace uni/bidireccional?	Es el vecino 2 el clústerhead
...	...	...
Vecino N	Enlace uni/bidireccional?	Es el vecino N el clústerhead

  Fuente: (Fernandez & Mena, 2004)

  Elaboración: La Autora

- **Tabla de Adyacentes:** mantiene la información acerca de los clústeres adyacentes. Utilizada por los JC. Contiene información de identificación del Jefe de Clúster, información del nodo puente que permite llegar al clúster adyacente y el estado de enlace desde el nodo puente al jefe clúster adyacente.

- **Base de Datos de la topología de dos saltos:** Cada nodo existente difunde su información de la tabla de enrutamiento mediante el uso del paquete HELLO. Por lo tanto, un nodo podrá recopilar la información completa para proyectar o generar la topología de red con un diámetro de dos saltos de sí mismo.

3.1.4. Operaciones de CBRP

La operación del protocolo CBRP es distribuida y se basa en los siguientes componentes (Maldonado, 2012).

- **Formación de clúster:** Principalmente CBRP define estados para cada nodo:
  - ✓ C_UNDECIDED: nodo indeciso, es decir que está todavía en búsqueda de su host clúster.
  - ✓ C_MEMBER: miembro del clúster.
  - ✓ CLÚSTER_HEAD: el clúster ha sido seleccionado como Jefe de Clúster.

  El principal objetivo de la formación de clúster es imponer algún tipo de estructura en las redes ad hoc. El nodo que posee el ID más bajo y además un enlace bi-direccional en la tabla de vecinos, será un "clústerhead" es decir un Jefe clúster.
  Cada uno de los nodos mantiene una tabla donde almacena la información de los nodos vecinos. La tabla de vecino se mantiene periódicamente actualizada mediante la difusión de mensajes HELLO que contiene información sobre el estado del nodo, la tabla de vecinos y la tabla de adyacencia (Medina & Macas, 2013)**.**

- **Descubrimiento de adyacencia al clúster.-** el clústerhead descubre todos los enlaces bi-direccionales vinculados en sus agrupaciones adyacentes, utilizando información de los clústerhead vecinos que cada nodo mantiene en su tabla de adyacencia (CAT) que registra la información de todas sus cabezas de clúster.

- **Enrutamiento.**- primero descubre la ruta desde el nodo inicial al nodo final y después enruta los paquetes. Es muy importante mencionar que CBRP puede reparar rutas defectuosas a diferencia de otros protocolos. CBRP utiliza algunos mecanismo para informar o mejorar:
  - ✓ Acortamiento de ruta: una ruta de origen puede ser menos óptima en el tiempo ya sea por el movimiento de nodo o por alguna otra razón y debido a esto se debería acortar la ruta siempre que sea posible mediante el uso de mensajes de enrutamiento RREP o RREQ.
  - ✓ Error de ruta: cuando un nodo averigua que el salto siguiente ya no es accesible, crea un paquete de error de ruta, llamado también ERR y lo envía de nuevo al origen del paquete para notificar su fallo.
  - ✓ Reparación local: cuando el nodo detecta mediante el mensaje ERR una ruta rota, intenta rescatar el paquete de datos de la mejor manera posible "gracias a la

información almacenada localmente en la base de datos de la topología de 2 saltos" (Torres, 2011), repara y reenvía el paquete.

3.2. Protocolo BCHP

En el apartado 2.4.3.2 se menciona la descripción del protocolo BCHP, mismo que se propone como una optimización al protocolo CBRP que ha sido desarrollado para redes móviles Ad Hoc, debido a que las redes móviles tienen un cambio dinámico en la topología por lo que los protocolos de enrutamiento deben descubrir la ruta adecuada. A continuación se describe las características, paquetes, tablas de enrutamiento y estrategias utilizadas para la implementación de este protocolo.

3.2.1. Características de BCHP

Este protocolo mejora la disponibilidad de la red incluyendo nodo JCR, es decir nodos de Jefe de Clúster de Respaldo pero lo realiza de una forma reactiva y además tiene la ventaja de minimizar el tamaño de la tabla de enrutamiento debido a que los nodos envían información a su jefe de clúster (Torres, 2011).

Para reducir la información que se transfiere a todos los nodos, BCHP disminuye la sobrecarga de red utilizando los Jefe de clúster para el envió de los mensajes de enrutamiento a la red (Torres, 2011). Además incluye un nodo llamado Jefe de Clúster de respaldo JCR para cuando un JC no pueda cumplir sus funciones.

Para mejorar la gestión de la red, cada uno de los nodos en BCHP debe recopilar su información para ser enviada a su nodo JC. Si el nodo es JC, debe recopilar la información de gestión de todos los nodos del clúster, acumularla, comprimirla, asegurarla, para finalmente enviarla a través del *backbone* al Servidor de Administración (Torres, 2011).

3.2.2. Operaciones

- **Creación del clúster:**

Al inicio los nodos no pertenecen a ningún clúster, por lo que su estado es UNDECIDED, en el cual se calcula la métrica de acuerdo a las características de velocidad, estado de batería y ubicación; esta métrica es enviada mediante mensajes de difusión. Con estos valores cada uno de los nodos crea su tabla de enrutamiento y determinan cuales vecinos poseen enlaces bidireccionales hacia él, además se puede determinar los nodos con mejores características, entre ellos, el nodo JC (Jefe de Clúster) y JCR (Jefe de Clúster de Respaldo) (Torres, 2011)**.** En la figura 9 se muestra como empieza el proceso de la creación de un clúster.

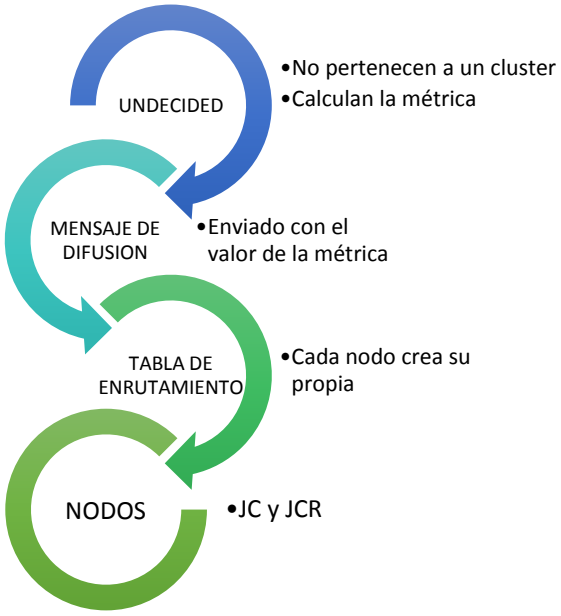


Figura 9: Creación del clúster

Fuente y Elaboración: La autora

- **Mantenimiento del clúster**

En la figura 10 se muestran los conceptos básico para el mantenimiento de un clúster.

Primeramente se debe mantener una jerarquía ya que debido al movimiento de los nodos, el consumo de energía o al fallo de los nodos JC, pueden cambiar en densidad y ubicación (Torres, 2011).

Al momento en que se unen dos clúster es posible que un nodo JC se encuentre dentro del área de cobertura de otro nodo JC (Criollo & Ruilova, 2013)**,** en ese momento se inicia un periodo denominado *contención.* En el instante en que la contención expira, se crea un nuevo clúster por ende se debe elegir un nuevo nodo JC y un nodo JCR.

El nodo JCR determina que un nodo JC no se encuentra disponible por medio de mensajes periódicos y de actualizaciones entre ellos (Torres, 2011)**.** Cuando esto sucede el nodo JCR toma el valor JC y asume la jerarquía, es decir se convierte en nodo JC, por esta razón envía mensaje por difusión a todos los nodos del clúster informando su nuevo estado y se debe elegir un nuevo nodo JCR de entre todos los nodos.

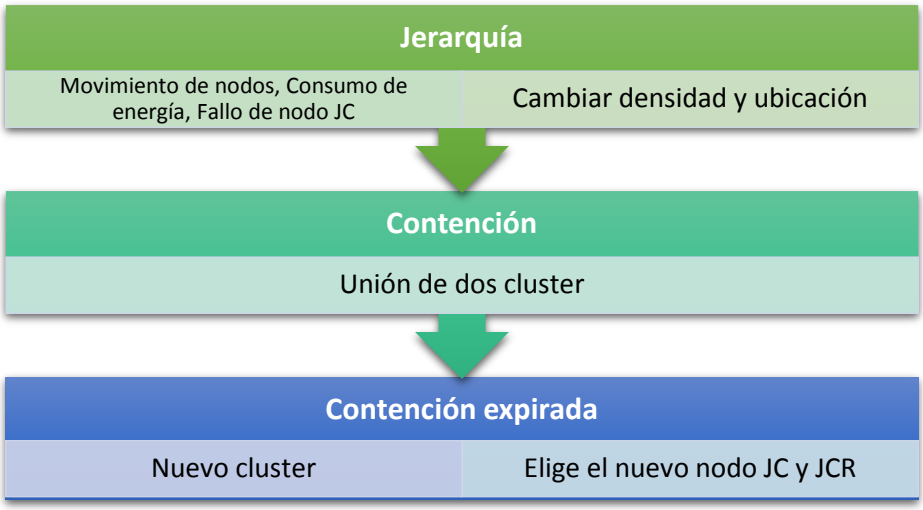


Figura 10: Mantenimiento de clúster

Fuente y Elaboración: La Autora

- **Enrutamiento**

Según Torres (2011), el protocolo BCHP define tres niveles para apoyar las funciones de enrutamiento:

1. *Dentro del área de transmisión:* en este nivel el nodo origen envía directamente la información al nodo destino, además los nodos son alcanzados directamente ya que poseen un enlace bidireccional entre ellos.

2. *Dentro del clúster:* en este nivel los nodos se comunica entre sí mediante su tabla de vecino. Cada uno de los nodos mantiene una tabla de enrutamiento de vecinos, donde se obtiene la dirección del nodo que servirá como enrutador (Torres, 2011).

3. *Fuera del clúster:* en este nivel cada Jefe de Clúster agrupa la información y la envía hacia el clúster de destino. Cuando un nodo determina que no existe ruta hacia un nodo destino genera un RREQ que es entregado al JC (Criollo & Ruilova, 2013). El Jefe de Clúster contiene una tabla de adyacencias para la comunicación entre clúster. La ruta es informada al nodo origen a través de mensajes de RREP enviados por el nodo destino o a su vez por un jefe de clúster que posee la información actualizada (Torres, 2011).

En la figura 11, se muestra el proceso para el enrutamiento entre clúster del protocolo BCHP.

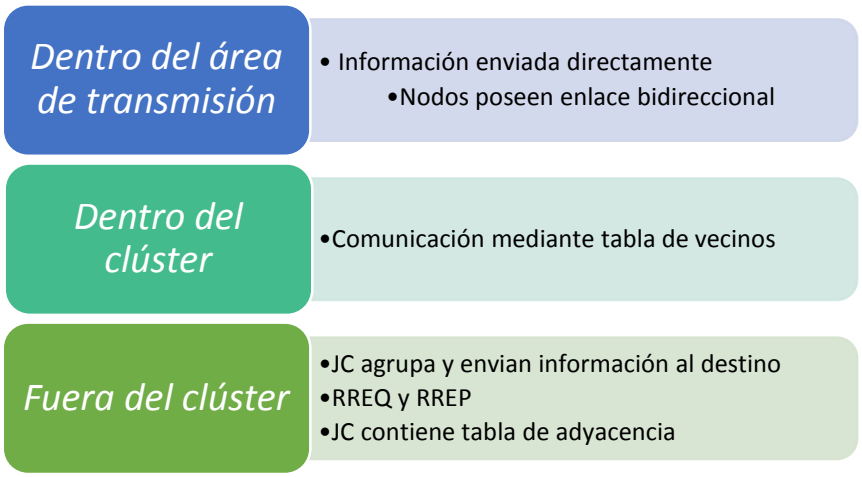


Figura 11: Enrutamiento

Fuente y Elaboración: La Autora

3.2.3. Paquetes

BCHP mejora los paquetes que intervienen para el descubrimientos de rutas en el protocolo CBRP, estos paquetes serán usados en la elección de los nodos JC, JCR, SA, SAR.

- **Mensajes HELLO:** Estos mensajes son utilizados para mantener actualizada la topología del clúster y además para definir y mantener la unión entre clúster. Consumen ancho de banda y no permiten a un nodo entrar en modo espera. Se clasifica en dos tipos de mensajes:

- **HELLO Vecino:** Los mensajes HELLO Vecino sirven para actualizar las tablas de rutas y además determina la elección de los estados entre los nodos Jefe Clúster y Jefe Clúster de Respaldo.

En la figura 12, se muestra la estructura del paquete HELLO Vecino sin la cabecera BCHP.

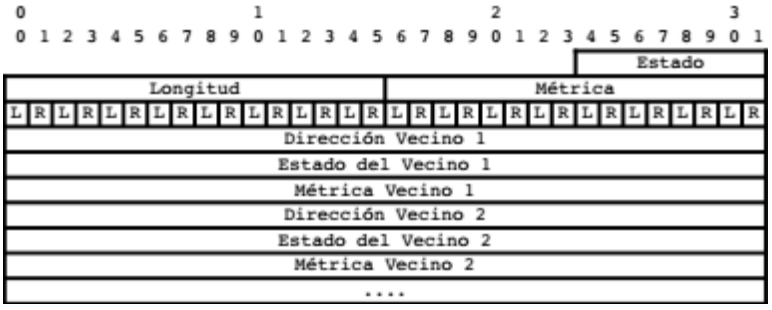


Figura 12: Estructura del paquete HELLO Vecino

Fuente: (Torres, 2011)

Elaboración: (Torres, 2011)

En el momento que un nodo recibe un mensaje HELLO, actualiza la entrada en la tabla de enrutamiento y en caso de no encontrar crea una nueva entrada. A través de este paquete se intercambian métricas de los nodos vecinos.

Para el funcionamiento de este paquete Torres (2011) ha agregado los campos de *métrica y métrica del vecino.* Los demás campo son los mismos del protocolo CBRP, a continuación se los lista:

- ✓ **Estado:** existen diferentes valores 0: UNDECIDED, 1: Jefe Clúster, 2: Nodo Manejado, 3: Jefe Clúster de Respaldo, 4: Servidor de Administración y 5: Servidor de Administración de Respaldo. Cada nodo informa de acuerdo a cada uno de los estados antes mencionados.
- ✓ **Longitud:** muestra la cantidad de vecinos listados.
- ✓ **Métrica:** se almacena el valor obtenido (Chatterjee, Das, & Turgut, 2002) del cálculo relacionado con las capacidades del nodo: procesamiento, movilidad y estado de la batería.
- ✓ **Bandera L:** informa del estado del enlace con el vecino, ya sea bidireccional o unidireccional.
- ✓ **Bandera R:** identifica si el vecino tiene el estado de Jefe de Clúster.
- ✓ **Dirección del Vecino:** Identificador único del vecino
- ✓ **Rol del Vecino:** informa el estado del vecino remitente.
- ✓ **Métrica del Vecino:** peso calculado que sirve para la elección del nodo Jefe de Clúster y Servidor de Administración

- **HELLO Clúster:** El mensaje HELLO Clúster es utilizado para el descubrimiento de clúster adyacentes, además también es utilizado para la elección de los nodos SA y SAR. En la figura 13 se muestra la estructura de este paquete sin la cabecera general de BCHP.

  Los nodos Jefe de Clúster intercambian su tabla de adyacencia para determinar la topología de la red, además utilizan la información para determinar cuáles nodos son adecuados para la responsabilidad de la gestión de red.

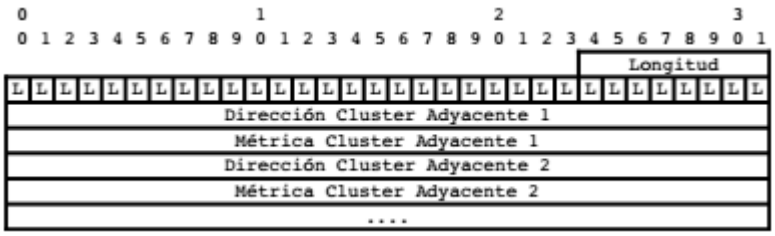


Figura 13: Estructura del paquete HELLO Clúster

Fuente: (Torres, 2011)

Elaboración: (Torres, 2011)

Para el funcionamiento del modelo Torres (2011) a agregado el campo de Métrica del clúster adyacente. Los demás campos son los mismos del protocolo CBRP, a continuación se los lista:

- ✓ **Longitud:** se almacena todos los nodos Jefe de Clúster adyacentes
- ✓ **Bandera L:** informa el tipo de enlace del Jefe Clúster adyacente, ya sea unidireccional o bidireccional
- ✓ **Dirección del JC adyacentes:** se almacena la dirección del jefe clúster adyacente
- ✓ **Dirección del vecino:** identificador del vecino
- ✓ **Métrica del JC adyacente: "**es el peso calculado que sirve para la elección del nodo SA y SAR" (Torres, 2011)

- **Solicitud de Ruta (RREQ):** Los mensajes de solicitud de ruta o conocidos como RREQ son utilizados para conocer cómo llegar a un destino no conocido. En la figura 14 se muestra el paquete de Solicitud de ruta.

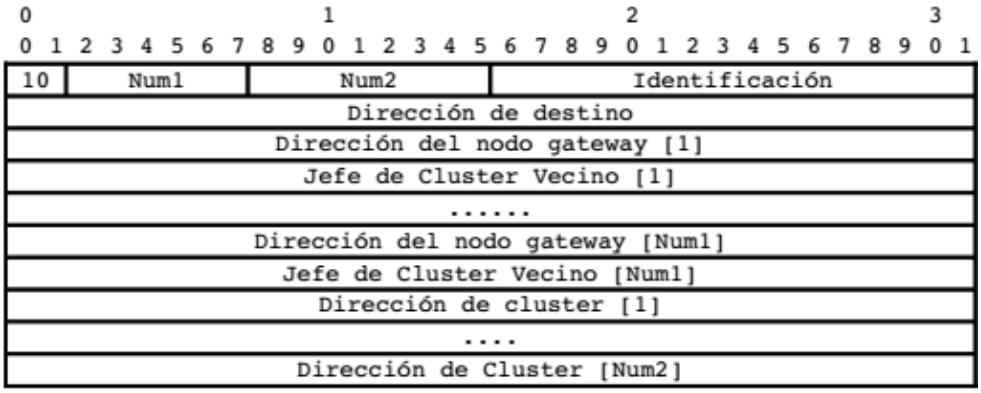


Figura 14: Estructura del paquete de solicitud de ruta

Fuente: (Jiang, Li, & Tay, 1999)

Elaboración: (Jiang, Li, & Tay, 1999)

Cuando un nodo no conoce el destino, envía un paquete RREQ que es recibido por un JC que se encarga de enviar a los Clúster adyacentes que se encuentran en su diámetro. Todo el proceso se realiza hasta que el mensaje llega a su destino.

El protocolo BCHP utiliza los mismos campos para el descubrimiento de ruta que el protocolo CBRP, la ventaja de BCHP "es que trata de mantener siempre un JC activo lo que permite que los procesos de enrutamiento mantengan su rendimiento con normalidad" (Torres, 2011). A continuación se detallan los campos del paquete RREQ:

- ✓ **10:** tipo de paquete de solicitud de ruta, este protocolo siempre colocara este valor binario
- ✓ **Num1:** se almacena la cantidad de pares de Clúster Adyacentes y nodos Gateway, los cuales reciben la solicitud de ruta.
- ✓ **Num2:** se almacena la cantidad de dirección de clúster que la solicitud ha atravesado, se incrementa cada que el paquete pasa por un clúster.
- ✓ **Identificación:** identificación única de la solicitud de ruta que genera el mismo campo.
- ✓ **Dirección de destino:** identificación de destino a la cual se enviara la información.
- ✓ **Dirección del nodo Gateway [n]:** se almacena la dirección del nodo puente por el cual se puede llegar a un Jefe Clúster (Torres, 2011).
- ✓ **Dirección del Jefe Clúster Vecino [n]:** conjuntamente con la dirección del nodo forman un solo conjunto.
- ✓ **Dirección del clúster [n]:** se almacena la dirección del JC cada vez que la solicitud de ruta pasa por un clúster, de esta manera el nodo destino tiene la opción de utilizar el camino de reversa creado por este campo.

- **Respuesta de solicitud de Ruta (RREP):** Este paquete ayuda para informar al nodo origen de una ruta disponible para el envió de información mediante sus mensajes RREP. La figura 15 muestra la estructura del paquete de respuesta de solicitud de ruta.

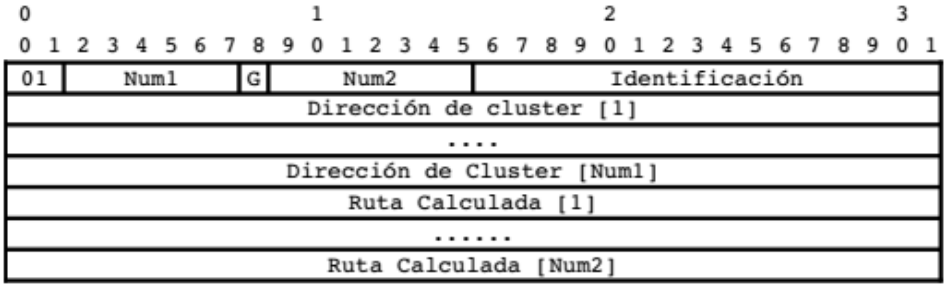


Figura 15: Estructura del paquete de respuesta de solicitud de ruta

Fuente: (Jiang et al., 1999)

Elaboración: (Jiang, Li, & Tay, 1999)

El jefe de clúster o el nodo destino por medios de mensajes RREP informa la ruta disponible. A continuación se detalla cada uno de los ampos:

- ✓ **01:** cuando el paquete es RREP, el protocolo lo identifica con el campo con el valor 01
- ✓ **Num1:** se almacena la cantidad de clúster que el paquete RREQ atraviesa en el momento, incrementa cada que pasa por un clúster.
- ✓ **Bandera G:** este campo indica si el paquete es un RREP.
- ✓ **Num2: "**almacena el números de direcciones en la ruta calculada" (Jiang et al., 1999)
- ✓ **Identificación:** identificación propia del paquete, pero la misma identificación que la solicitud de ruta.
- ✓ **Dirección de clúster:** se almacena la secuencia de JC que atravesado el paquete RREQ y es la misma dirección del paquete RREQ.
- ✓ **Ruta calculada:** Una secuencia de direcciones de la ruta de salto por salto calculada por el jefe de clúster.

- **Enrutamiento:** El paquete de enrutamiento es utilizado para dar a conocer al nodo origen que se encuentra una mejor ruta para enviar información hacia el nodo destino. En la figura 16, se muestra la estructura de paquete de enrutamiento.

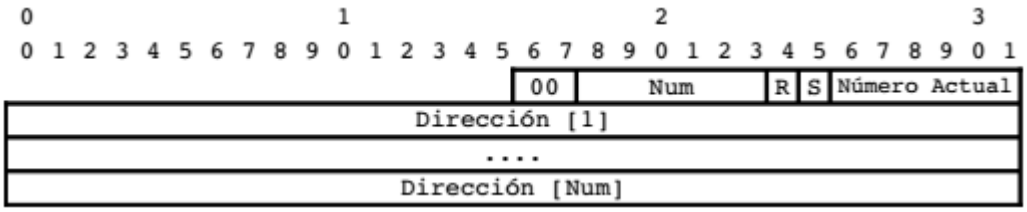


Figura 16: Estructura del paquete de enrutamiento

Fuente: (Jiang et al., 1999)

Elaboración: (Jiang, Li, & Tay, 1999)

- ✓ **00:** determina el tipo de paquete
- ✓ **Num:** almacenan la cantidad de direcciones de la ruta fuente
- ✓ **Bandera R:** indica si la ruta se ha recuperado localmente (Jiang et al., 1999)
- ✓ **Bandera S:** indica si la ruta ha sido acortada (Jiang et al., 1999)
- ✓ **Número actual:** indica la dirección del ultimo nodo visitado (Torres, 2011)
- ✓ **Dirección:** almacena la secuencia de direcciones que establecen la ruta.

- **Error:** El paquete error o también conocido como paquete ERR. Este paquete se crea al momento de encontrar un nodo inaccesible, en este momento se enviara el paquete al nodo origen para informar del fallo de la red. “El protocolo BCHP utiliza el mismo paquete que el definido en CBRP” (Torres, 2011) . En la figura 17 se muestra el formato del paquete error:

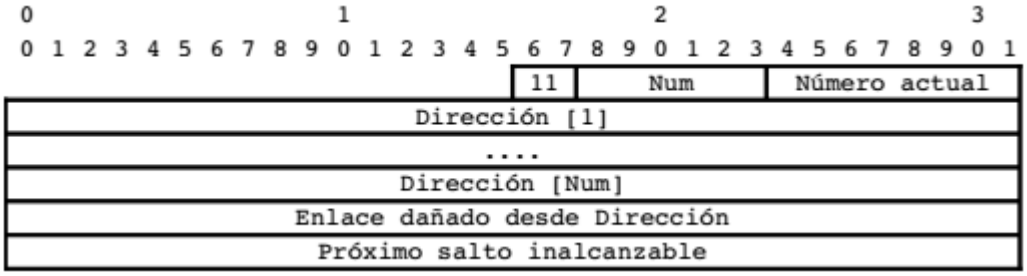


Figura 17: Estructura del paquete de error

Fuente: (Jiang et al., 1999)

Elaboración: (Jiang, Li, & Tay, 1999)

- ✓ **11:** tipo de paquete de error de ruta con el valor binario 11
- ✓ **Num:** cantidad de direcciones que forman la ruta de origen (Jiang et al., 1999)
- ✓ **Número actual:** se almacena la dirección del ultimo nodo visitado (Torres, 2011)

- ✓ **Dirección:** La ruta de origen que el paquete debe de recorrer para alcanzar su destino
- ✓ **Enlace dañado desde dirección:** se almacena la dirección del nodo que origina el paquete error.
- ✓ **Próximo salto inalcanzable:** dirección del nodo que se encuentra en la ruta original pero que el nodo informante no pudo alcanzar visitado (Torres, 2011)

3.2.4. Tablas

Además de los orígenes de las tablas de enrutamiento, depende del tipo de nodo ya que éste puede o no contar con dos tablas de enrutamiento. Si es un nodo manipulado solo contará con una tabla de ruta de vecinos, en cambio sí es un nodo JC (Jefe de Clúster), JCR (Jefe de Clúster de respaldo), SA (Servidor de Administración) y SAR (Servidor de Administración de Respaldo) contará con una tabla extra que será la tabla de adyacencia (Torres, 2011)

Tabla 4: Entradas para la tabla de rutas de vecinos

Id Vecino	Estado Vecino	Estampa de Tiempo	Estado de Enlace	Contención	Métrica

Fuente: (Torres, 2011)

Elaboración: La Autora

Debido a que cualquier nodo poseerá una tabla ruta de vecinos, en la tabla 4 se muestra las entradas para la tabla de rutas de vecinos. Esta tabla permite al nodo tener el conocimiento exacto de la topología del clúster (Medina & Macas, 2013). A continuación se detallan todas las entradas de la tabla de ruta de vecinos:

- **Identificador del vecino:** es donde se recopila la identificación de cada uno de los nodos vecinos y además puede almacenar direcciones IPV4 e IPV6 (Medina & Macas, 2013)**.** Se define con un tamaño de 128 bits.
- **Estado vecino:** es donde se almacena que tipo de nodo es el nodo vecino y puede definirse con un tamaño de 3 bits (Torres, 2011)**.**
- **Estampa de tiempo:** se almacena el tiempo de la última actualización de la ruta y puede definir con un tamaño de 64 bits
- **Estado del enlace:** es donde se establece si el nodo se encuentra dentro del rango de transmisión y puede definir con un tamaño de 1 bit (Medina & Macas, 2013)

- **Contención:** nos permite determinar cuál es el nodo que está en impedimento con el Jefe Clúster (JC), esto se debe cuando dos nodos con estado JC se encuentran dentro del dominio de un clúster. Se define con un tamaño de 2 bits.

- **Métrica:** se almacenan las métricas calculadas y enviadas por los nodos vecinos y sirve para el mantenimiento del clúster (Torres, 2011)**.**

Como se mencionó anteriormente si un nodo es JC, JCR, SA o SAR, además de mantener una tabla de rutas de vecinos adicionalmente debe mantener una tabla de adyacencias, "la cual mantiene registros de los clúster vecino" (Torres, 2011)**.**

Tabla 5: Entradas para la tabla de rutas de adyacencia

Id Vecino	Estado Vecino	Estampa de Tiempo	Métrica	NP1	Estampa de tiempo NP1
				NP2	Estampa de tiempo NP2
				NPn	Estampa de tiempo NPn

Fuente: (Torres, 2011)

Elaboración: La Autora

La tabla 5 muestra las entradas para la tabla de rutas de adyacencia para nodos JC, JCR, SA, SAR. A continuación se describe cada uno de los campos de la respectiva tabla:

- **Identificador del vecino:** al igual que el identificador de la tabla de ruta de vecinos almacena la identificación de todos los nodos vecinos, se define por su mismo tamaño y almacena direcciones IPV4 e IPV6

- **Estado vecino:** se almacena que tipo de nodo el Jefe Clúster (JC) y además identifica cuál de los nodos es Servidor de Administración (SA) y Servidor de Administración de Respaldo (SAR). De la misma manera que la tabla de rutas de vecinos posee un tamaño de 3 bits.

- **Estampa de tiempo:** al igual que la tabla de ruta de vecinos, almacena el tiempo de la última actualización de la ruta, con un tamaño de 64 bits.

- **Métrica:** de la misma forma que la tabla de ruta de vecinos, almacena las métricas calculadas y enviadas por los nodos vecinos y además “sirve para el proceso de la elección de SA y SAR” (Torres, 2011)
- **NP1… NPn:** NP son los nodos puente y pueden tener más de un nodo JC, es decir pertenecer a más de un clúster. En esta dupla se almacenan las identificaciones de los nodos puente que sirven para comunicarse con los JC vecinos (Medina & Macas, 2013)**.**
- **Estampa de tiempo NP1…NPn:** almacena la última actualización hacia el nodo fuente respectivamente (Torres, 2011)**.**

La diferencia principal de estos protocolos antes mencionados es que gracias a la inclusión de los nodos JCR en el protocolo BCHP se ha mejorado la disponibilidad de la red, además este protocolo tiene la ventaja de minimizar el tamaño de tabla de enrutamiento gracias a los nodos que envían información al nodo JC.

CAPITULO 4

IMPLEMENTACIÓN DE PROTOCOLOS DE ENRUTAMIENTO CBRP y BCHP

4. Implementación de protocolos de enrutamiento CBRP y BCHP

En este capítulo se describe cada una de los módulos del protocolo BCHP. Se especificará principalmente los archivos modificados dentro de la herramienta de simulación, después la estructura del protocolo de enrutamiento. Por otra parte, también se definen las clases del protocolo y los algoritmos propuesto del protocolo BCHP. Debido a que la implementación es muy similar con el protocolo CBRP y lo que difiere es la estructura de cada protocolo, sólo se tomará en cuenta el protocolo BCHP en este capítulo.

4.1. Archivos modificados en NS2

Para la implementación del protocolo BCHP en la herramienta de simulación se debe incluir el algoritmo seleccionado dentro del directorio de NS2. A continuación en la tabla 6 se describe cada uno de los archivos en los que se debe realizar los cambios:

Tabla 6: Inclusión de protocolo BCHP en la herramienta de simulación.

Archivo	**Descripción**
ns-2.35/common/packet.h	Se define el método de acceso a la cabecera del paquete. Se incluye el paquete BCHP en la lista de paquetes reconocidos por NS2. Define a BCHP como protocolo de enrutamiento. Define la palabra BCHP para identificar el protocolo en las trazas.
ns-2.35/trace/cmu-trace.h	Declaración de la función de registro de traza.
ns-2.35/trace/cmu-trace.c	Inclusión de las librerías del encabezado del paquete. Implementación de la función de registro de trazas y obtención del desplazamiento del encabezado BCHP con respecto al paquete.
ns-2.35/tcl/lib/ns-packet.tcl	Agregar el encabezado del paquete BCHP al paquete general.
ns-2.35/tcl/lib/ns-lib.tcl	Inicializar el Agente de enrutamiento en el entorno OTcl. Permitir la coexistencia de BCHP y otros protocolos de enrutamiento en el caso de que exista una estación base.

	Definir el nodo como de tipo BCHP. Enlazar los agentes de los nodos con el protocolo de enrutamiento, en este caso BCHP
ns-2.35/tcl/lib/ns-mobilenode.tcl	Implementación de las funciones en OTcl para instanciar y relacionar el agente y el protocolo de enrutamiento en el simulador.
ns-2.35/Makefile	Incluir el código fuente para que se compile con el simulador.

Fuente: (Torres, 2011)

Elaboración: La Autora

4.2. Estructura del protocolo BCHP

Para la implementación del protocolo de enrutamiento jerárquico proactivo se debe crear una estructura de archivos dentro de la herramienta de simulación NS2. Esta estructura posee archivos de cabecera y código fuente (Torres, 2011) para el funcionamiento del protocolo BCHP. Dentro del directorio ns2.35 se colocará una nueva carpeta para el nuevo protocolo, dentro se ubicarán los siguientes archivos. A continuación se describe cada uno de ellos. Véase tabla 7.

Tabla 7: Archivos de protocolo BCHP

Archivo	**Descripción**
bchpagent.h	Se definen cada una de las propiedades y métodos del agente BCHP.
bchpagent.cc	Se encuentran los constructores, métodos para el envío y recepción de los paquetes del protocolo BCHP, métodos para el manejo de fallas y errores de comunicación.
bchpntable.h	Se definen las clases, propiedades y métodos para el manejo de las tablas de enrutamiento del protocolo BCHP.
bchpntable.cc	Se encuentra el código para implementar los métodos y el acceso a las propiedades de las clases que definen el protocolo BCHP (Torres, 2011). Además también se define la implementación de temporizadores, los procesos para la creación y mantenimiento de un clúster.

bchp_packet.h	Se define la estructura de paquete BCHP que será utilizado para enviar y recibir la información del protocolo.
hdr_bchp.h	Se define la estructura de la cabecera del paquete BCHP, teniendo en cuenta sus métodos y propiedades.
hdr bchp.cc	Se define la implementación de las clases para la unión de las clases en C++ y OTcl.

Fuente: (Torres, 2011)

Elaboración: La Autora

4.3. Definición de las clases del protocolo BCHP

Un nodo tipo BCHP según (Criollo & Ruilova, 2013) contiene los siguientes componentes:

- **Agente:** selecciona la información del nodo principal.
- **Tablas de enrutamiento:** se guarda la información de cada uno de los nodos y los clúster sirven como entrada para los procesos del clúster (Torres, 2011)

A continuación se detallarán cada una de las clases para los componentes de los nodos de BCHP.

4.3.1. Agente

En la figura 18 se podrá observar las clases principales con cada uno de los atributos y método del nodo BCHP.

Un nodo BCHP que ya ha sido implementado en la herramienta de simulación hereda y puede modificar el comportamiento general y básico definido en las clases **MobileNode** y **Agent**. Esto sirve para el proceso de envío y recepción de los paquetes para los nodos normales e inalámbricos (Criollo & Ruilova, 2013).

La clase **bchpagent** es la principal y se definen las propiedades que heredan comportamientos de sus clases superiores, y los métodos. Esto permite interactuar con los siguientes componentes del simulador.

El nodo BCHP hereda eventos y manejadores para la interacción con el simulador, las clases que ayudan a este proceso son: **NsObject, Event, Handler.**

El nodo BCHP también hereda de la clase **trace** para obtener métodos de manejo de trazas.

El nodo BCHP maneja una tabla de enrutamiento llamada **Agent::HNeighborTable.**

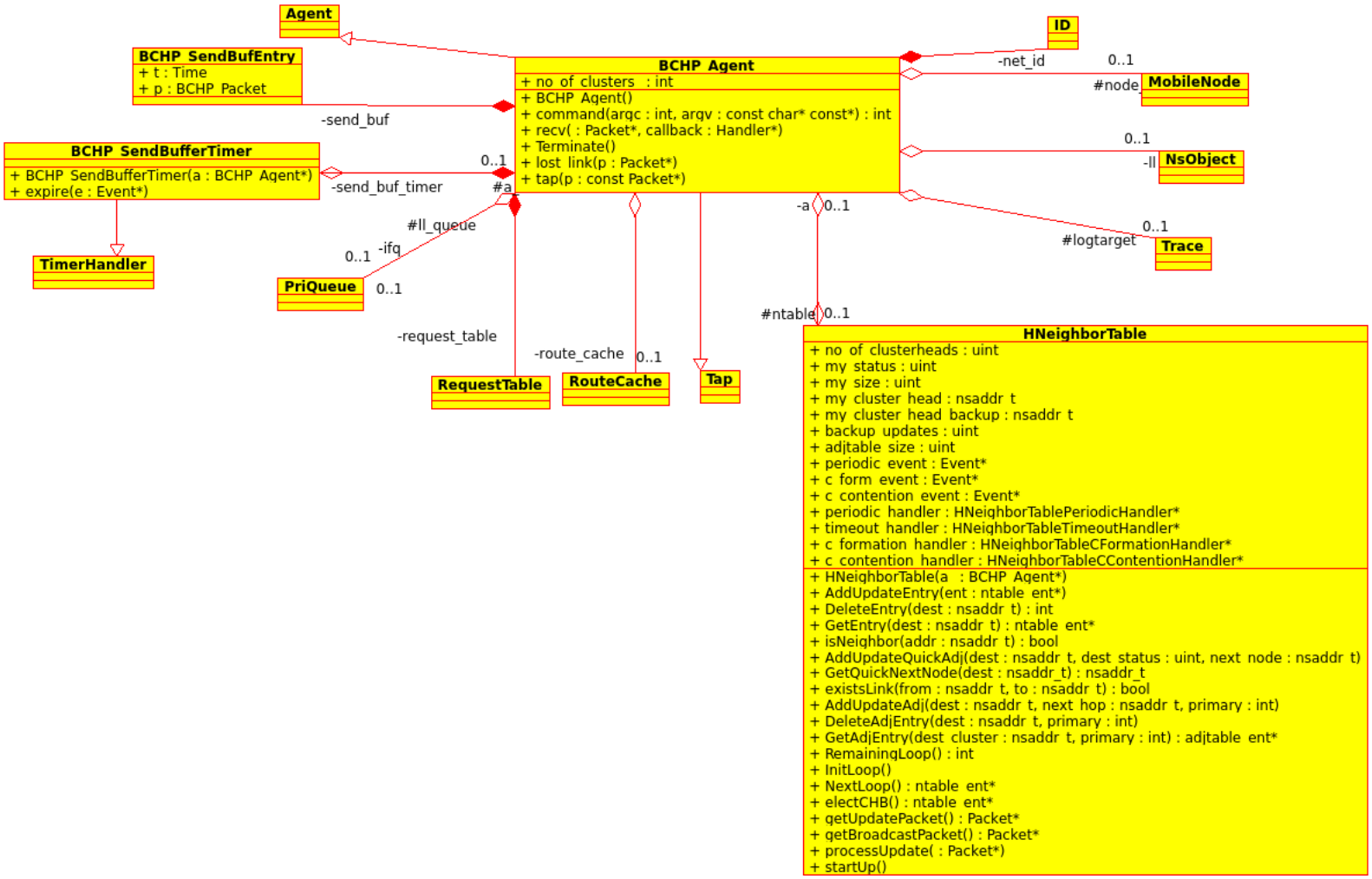


Figura 18: Diagrama de Clases del nodo BCHP

Fuente y Elaboración: La Autora

4.3.2. Tabla de enrutamiento de BCHP

La figura 19 se muestra los atributos y métodos que interactúan con la tabla de enrutamiento de un nodo BCHP.

El protocolo de enrutamiento puede contar con dos tablas de enrutamiento que depende del tipo de nodo, es decir si es un nodo manejado cuenta solo con la tabla de ruta de vecinos; y es un nodo JC o JCR, contará además de la tabla de vecinos con la tabla de adyacencia que "se forman cuando dos enrutadores vecinos intercambian información de enrutamiento y han sincronizado sus tablas" (Criollo & Ruilova, 2013).

NHeighborTable es la clase principal de las tablas de enrutamiento define las rutas de cada tabla, entre ellas la tabla de rutas para los nodos vecino y la tabla de rutas para los JC y JCR. Además se definen los métodos para crear, borrar y actualizar registros de las tablas de enrutamiento; como también para seleccionar el nodo JCR más adecuado, para generar paquete de actualización y de broadcast, todo esto permite tener actualizada todas las tablas de enrutamiento para un mejor envió de paquetes sin tener perdida de los mismos.

Las tablas de enrutamiento poseen algunas características importantes, a continuación se las menciona:

Los campos **my_clúster_head** y **my_clúster_head_backup** en la clase principal ayudan a identificar los nodos JC y JCR en la tabla de enrutamiento, dependiendo de nodos correspondientes.

En la clase **ntable_ent** se encuentra para cada nodo existe una tabla de ruta de vecinos.

Existe una tabla de JC en la que se define el *backbone* que contiene una dupla que posee entradas para los nodos JC y los nodos Gateway, así mismo la entrada **adjtable_ent** es una dirección del nodo JC adyacente y la entrada **nexthop_ent** nos ayuda para los nodos Gateway entre clúster y nodo JC adyacente.

Una entrada hacia un nodo destino puede poseer más de un camino de enrutamiento.

La clase **Event** ayuda a la interacción con el simulador mediante eventos y manejadores que son heredados por los atributos que se encuentran en la tabla de enrutamiento.

Cada clúster solo tiene permitido tener un Jefe de Clúster y a su vez un Jefe de Clúster de Respaldo, por tal motivo la clase **HNeighborContentionHandler** permite organizar los clúster definiendo los procesos que han sido invocados cuando existe un nodo JC en el mismo clúster.

HNeighborCFormationHandler define procesos para la formación de clúster, esta clase es invocada cuando un clúster pierde su nodo JC y su nodo JCR, para escoger s nuevo nodo JC.

La clase **HNeighborPeriodicHandler** realiza el mantenimiento periódico de las tablas de enrutamiento y de los clústers, mediante los métodos propios de la clase. Es importante tener actualizados las tablas y el clúster para que el envío de paquetes sea exitoso.

Por otro lado la clase **HNeighborTimeoutHandler** define el control de temporizadores para eventos que ya han expirado a través los métodos propios de esta clase.

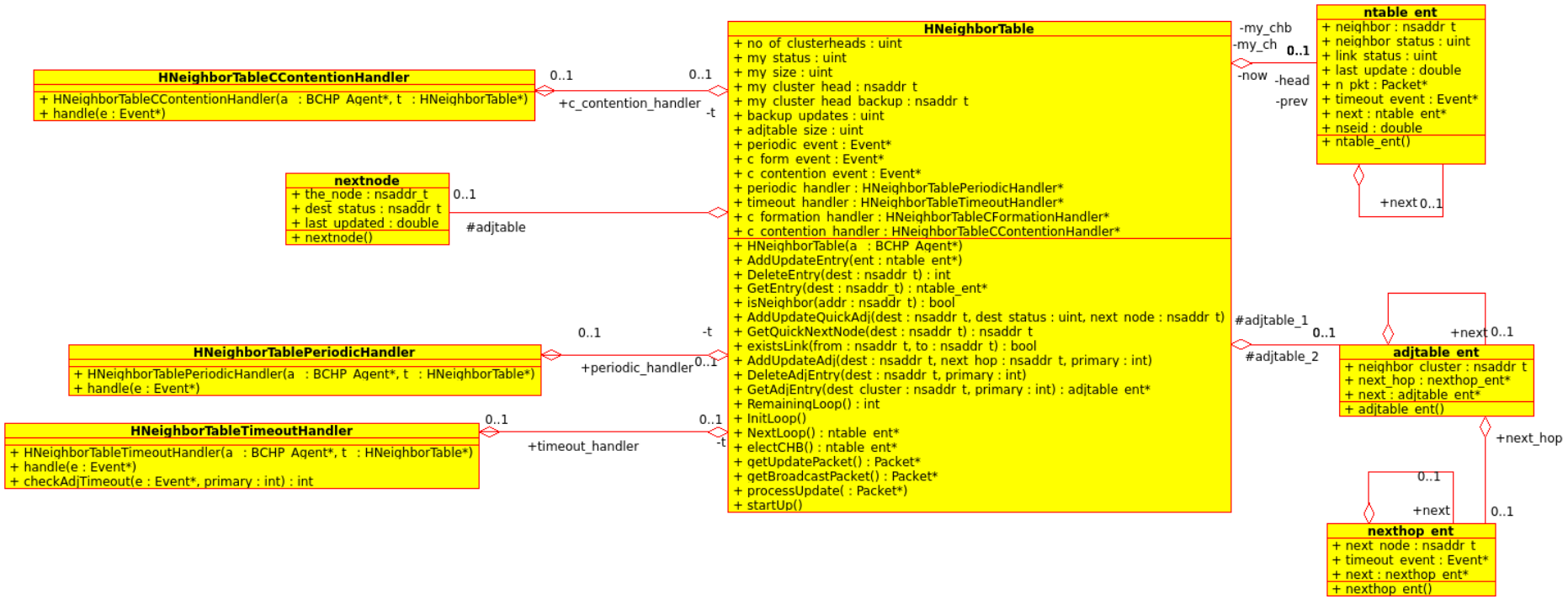


Figura 19: Diagrama de Clases de tablas de encaminamiento del protocolo BCHP

Fuente y Elaboración: La Autora

4.4. Algoritmos propuestos en la implementación del protocolo BCHP

Para la implementación y funcionamiento del protocolo BCHP se detallan a continuación los siguientes algoritmos:

4.4.1. Inicialización del nodo

Al momento que el nodo se inicia siempre comienza con un estado UNDECIDED o indeciso, debido a que no sabe el camino a escoger. Además según “las características de batería, movilidad y grado” (Torres, 2011) se obtiene una métrica la cual junto con la dirección del nodo servirán para la elección de los nodos JC, JCR, SA y SAR.

Las tablas de enrutamiento deben ser inicializadas por el nodo y enviadas a todos sus vecinos en forma de broadcast, después de haber informado a sus vecinos empezará la formación del clúster. El nodo JC y JCR llevan registros de cada uno de ellos para mejorar la disponibilidad de la red. Luego de ser inicializadas, se invocan los temporizadores para la formación del clúster, mantenimiento del clúster y expiración de entradas de las tablas y contención del clúster (Torres, 2011).

Cada uno de las tablas de enrutamiento son enviadas a través de mensajes HELLO, además incluye información del estado del nodo. (Véase figura 20)

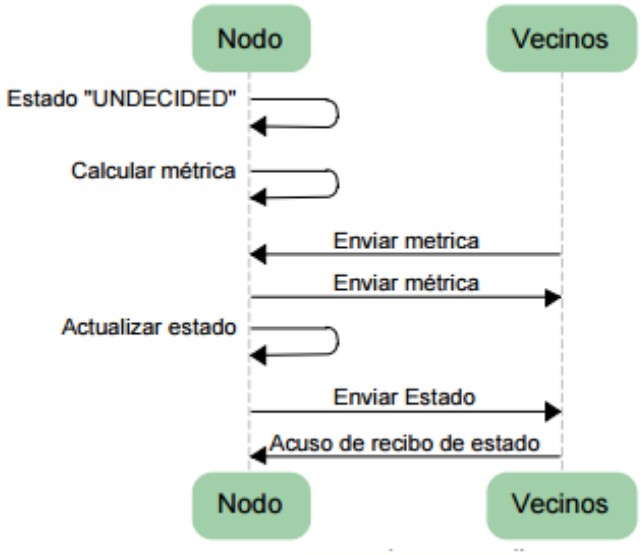


Figura 20: Inicialización del nodo

Fuente: (Torres, 2011)

Elaboración: (Torres, 2011)

4.4.2. Formación del clúster

Los menajes HELLO que son enviados por los nodos, entra el proceso de formación del clúster incluyendo la elección del nodo JC y JCR y la identificación de todos los clúster pertenecientes.

Se utiliza una métrica para la elección del nodo JC reflejada en el estado de cada uno de los nodos, y en el caso de existir más de dos nodos con la misma métrica se deberá usar el valor más bajo de identificador del nodo. (Véase figura 21)

En caso de cambiar el nodo de estado, forma un paquete BCHP y es enviado a todos sus nodos vecinos por medio de un mensaje broadcast (Torres, 2011).

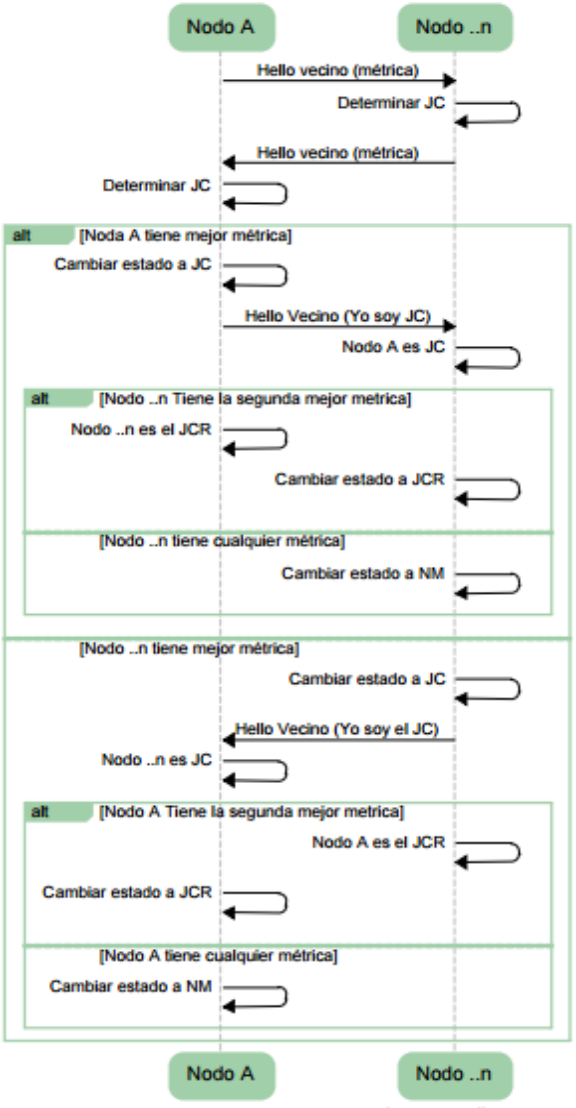


Figura 21: Formación del clúster.

Fuente: (Torres, 2011)

Elaboración: (Torres, 2011)

4.4.3. Inclusión de un nodo en el clúster

Por inicio o movilidad un nodo puede aparecer en el dominio del clúster enviando un mensaje HELLO Vecino con su estado a todos los nodos del clúster incluyendo el nodo JC; si este nodo posee una mejor métrica no es tomado en cuenta para ser el nodo JC, pero es tomado en consideración para ocupar los cargos del nodo JCR. Si un nodo posee una peor métrica pasa a ser un nodo NM del clúster y a su vez actualiza su información de pertenencia del clúster. (Véase figura 22)

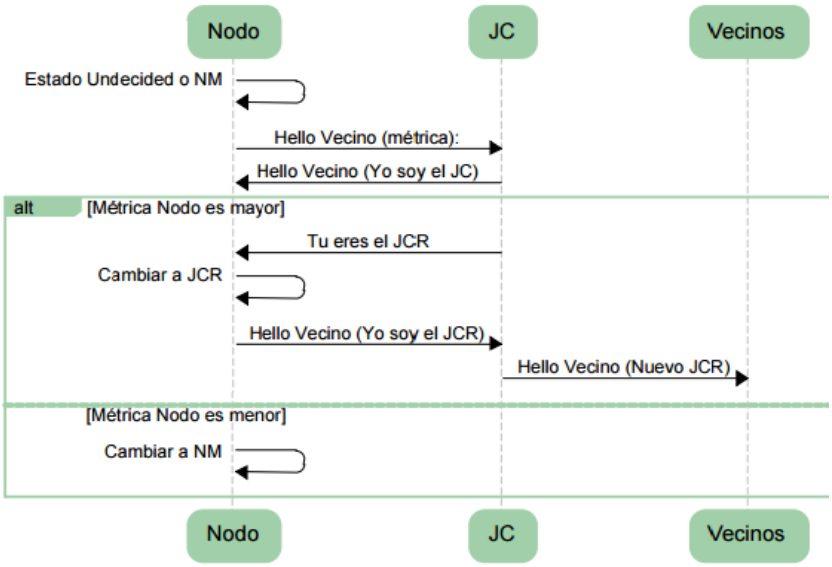


Figura 22: Inclusión de un nodo en el clúster

Fuente: (Torres, 2011)

Elaboración: (Torres, 2011)

4.4.4. Pérdida del nodo JC

Al momento que el nodo JC no envía información en un tiempo determinado al JCR, el nodo JCR cambia su estado a Jefe de Clúster e informa mediante un mensaje HELLO su nuevo estado a todos sus nodos. Esto también puede ocurrir cuando las entradas de las tablas de enrutamiento han expirado. (Véase figura 23)

El mismo caso, cuando un nodo NM no recibe información de su nodo JC dependerá de su métrica para proclamarse nodo JCR, pero se informará su nuevo estado por medio de un mensaje de broadcast.

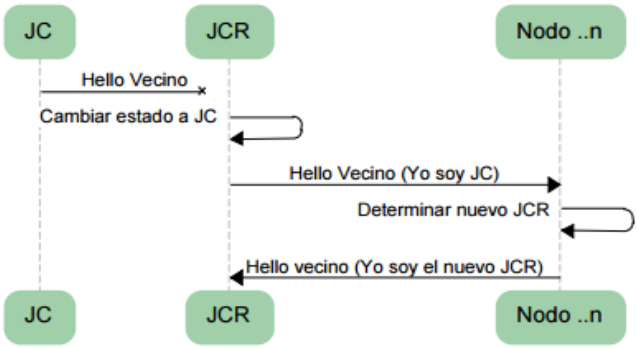


Figura 23: Pérdida del nodo JC

Fuente: (Torres, 2011)

Elaboración: (Torres, 2011)

4.4.5. Pérdida del nodo JCR

Los motivos por los cuales el nodo JCR no es detectado por el nodo JC puede ser que no se ha recibido información de gestión actualizada o los tiempos de actualización han vencido. Entonces el nodo JC de su tabla de enrutamiento selecciona la mejor métrica para ser el JCR e informa sus nuevas responsabilidades dentro del clúster (Torres, 2011). (Véase figura 24)

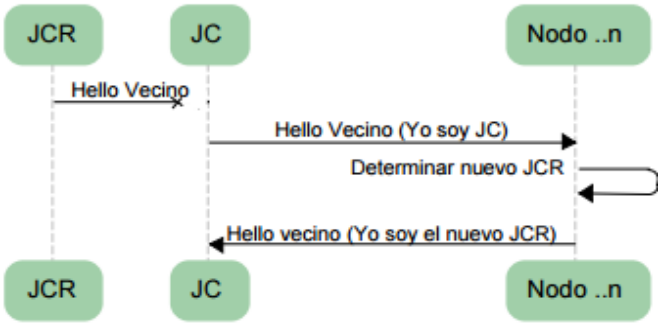


Figura 24: Pérdida del nodo JCR

Fuente: (Torres, 2011)

Elaboración: (Torres, 2011)

4.4.6. Mantenimiento de las tablas de enrutamiento.

Cuando un paquete llega a un nodo, se toma la métrica, el estado y la tabla de enrutamiento del nodo que envía el paquete y de esta manera el nodo receptor podrá actualizar su tabla de enrutamiento. Si un nuevo nodo envía información, se agrega una entrada a la tabla de enrutamiento y se determina su métrica para saber si puede pasar a tomar el cargo de nodo JCR, si posee una mejor métrica el estado del nodo de la nueva entrada es modificado e informa al resto para su actualización respectiva (Torres, 2011).

CAPITULO 5

ANALISIS Y DISCUSIÓN DE RESULTADOS

5. Análisis y discusión de resultados

Para la realización del análisis y discusión de resultados del presente trabajo de titulación consta de 3 partes:

- Proceso de simulación que se realiza para la respectiva implementación de los protocolos dentro de la herramienta de simulación.
- Discusión de resultados obtenidos en la herramienta NS2.35 de los protocolos BCHP y CBRP con sus respectivos antecedentes de simulación y sus parámetros de medición, los cuales son rendimiento, tasa de envío de paquetes, pérdida que paquetes, retardo promedio, sobrecarga de protocolo.
- Para finalizar se realiza una comparación de versiones de la herramienta de simulación implementando los protocolos tanto en ns2.35 como ns2.34 para verificar el comportamiento en cada versión.

5.1. Proceso de simulación

El presente trabajo de titulación muestra el proceso de simulación que consta de 3 fases secuenciales: definición de escenarios, simulación y post-simulación.

En la figura 25 se puede observar el proceso de simulación. Entre los elementos que representan dicho proceso tenemos:

- **Flujo de información:** representa las fechas que delimitan el origen y el destino de la información.
- **Procesos:** se representan mediante rectángulos con puntas redondeadas, los mismos que tienen argumentos de entrada y a su vez generan información que puede ser usada en otros procesos (Torres, 2011).
- **Entradas y Salidas:** son representados mediante rectángulos y poseen información necesaria de otro proceso.

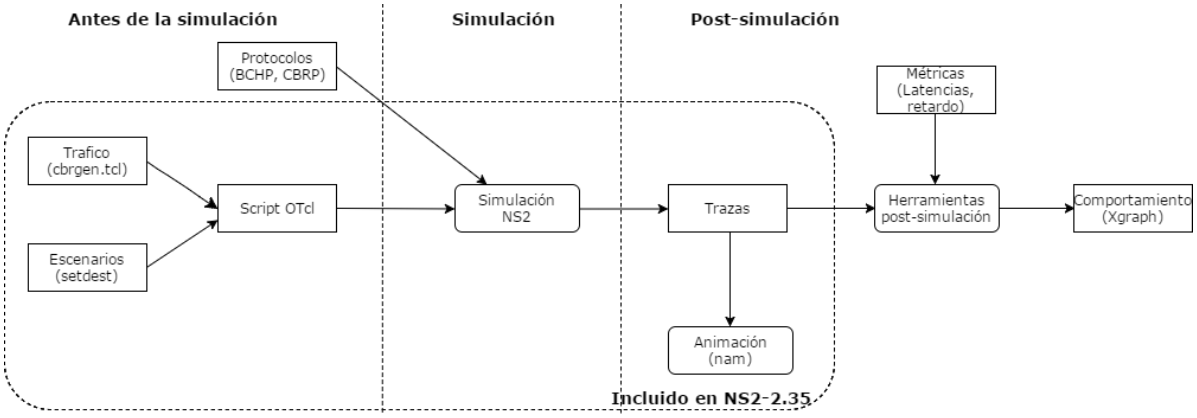


Figura 25: Proceso de simulación en NS2

Fuente: (Torres, 2011).

Elaboración: La Autora

En el presente trabajo de titulación se han implementado en NS2 los protocolos CBRP y BCHP, los mismos que se encuentran fuera del rectángulo mayor ya que son elementos externos a la herramienta de simulación. Además de utilizar herramientas para los procesos de animación de cada escenario creado y para la obtención de los comportamientos de cada uno de ellos.

- **Pre-simulación**

Es necesario satisfacer todos los requisitos de entrada para la correcta simulación. A continuación se describe lo que se realizó en la primera etapa:

- ✓ Analizar la estructura y el funcionamiento de la herramienta NS2 para posteriormente realizar la instalación, configuración y validación de la misma. Revisar ANEXO 1.
- ✓ Implementar los protocolos a la herramienta de simulación. Revisar ANEXO 2
- ✓ Definir la topología de los escenarios, esto incluye: el tamaño del plano, ubicación inicial de cada nodo, el modelo de movilidad. Con la ayuda de la herramienta *scengen* desarrollada por (Qiming, 2002), se genera la topología que exprese el modelo de movilidad deseado. ANEXO 3
- ✓ Definir la cantidad de conexiones por cada escenario y los patrones de tráfico a utilizar en la simulación. Para este proceso se utilizó la herramienta incluida en NS2 llamada *cbrgen. ANEXO 3*
- ✓ La herramienta Umbrello (Umbrello, 2010) es utilizada para obtener y analizar el modelo de clases de la herramienta de simulación y de los protocolos implementados.

- ✓ Para concluir con los requisitos de entrada se debe generar la secuencia de comandos de simulación en el lenguaje OTcl, es aquí donde se configura las características iniciales de los nodos, se inicializa el simulador, se llaman a los archivos de definición de escenarios y movilidad (Torres, 2011). Además también se definen el trazado y los nombres de los archivos para almacenar la información. Revisar ANEXO 4

- **Simulación**

En esta etapa se ejecutan las secuencias de comandos una por una hasta terminar. Además se instancia los objetos de las clases, el simulador y el organizador.

- **Post-simulación**

Esta etapa contiene los siguientes procesos:

- ✓ La herramienta NAM (Network Animator) es utilizada para la animación de las simulaciones. Esta herramienta es propia de NS2 y para ejecutarse toma como base el archivo .nam generado en cada una de las simulaciones.
- ✓ En el proceso de post-simulación se permite obtener un resultado del comportamiento del protocolo. Para generar éstos resultados se utilizó archivos awk para generar scripts de cada uno de los escenarios. ANEXO 6
- ✓ Para mostrar los resultados de forma gráfica se utilizan tablas y gráficos estadísticos para poder realizar una comparación entre los protocolos implementados.

5.2. Discusión de resultados en NS2.35

En esta sección se detalla de manera general las características, topología y atributos de los nodos que se llevarán a cabo para la simulación de cada escenario.

- Se implementó en la herramienta de simulación NS-2.35
- El área de simulación es 500m x 500m
- Escenarios creados son 10, 20, 30, 40, 50, 60, 70, 80 nodos y con 5, 10, 15, 20, 25, 30, 35, 40 conexiones respectivamente.
- Los protocolos de la capa de red implementados son CBRP y BCHP.
- El protocolo de la capa de transporte simulado es CBR (Constant Bit Rate)

Una vez concluida la implementación de los protocolos, se obtienen los archivos de trazas que contiene la información referente a cada uno de los procesos de simulación.

A los protocolos CBRP y BCHP implementados en la versión NS-2.35 de la herramienta de simulación, se les realiza una comparación para determinar cuál de estos protocolos es mejor en base a los siguientes parámetros de medición:

- Rendimiento
- Tasa de envió de paquetes
- Pérdida de paquetes
- Retardo promedio
- Sobrecarga de protocolo

5.2.1. Rendimiento

Es el total de paquetes recibidos exitosamente por un destino incluyendo los paquetes de aplicación y enrutamiento. Mide la eficiencia de un protocolo de enrutamiento. Para determinar que una red ha tenido un excelente rendimiento depende de los valores de *throughput*, ya que mientras más alto sea el valor la entrega de paquetes será confiable.

$$TH = \frac{Br}{Ts}[kbps]$$

Donde:

TH: resultado *de throughput*

Br: Bytes recibidos

Ts: Tiempo de simulación.

Se ha medido el rendimiento de toda la red variando la cantidad de nodos para diferentes escenarios y para cada protocolo implementado. Como se puede observar en las figuras 26, 27, 28, 29, 30, 31, 32 y 33 se obtiene el rendimiento para 10, 20, 30, 40, 50, 60, 70 y 80 nodos respectivamente.

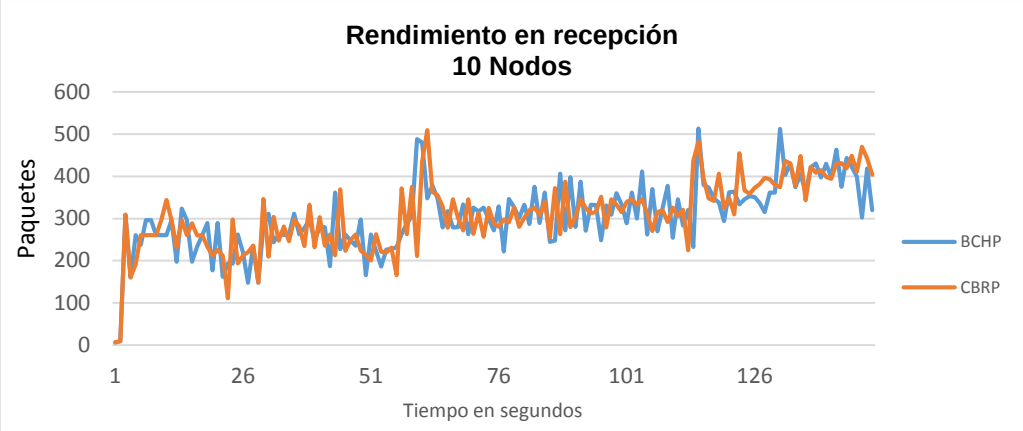


Figura 26: Rendimiento de 10 nodos

Fuente y Elaboración: La Autora

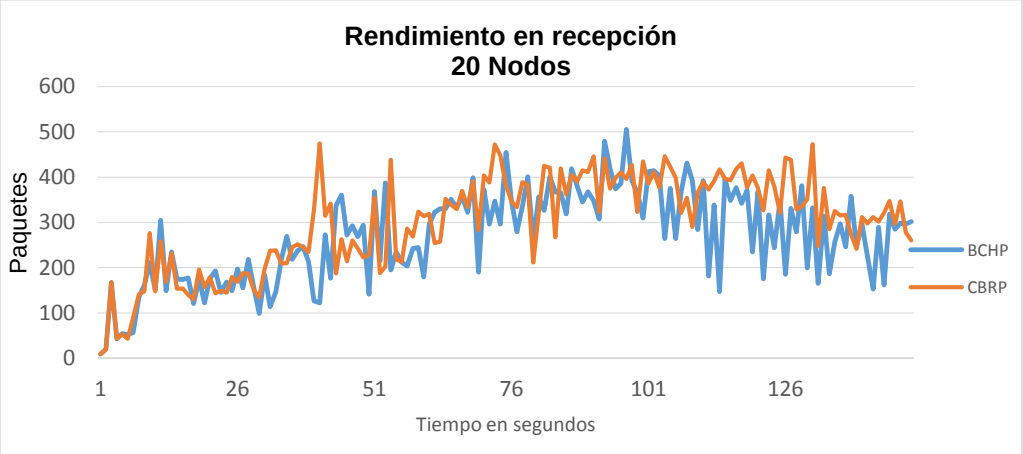


Figura 27: Rendimiento con 20 nodos

Fuente y Elaboración: La Autora

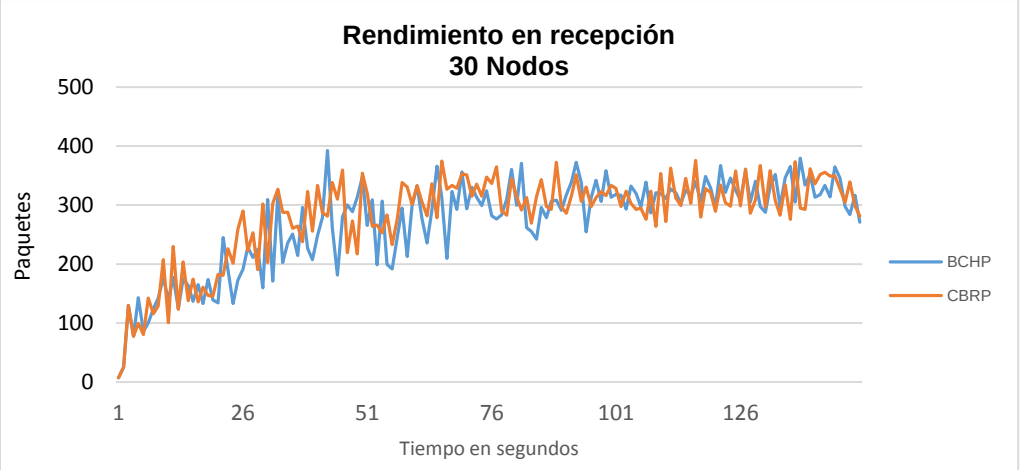


Figura 28: Rendimiento de 30 nodos

Fuente y Elaboración: La Autora

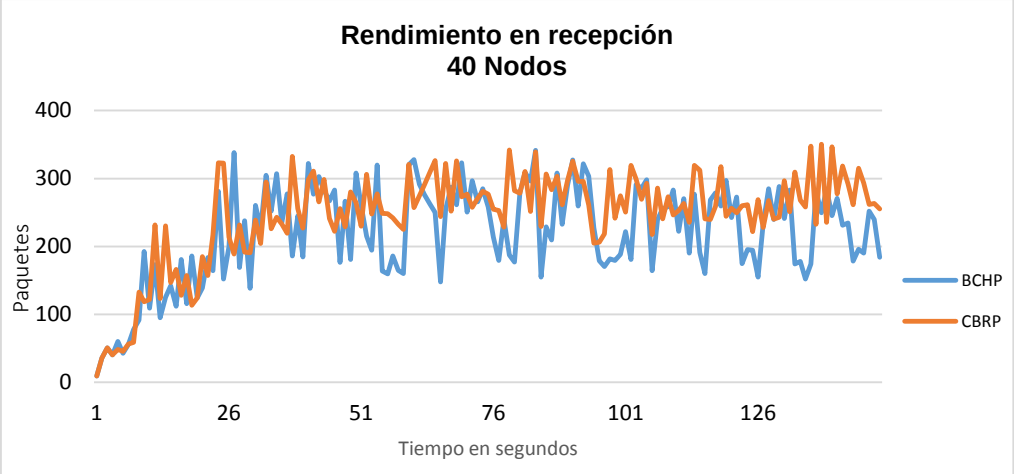


Figura 29: Rendimiento con 40 nodos

Fuente y Elaboración: La Autora

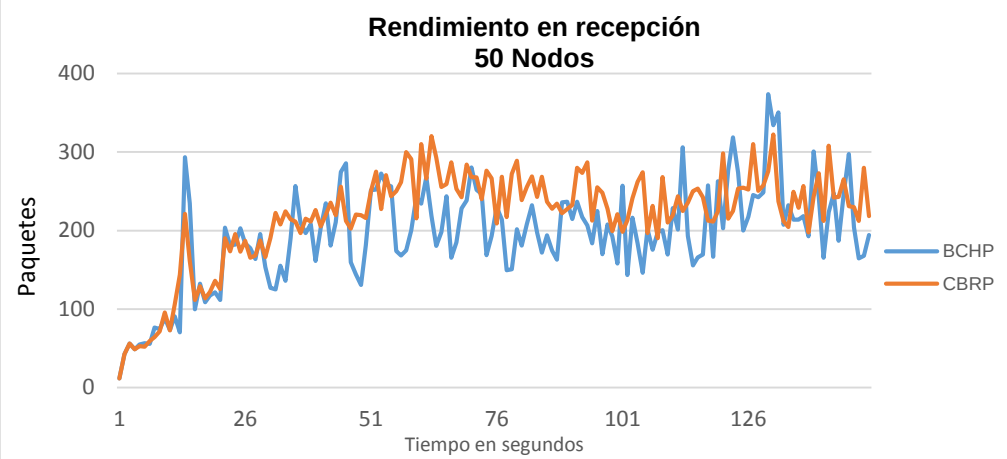


Figura 30: Rendimiento de 50 nodos

Fuente y Elaboración: La Autora

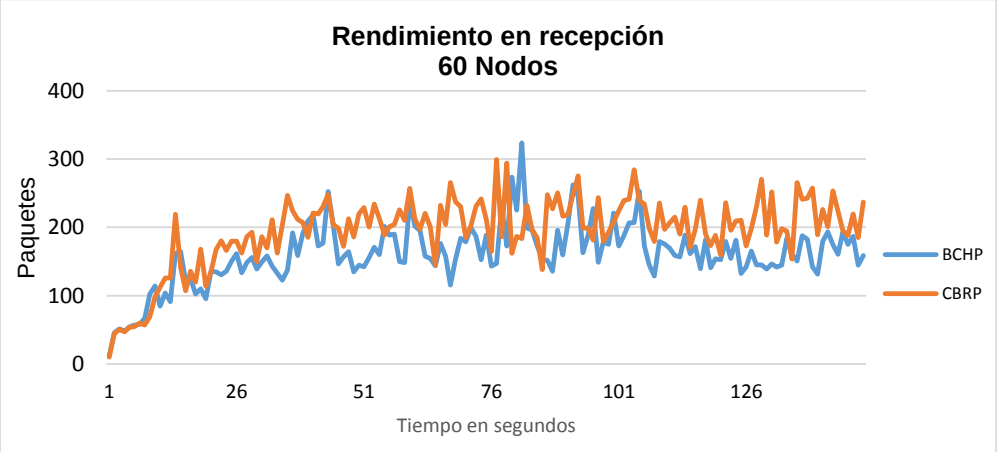


Figura 31: Rendimiento con 60 nodos

Fuente y Elaboración: La Autora

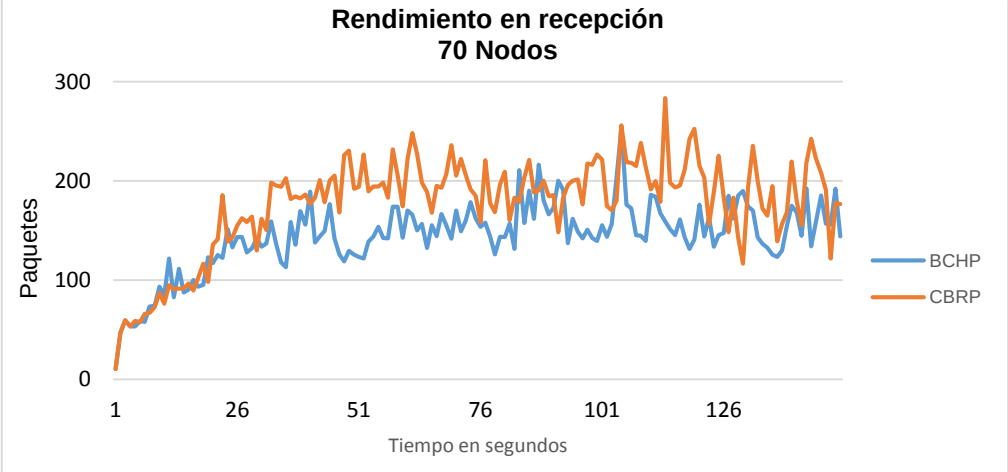


Figura 32: Rendimiento 70 nodos

Fuente y Elaboración: La Autora

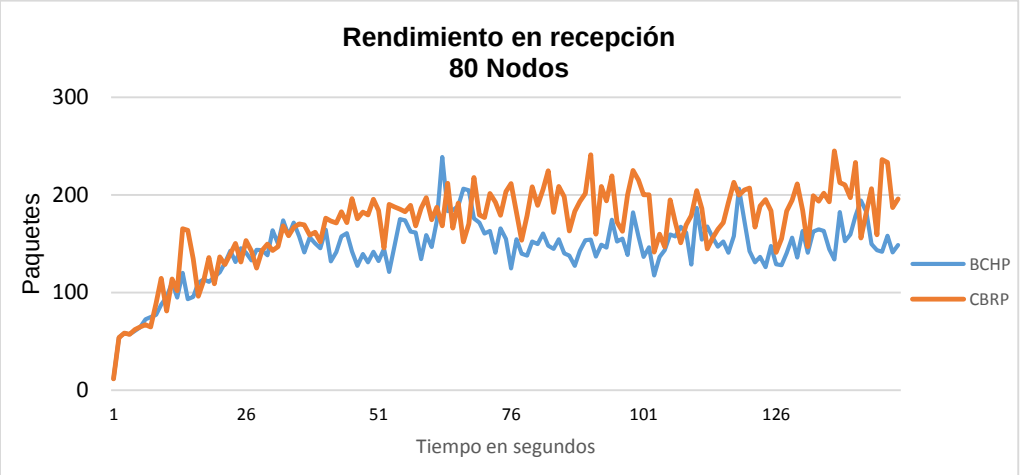


Figura 33: Rendimiento con 80 nodos

Fuente y Elaboración: La Autora

La información que se visualizan en las figuras anteriores muestran la conducta del tamaño de ventana TCP durante el envío desde el nodo origen (0) hasta el nodo de los protocolos de enrutamiento jerárquico implementados. En el eje x de cada una de gráficas se representa el tiempo de simulación de cada escenario (150seg) y en el eje y se representa los paquetes recibidos.

Se muestra en los escenarios de 10, 20, 30, 40, 50, 60, 70 y 80 que el rendimiento incrementa a medida que avanza el tiempo de simulación, el protocolo CBRP posee un promedio de 8.44%, 6.23%, 9.22% y 6.31% respectivamente, sin embargo cabe mencionar que tiene un mayor rendimiento debido a que genera menor cantidad de paquetes de enrutamiento.

5.2.2. Tasa de envío de paquetes

La tasa de envío de paquetes se obtiene entre el número de paquetes enviados para el número de paquetes recibidos:

$$TPE = \frac{Pr}{Pe} * 100$$

Donde:

TPE: Tasa de paquetes entregados

Pr: Paquetes recibidos

Pe: Paquetes enviados

La cantidad de paquetes entregados varía en cada protocolo dependiendo del tiempo de comunicación que realicen en un lapso de tiempo. Debido a que la información y mantenimiento del clúster necesita el intercambio de paquetes para mantener su información actualizada, en la figura 34 se muestra la tasa de envió de todos los paquetes donde se visualiza el porcentaje de cada uno de los escenarios propuestos.

En la figura 35 se muestra la tasa de envío de paquetes a nivel de aplicación. Se obtiene dividiendo los paquetes de aplicación recibidos para los paquetes de aplicación enviados, sin tomar en cuenta los paquetes de la capa de encaminamiento.

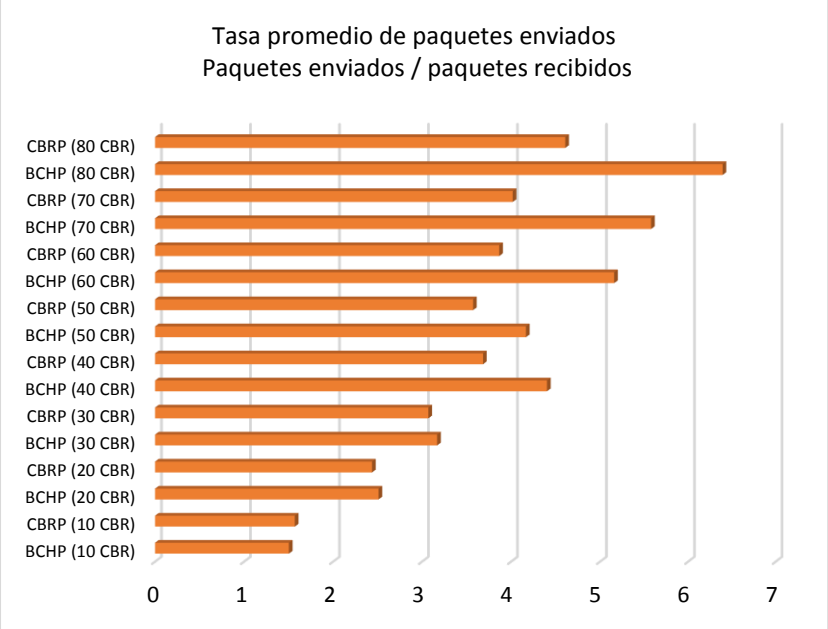


Figura 34: Tasa de envío de paquetes.

Fuente y Elaboración: La Autora

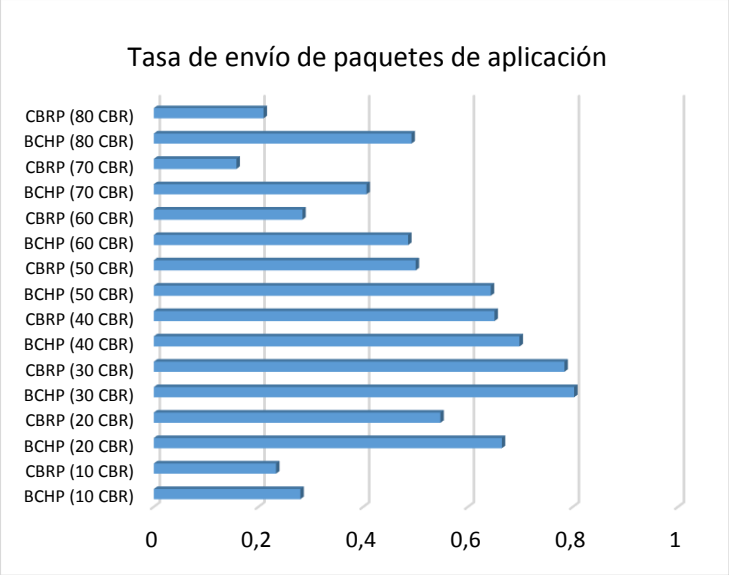


Figura 35: Tasa de envío de paquetes de aplicación

Fuente y Elaboración: La Autora

En la imagen 34 se pude visualizar que el protocolo BCHP posee una mejor tasa de envío de paquetes con respecto al CBRP con un 6.05%. Por otro lado, en la figura 35 se puede observar que el protocolo BCHP también posee una mejor entrega de paquetes de aplicación. Por lo tanto BCHP tiene un mejor envió de paquetes generales como paquetes de aplicación, siendo así mismo más efectivo que el protocolo CBRP.

5.2.3. Pérdida de paquetes

La pérdida de paquetes es la cantidad de paquetes eliminados por los nodos intermedios a causa de efectos producidos por la movilidad de los nodos, expiración de temporizadores, destinos inalcanzables o borrados por ARP (Protocolo de resolución de dirección).

En la figura 36 se muestran la cantidad de tipos de pérdida de paquetes por cada uno de los escenarios creados. Donde:

arq: responsable de encontrar la dirección de hardware que corresponde a una determinada dirección IP.

tout: falla a nivel de capa de enlace.

nrte: no existe ruta.

ttl: tiempo de vida expirado del paquete.

Se observa que existen una mayor cantidad de paquetes perdido debido a la expiración del tiempo de simulación en el protocolo BCHP con respecto a CBRP

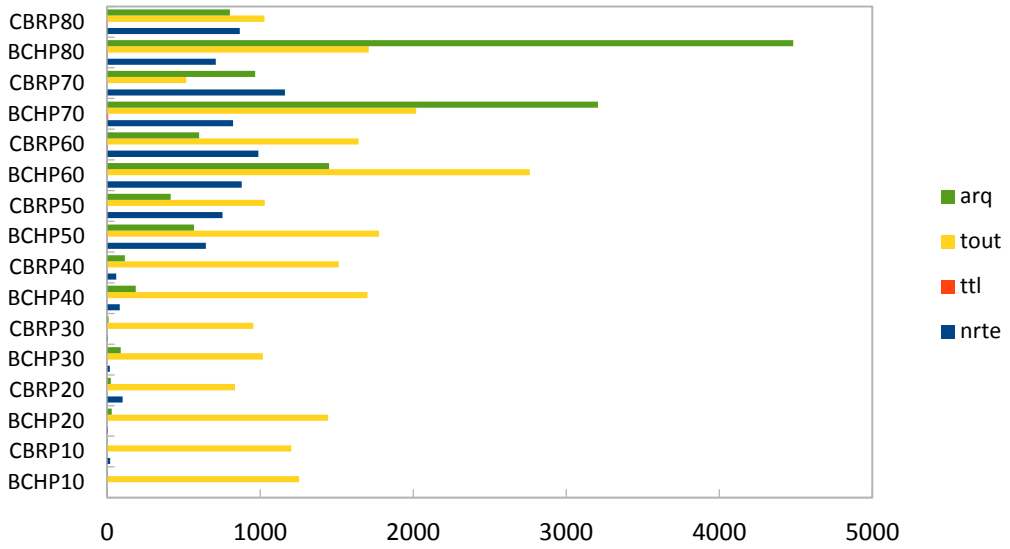


Figura 36: Pérdida de paquetes

Fuente y Elaboración: La Autora

5.2.4. Retardo promedio

Esta métrica se refiere al tiempo que tarda un paquete en transmitirse a través de una red desde la fuente hasta el destino.

En la figura 37 el protocolo BCHP tiene un retardo promedio de 9.21% y el protocolo CBRP posee un retardo promedio de 21.48%, dando como resultado que el protocolo BCHP tiene un retardo mejor de aproximadamente 12.26% con respecto a CBRP porque se utiliza como trafico conexiones con TCP.

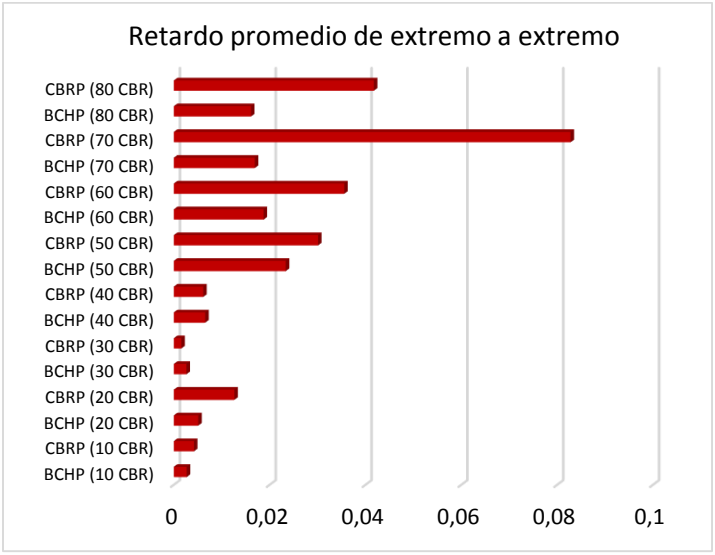


Figura 37: Retardo extremo a extremo

Fuente y Elaboración: La Autora

5.2.5. Sobrecarga de protocolo

La sobrecarga del protocolo se obtiene dividendo el número total de paquetes de enrutamiento enviados durante la simulación para el número de paquetes de aplicación. Esta métrica determina la eficiencia de un protocolo de enrutamiento Ad-Hoc (Torres, 2011). Mientras el valor esté más cerca de 0, el protocolo será más eficiente.

$$SP = \frac{Pr}{Pt}$$

Donde:

Pr: paquetes de enrutamiento transmitidos

Pt: paquetes de aplicación transmitidos

En la figura 38 se puede observar que los protocolos CBRP y BCHP generan un rango de tráfico casi similar debido a las operaciones de mantenimiento de clúster que utilizan. Sin embargo el protocolo BCHP es mejor con 1.16% que el protocolo CBRP debido a la utilización de Jefe Clúster de respaldo que disminuye el intercambio de información para meter el clúster.

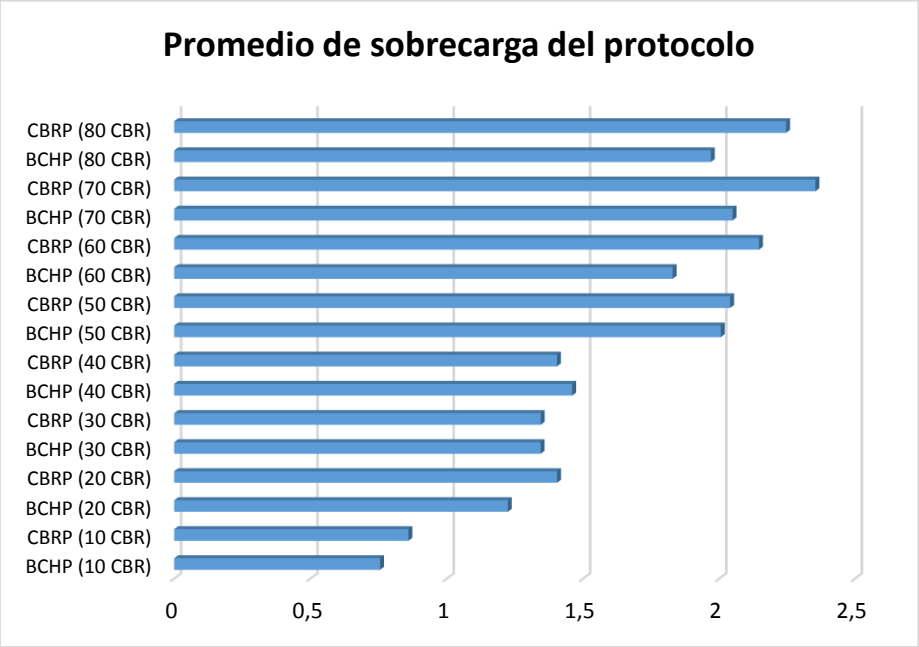


Figura 38: Sobrecarga del protocolo

Fuente y Elaboración: La Autora

5.3. Comparación de la implementación del protocolo en las versiones de NS2.34 y NS2.35

A continuación se realiza una comparación de los resultados en la herramienta de simulación NS2, una versión anterior (Torres, 2011), con resultados realizados que poseen los mismo escenarios de simulación en la versión NS2.35.

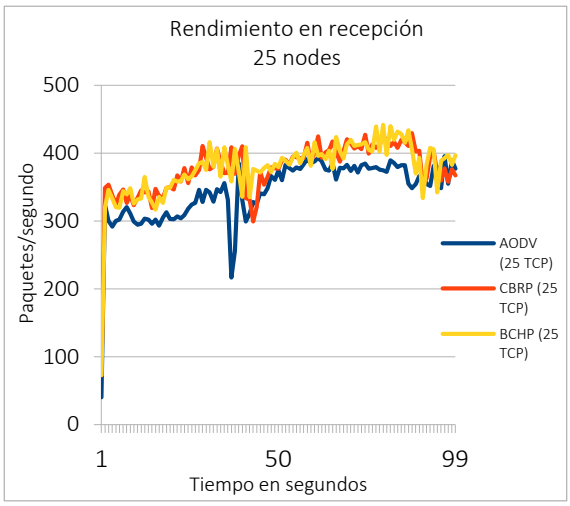
- Área de simulación: 600m x 500m
- Cantidad de nodos: 25, 40 60 y 90 nodos
- Cantidad de conexiones: 20
- Tiempo de simulación: 100 segundos
- Protocolos de la capa de transporte: TCP y CBR
- Protocolos de la capa de red: CBRP, BCHP y AODV

Esta comparación es para verificar que el funcionamiento es similar para las diferentes versiones de la herramienta NS2 teniendo en cuenta la mismas métricas de: rendimiento, tasa de envío de paquetes, pérdida de paquetes, retardo promedio y sobrecarga de protocolos, o si difieren estas métricas y por qué.

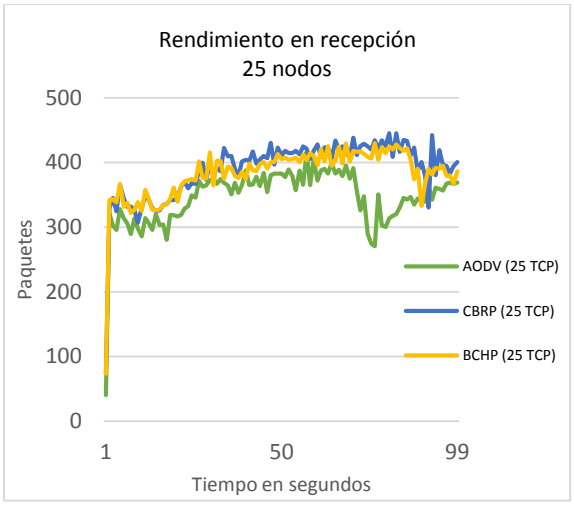
En cada figura existen dos literales para la respectiva comparación de las métricas según el versionamiento de la herramienta. Teniendo literal a) versión NS2.34 y literal b) versión NS2.35

5.3.1. Rendimiento

Como se puede observar en la figura 39, 40, 41 y 42 los protocolos de enrutamiento jerárquico funcionan entre un 10% a un 15% mejor que el protocolo AODV en las dos versiones de la herramienta de simulación. Sin embargo en el literal b) de cada una de las figuras el protocolo CBRP es ligeramente más estable en la herramienta de simulación versión NS2.35 a diferencia de la versión anterior porque genera más paquetes de enrutamiento.



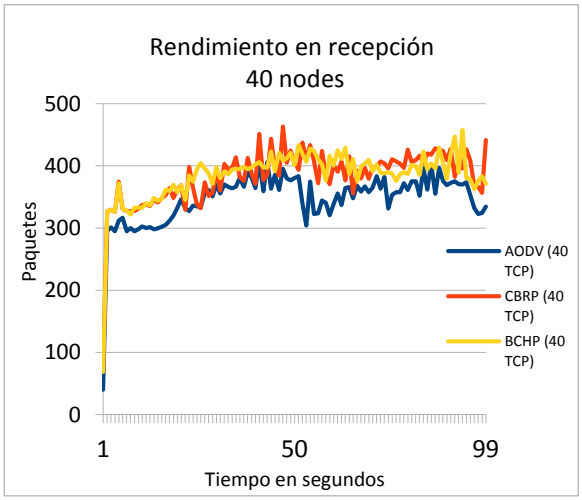
a)



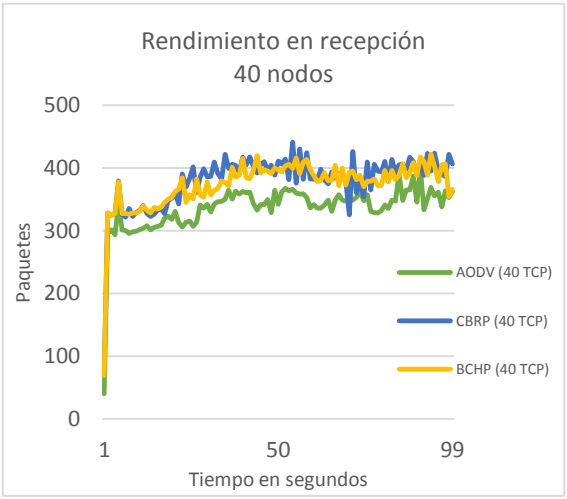
b)

Figura 39: Rendimiento de 25 nodos

Fuente y Elaboración: La Autora



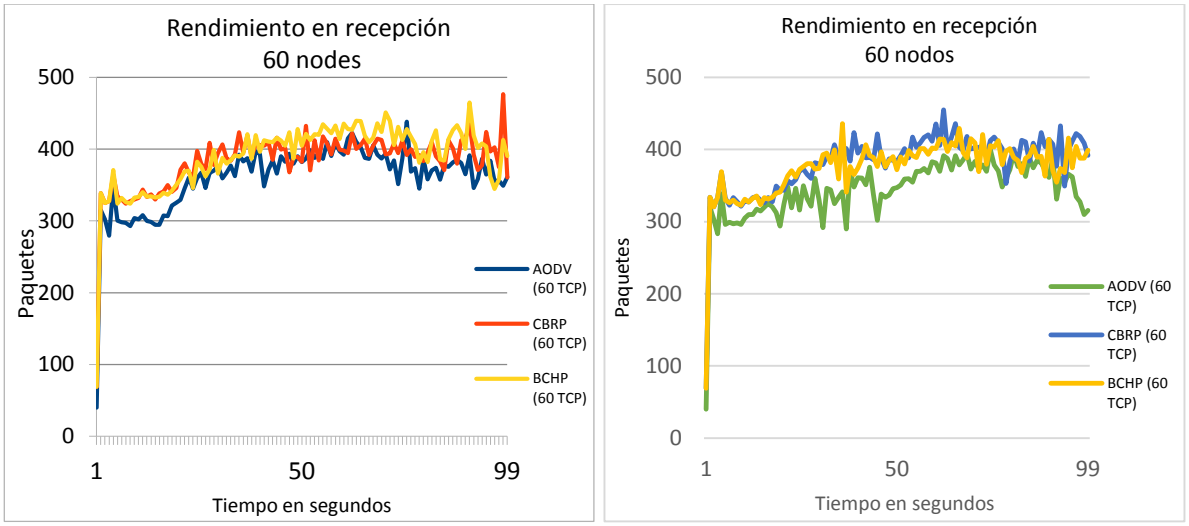
a)



b)

Figura 40: Rendimiento de 40 nodos

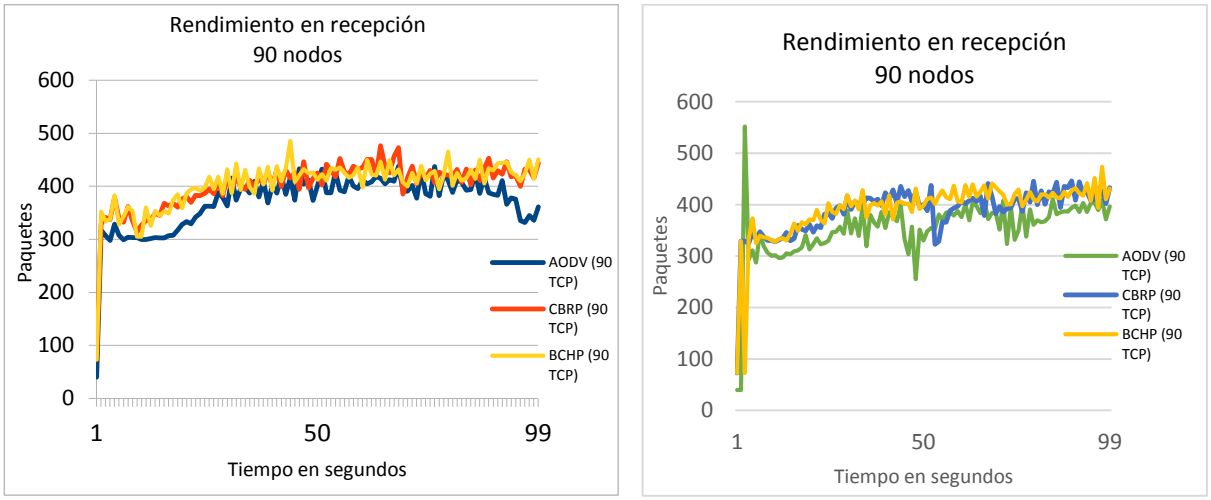
Fuente y Elaboración: La Autora



a) b)

Figura 41: Rendimiento de 60 nodos

Fuente y Elaboración: La Autora



a) b)

Figura 42: Rendimiento de 90 nodos

Fuente y Elaboración: La Autora

5.3.2. Tasa de envío de paquetes

En la figura 43 se muestra la tasa promedio del envío de paquetes. Los protocolos de enrutamiento jerárquico CBRP y BCHP tienen una tasa de envió de un 50% mayor que el protocolo AODV en las dos versiones y además estos protocolos poseen un promedio similar de paquetes enviados sin existir diferencia significativa en ambas versiones. Por otra parte en la figura 44 se muestra la tasa de envío de paquetes a nivel de aplicación sin tomar en cuenta

la capa de enrutamiento, el protocolo AODV al usar conexiones TCP es levemente mejor que los protocolos CBRP y BCHP, pero al usar conexiones CBR todos los protocolos mantienen el mismo comportamiento en ambas versiones según cada uno de sus literales.

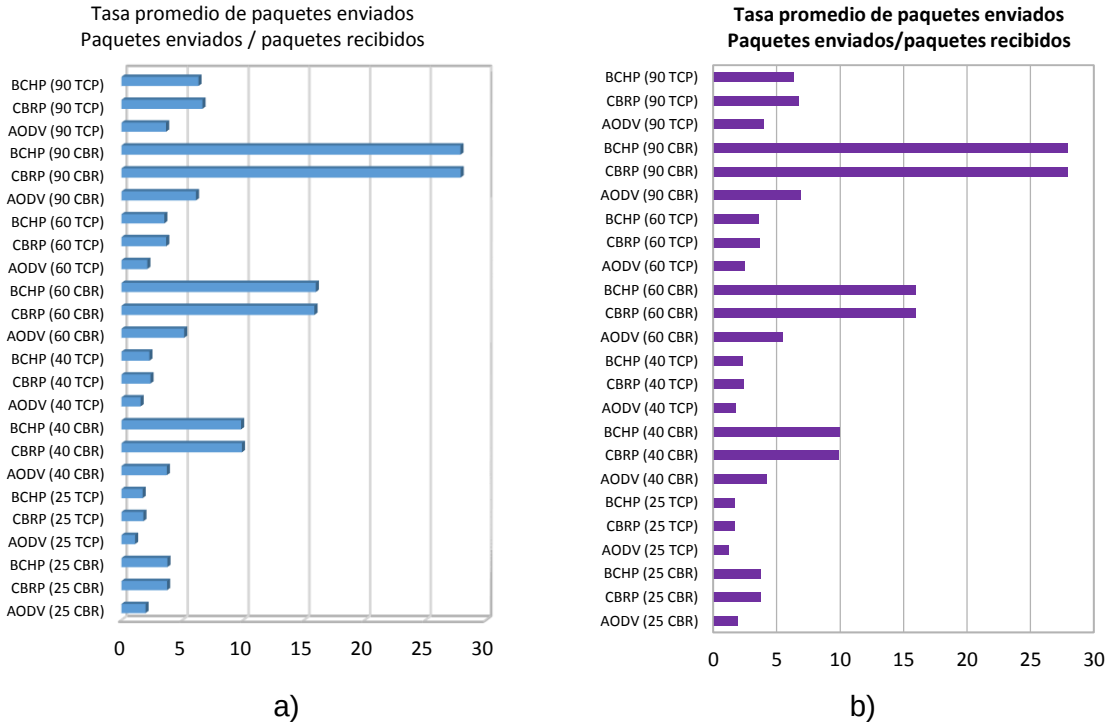


Figura 43: Tasa de envío de paquetes

Fuente y Elaboración: La Autora

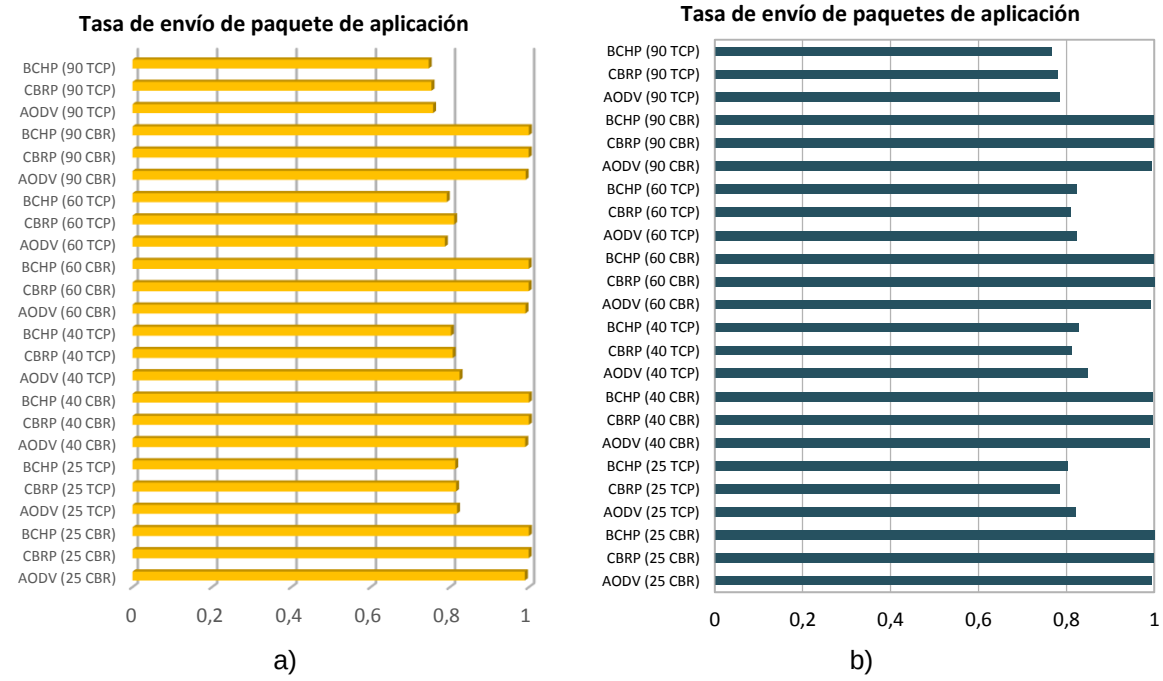


Figura 44: Tasa de envío de paquetes de aplicación

Fuente y Elaboración: La Autora

5.3.3. Pérdida de paquetes

En la figura 45 se puede observar que el protocolo BCHP tiene menor cantidad de paquetes borrados en las dos versiones implementadas, a diferencia del protocolo AODV que tiene una cantidad considerable de paquetes borrados. Para BCHP en el literal a) posee menor cantidad de paquetes borrados cuando no existe ruta, lo que demuestra la inclusión de JCR mejora la disponibilidad de la red (Torres, 2011); mientras en el literal b) BCHP posee menor cantidad de paquetes borrados por el protocolo ARP, lo que determina que este protocolo encuentra más rápido la dirección de hardware.

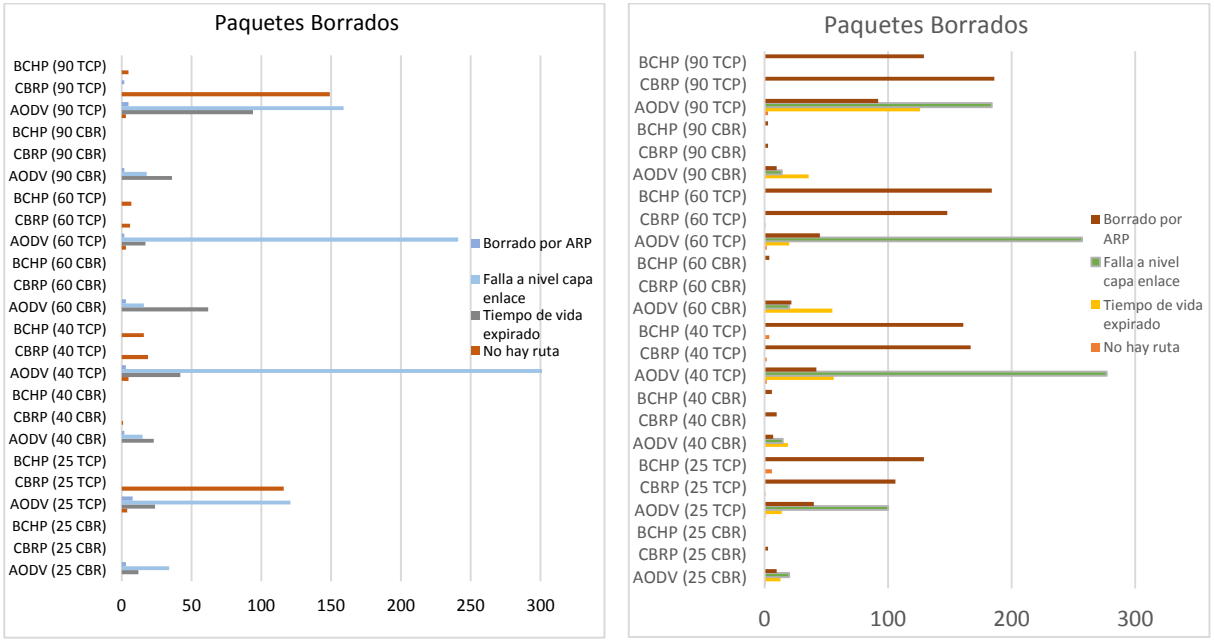


Figura 45: Perdida de paquetes

Fuente y Elaboración: La Autora

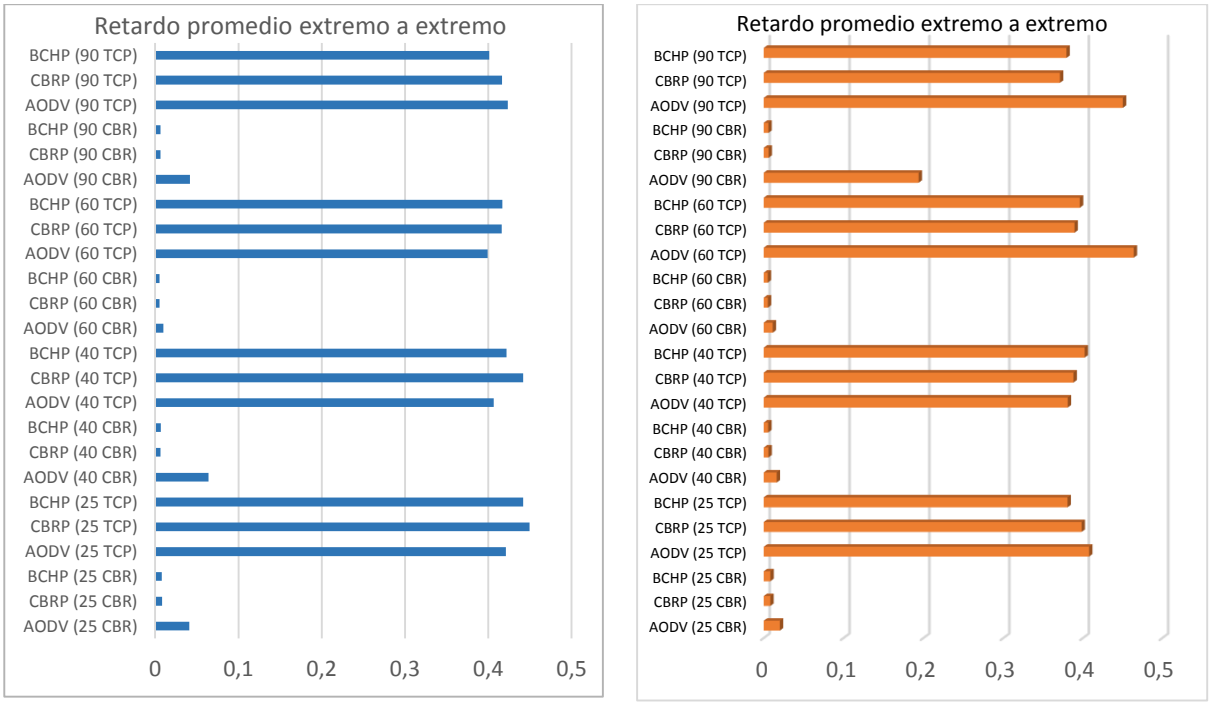
5.3.4. Retardo promedio

En la figura 46 el protocolo BHCP es mejor que el protocolo CBRP con un 1% con conexiones TCP en ambas versiones de la herramienta. En cambio cuando se utiliza conexiones CBR el protocolo AODV es mejor que los protocolos de enrutamiento jerárquico CBRP y BCHP, además los protocolos de enrutamiento jerárquico no difieren su retardo con las conexiones CBR en ambas versiones.

5.3.5. Sobrecarga del protocolo

En la figura 47 se puede visualizar que los protocolos BCHP y CBRP generan mayor tráfico de enrutamiento en conexione TCP como en conexiones CBR a diferencia del protocolo AODV y es debido a las operaciones de mantenimiento de clúster que poseen los protocolos de enrutamiento jerárquico. Como se puede observar en los literales el protocolo BCHP es

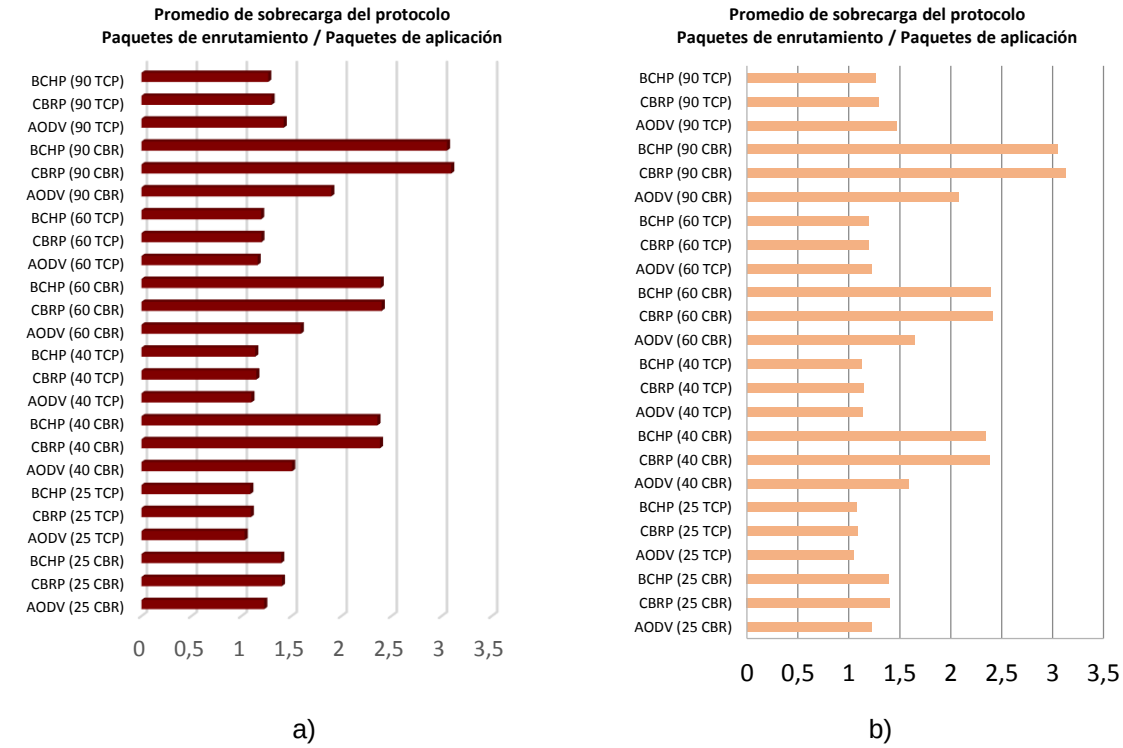
mejor que CBRP tanto en conexione TCP y CBR para ambas versiones de la herramienta de simulación.



a) b)

Figura 46: Retardo promedio de extremo a extremo

Fuente y Elaboración: La Autora



a) b)

Figura 47: Sobrecarga de protocolo

Fuente y Elaboración: La Autora

5.3.6. Variación del retardo del paquete

Como se puede visualizar en la figura 48 se obtuvo el promedio de variación del retardo de paquete. Los protocolos CBRP y BCHP poseen un mejor funcionamiento con respecto al protocolo AODV. El literal a) que corresponde a la versión NS2.34 el protocolo BCHP funciona mejor que CBRP cuando la densidad de los nodos es de 25, 40 y 60 nodo, en cambio en el literal b) que corresponde a la versión NS2.35 el protocolo CBRP es mejor que el protocolo BCHP en la misma densidad de nodos.

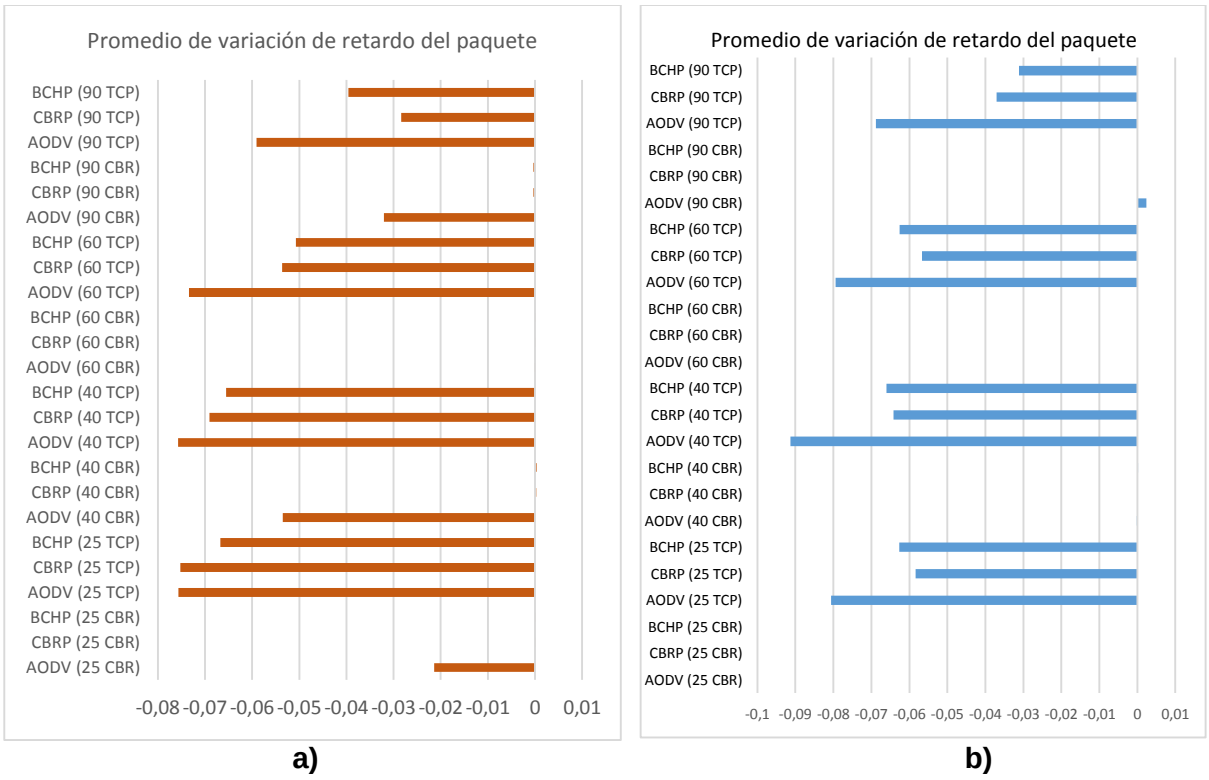


Figura 48: Variación del retardo del paquete

Fuente y Elaboración: La Autora

Comparando cada una de las métricas en las dos versiones de NS2, se puede determinar que en la versión 2.34 el protocolo BCHP es ligeramente mejor en rendimiento, tasa de envío de paquetes, pérdida de paquetes, retardo promedio, sobrecarga de protocolos y variación del retardo del paquete. Sin embargo, en la versión 2.35 el protocolo BCHP es también ligeramente mejor sólo en tasa de envío de paquetes, pérdida de paquetes, retardo promedio, sobrecarga de protocolos, pero en rendimiento y variación del retardo del paquete el protocolo CBRP es mejor que BCHP.

TRABAJOS FUTUROS

A partir del trabajo de titulación provienen varias líneas de trabajo a realizarse, las cuales quedan como recomendaciones que pueden realizarse en un fututo trabajo. Desde el punto de vista de este trabajo las más importantes son:

- Las redes MANET ofrecen un sin número de aspectos a investigar que no deberían ser limitaciones para desarrollar nuevos protocolos o a su vez mejorar significativamente alguno de los protocolos ya existentes, ya que pueden ser base de aplicaciones que permitan colaborar a situaciones emergentes o a su vez la determinación de rutas de escape.
- Debido a que la simulación fue realizada sin tomar en cuenta variables propias en un ambiente real, se recomienda la implementación de protocolos en dispositivos reales, es decir en teléfonos móviles, etc., para comprobar con los resultados obtenidos del presente trabajo.
- Es importante conocer todas las herramientas de simulación y que tan obsoletas están, para de acuerdo a eso hacer la selección respectiva y el aporte sea más significativo.
- Es importante conocer el funcionamiento de cada uno de los protocolos para verificar la necesidad que se puede mejorar y así el desarrollo sea significativo para la sociedad.
- Debido a que pocos son los protocolos que se encuentran implementados en las herramientas de simulación, es importante la implementación y el análisis de nuevos protocolos para la reutilización y la mejora de los mismos.

CONCLUSIONES

A partir de los resultados obtenidos en el capítulo 5 se analizan cualitativamente y cuantitativamente los protocolos BCHP y CBRP. Los beneficios de estos protocolos se verifican a través de los siguientes parámetros: rendimiento, tasa de envío de paquetes, pérdida de paquetes, retardo promedio y sobrecarga de protocolo.

Se puede concluir que los objetivos previamente planteados se han cumplido satisfactoriamente:

1. Como objetivo general se pudo implementar los protocolos de enrutamiento jerárquico BCHP y CBRP en la versión NS-2.35.

2. Como objetivos específicos:

   2.1. Luego de una comparación entre las herramientas de simulación existentes, se seleccionó NS2 debido a que: 1) principalmente es aceptada ampliamente por la comunidad mundial de docentes e investigadores, 2) tiene protocolos ya implementados y 3) se tiene los conocimientos necesarios y experiencia para la implementación de los nuevos protocolos seleccionados.

   2.2. Se definió un proceso para la inclusión de los protocolos de enrutamiento ara redes móviles específicamente para BCHP y CBRP en la herramienta de simulación NS2 en la versión 2.35.

   2.3. Se migro los protocolos de enrutamiento CBRP y BCHP de la versión NS2.34 a la versión NS2.3.

Además se menciona algunas conclusiones generales:

- Luego de la implementación, se comparó los algoritmos de enrutamiento BCHP y CBRP en la versión 2.35 a través de simulación con varios escenarios demostrando que BCHP tiene una mejor convergencia de red y disponibilidad con respecto a CBRP, debido a la inclusión de nodos JCR.

- En cuanto a la tasa de envío de todos los paquetes como también solo de los paquetes de aplicación, BCHP posee una mejor tasa de envío siendo así más efectivo ya que recibe mayor cantidad de paquetes generales y a su vez paquetes de aplicación.

- BCHP mantiene un mejor retardo de extremo a extremo con respecto a CBRP con un 1.10%, por ende el envío y recepción de información es más rápido gracias al Jefe de Clúster de Respaldo ya que éste toma las responsabilidades del Jefe de Clúster cuando éste no puede cumplirlas.

- BCHP es más eficiente al momento de la sobrecarga de protocolos con 1.16% menor con respecto a CBRP.
- Finalmente se hizo la comparación de la implementación de los protocolos en las dos versiones de la herramienta de simulación NS-2.34 y NS-2.35. En las dos versiones los protocolos BCHP y CBRP se comportan de forman similar.

REFERENCIAS

Capella, J. (n.d.). Introducción al simulador de redes NS-2. Retrieved from https://riunet.upv.es/bitstream/handle/10251/12735/Artículo docente NS-2.pdf?sequence=1

Carrión, B. O., & Delgado, L. A. (2015). *SIMULACIÓN Y ANÁLISIS DE REDES ESPORÁDICAS MÓVILES AD-HOC*. UNIVERSIDAD POLITÉCNICA SALESIANA. Retrieved from http://dspace.ups.edu.ec/bitstream/123456789/7777/1/UPS-CT004636.pdf

Chalmeta, J. (2009). *Estudio y análisis de prestaciones de redes móviles Ad Hoc mediante simulaciones NS-2 para validar modelos analíticos*. Retrieved from https://upcommons.upc.edu/bitstream/handle/2099.1/8374/PFC_Jordi_Chalmeta.pdf

Chatterjee, M., Das, S. K., & Turgut, D. (2002). WCA: A Weighted Clústering Algorithm for Mobile Ad Hoc Networks. *Clúster Computing*, *5*(2), 193–204. https://doi.org/10.1023/A:1013941929408

Criollo, J. W., & Ruilova, M. E. (2013). *Implementación de un protocolo de enrutamiento jerárquico proactivo en NS-2*. Universidad Técnica Particular de Loja. Retrieved from http://dspace.utpl.edu.ec/bitstream/123456789/7420/1/Implementación de un protocolo de enrutamiento jerárquico proactivo en ns2.pdf

Doumenc, H. (2008). *ESTUDIO COMPARATIVO DE PROTOCOLOS DE ENCAMINAMIENTO EN REDES VANET*. UNIVERSIDAD POLITÉCNICA DE MADRID. Retrieved from http://oa.upm.es/1444/1/PFC_HELENE_DOUMENEC.pdf

Enciso, L. (2013). *Un modelo de Encaminamiento en las Redes Móviles AD HOC con QoS*. Universidad Politécnica de Madrid. Retrieved from http://oa.upm.es/22726/1/LILIANA_ELVIRA_ENCISO_QUISPE.pdf

Fernandez, C. A., & Mena, S. A. (2004). *ANÁLISIS DE LOS PROTOCOLOS DE ENRUTAMIENTO Y ESTUDIO DE LOS PARÁMETROS Y ESQUEMAS PARA*

PROVEER CALIDAD DE SERVICIO (QoS) EN REDES MÓVILES AD HOC, SU OPERACIÓN Y APLICACIONES EN LOS SISTEMAS DE TELECOMUNICACIONES. ESCUELA POLITÉCNICA NACIONAL. Retrieved from http://bibdigital.epn.edu.ec/bitstream/15000/5397/1/T2335.pdf

Garzón, J. L. (2013). *ESTUDIO Y SIMULACIÓN DE LOS PROTOCOLOS DE ENRUTAMIENTO MÁS UTILIZADOS EN REDES VANETS JARRISON LEYTON GARZÓN*. UNIVERSIDAD CATÓLICA DE PEREIRA. Retrieved from http://ribuc.ucp.edu.co:8080/jspui/bitstream/handle/10785/1990/CDMIST74.pdf?sequence=1

Goitia, M. J. (2003a). *Protocolos de Enrutamiento*. Retrieved from http://exa.unne.edu.ar/informatica/SO/tfjuly.PDF

Goitia, M. J. (2003b). Protocolos de Enrutamiento Para la Capa de Red en Arquitecturas de Redes de Datos. Retrieved from http://exa.unne.edu.ar/informatica/SO/ProtocolosRed.PDF

Gomez, A. (n.d.). Protocolos de Enrutamiento Para la Capa de Red en Arquitecturas de Redes de Datos. Retrieved from http://www.monografias.com/trabajos-pdf/enrutamiento-redes-datos/enrutamiento-redes-datos.pdf

Herrera, J. M. (2004). NS2 -Network Simulator. Retrieved from https://www.inf.utfsm.cl/~jherrera/docs/mios/ns.pdf

Hollerung, T. D. (2004). The Clúster-Based Enrutamiento Protocol. Retrieved from http://s3.amazonaws.com/academia.edu.documents/34877064/the-clúster-based-enrutamiento-protocol.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1488813820&Signature=rv4TRCdY7%2BClsBY9PEcot71NZ8I%3D&response-content-disposition=inline%3B filename%3DThe_Cluste

Issariyakul, T., & Hossain, E. (2012). *Introduction to network simulator NS2*. *Introduction to Network Simulator NS2* (Vol. 9781461414). https://doi.org/10.1007/978-1-4614-1406-3

Jaramillo, E. (2004). Análisis general, características, aplicaciones, protocolos de enrutamiento y seguridad. Retrieved from https://secure.urkund.com/view/document/9715456-316597-105349/download

Jiang, M., Li, J., & Tay, Y. C. (1999). Clúster Based Enrutamiento Protocol• (CBRP), Functional Specification. Retrieved from http://www.ietf.org/ietf/1id-abstracts.txt.

Jiménez, E., Zerna, J., Chávez, P., & Basurto, J. (2011). Análisis de eficiencia de protocolos de enrutamiento en implementación de Red Inalámbrica Mesh en instalaciones de la Facultad de Ingeniería en Electricidad y Computación (FIEC) realizando llamadas de Voz sobre IP . Retrieved from http://www.rte.espol.edu.ec/index.php/tecnologica/article/view/82/45

Maldonado, V. (2012). *Comparación de protocolos de enrutamiento y modelos de movilidad para Redes Ad-Hoc Vehiculares usando mapas reales*. Retrieved from http://dspace.utpl.edu.ec/handle/123456789/4276

Martinez, O., & Palau, C. (2011). *INTRODUCCIÓN A LA PROGRAMACIÓN DE PROTOCOLOS DE COMUNICACIONES CON NETWORK SIMULATOR 2*. (Editorial Club Universitario, Ed.). Retrieved from http://www.editorial-club-universitario.es/pdf/498.pdf

Medina, A., & Macas, R. (2013). *Implementacion de un protocolo de enrutamiento Ad-Hoc en dispositivos móviles*. Universidad Técnica Particular de Loja. Retrieved from http://dspace.utpl.edu.ec/bitstream/123456789/8019/1/Macas Romero - Informatica.pdf

Mendoza, D. (2011). TABLAS Y PROTOCOLOS DE ENRUTAMIENTO | Telematika2. Retrieved March 31, 2017, from https://telematika2.wordpress.com/2011/02/15/tablas-y-protocolos-de-enrutamiento/

Modesto, L., & González, L. (n.d.). Protocolos de Enrutamiento. Retrieved from http://www.ieslosviveros.es/alumnos/asig8/carpeta812/PROTOCOLOS_DE_ENRUTAMIENTO.pdf

Mohseni, S., Hassan, R., Patel, A., & Razali, R. (2010). Comparative review study of reactive and proactive enrutamiento protocols in MANETs. In *4th IEEE International Conference on Digital Ecosystems and Technologies* (pp. 304–309). IEEE. https://doi.org/10.1109/DEST.2010.5610631

Morales, M., Calle, M. A., Tovar, J. D., & Cuéllar, J. C. (2013). Elección de una herramienta de simulación. Santiago de Cali: Facultad de Ingeniería. Retrieved from https://repository.icesi.edu.co/biblioteca_digital/bitstream/10906/68447/1/eleccion_herramienta_simulacion.pdf

Morató, D. (n.d.). Tipos de algoritmos de enrutamiento Enrutamiento Distance-Vector. Retrieved April 5, 2017, from https://www.tlm.unavarra.es/~daniel/docencia/ro_is/ro_is05_06/slides/Clase13-AlgoritmosyDV.pdf

Perkins, C. (1997). Mobile Ad Hoc Networking Terminology.

Radicelli, C. D. (2013). P2P En entornos móviles. Valencia-España. Retrieved from https://secure.urkund.com/view/document/14693519-835333-776418/download

Sevilla, J. (2011). *IMPLEMENTACION DE LOS PROTOCOLOS DE ENRUTAMIENTO OSPF Y RIP EN EL SIMULADOR NETWORK SIMULATOR 2*. UNIVERSIDAD POLITECNICA DE VALENCIA. Retrieved from https://riunet.upv.es/bitstream/handle/10251/9969/PFC_-_Jesus_Sevilla_Victoria.pdf;jsessionid=941BE2517A40F703A7A5121B06C4DBF2?sequence=1

Torres, R. (2011). *CONTRIBUCIÓN A LOS MODELOS DE GESTIÓN DE LAS REDES MÓVILES AD HOC*. Universidad Politécnica de Madrid. https://doi.org/10.1174/021435502753511268

Triviño, A., Casilari, E., & Ariza, A. (2006). IMPLEMENTACIÓN DE PROTOCOLOS EN EL NETWORK SIMULATOR (NS-2).

Ulloa, J. A. (2007). *Implementación de Protocolos de Enrutamiento mediante un Enrutador*

basado en Software de código abierto bajo Linux. Universidad Politécnica Nacional. Retrieved from http://bibdigital.epn.edu.ec/bitstream/15000/544/1/CD-1048.pdf

Umbrello, C. (2010). Umbrello UML Modeller. Retrieved August 5, 2017, from https://umbrello.kde.org/

Valverde, F. I. (2012). *IMPLEMENTACIÓN DEL PROTOCOLO HLMP EN ANDROID*. Universidad de Chile. Retrieved from http://repositorio.uchile.cl/bitstream/handle/2250/111920/cf-valverde_fc.pdf?sequence=1

Weingartner, E., Vom Lehn, H., & Wehrle, K. (2009). A Performance Comparison of Recent Network Simulators. https://doi.org/10.1109/ICC.2009.5198657

ANEXOS

ANEXO 1. Descarga e instalación de herramienta ns2

Para la descarga de la herramienta de simulación NS2 se lo puede realizar manualmente o por consola. En este caso se lo realizó por consola, primeramente ingresando a la carpeta donde se quiere descargar y con el siguiente comando:

sudo wget http://softlayer-sng.dl.sourceforge.net/project/nsnam/allinone/ns-allinone-2.35/ns-allinone-2.35.tar.gz

Se deberá descomprimir el archivo descargado con el siguiente comando:

tar -xvzf ns-allinone-2.35.tar.gz

Se deben instalar algunos paquetes que necesitan la herramienta NS2 para su posterior funcionamiento

- sudo apt-get install autoconf
- sudo apt-get install libc6-dev g++ gcc
- sudo apt-get install build-essential
- sudo apt-get install libx11-dev
- sudo apt-get install x-dev
- sudo apt-get install xorg-dev
- sudo apt-get install xgraph

Se debe modificar un archivo antes de comenzar con la instalación para posteriormente no tener errores durante la instalación. Se lo puede realizar manualmente o directamente desde el terminal. Este caso se lo realizó manualmente

- Ingresar a la carpeta ns-allinone-2.35
- Ir a la carpeta ns2.35/linkstate
- Buscar el archivo llamado ls.h
- En la línea 137 del archivo antes mencionado se encuentra la palabra **erase,** se debe agregar antes de la palabra **this->**. Tal como se muestra en la figura 49 y 50

```
        void eraseAll() {erase(baseMap::begin(), baseMap::end()); }
        T* findPtr(Key key) {
                iterator it = baseMap::find(key);
                return (it == baseMap::end()) ? (T *)NULL : &((*it).second);
        }
```

Figura 49: Modificar archivo ls.h

Fuente y Elaboración: La Autora

```
        void eraseAll() { this->erase(baseMap::begin(), baseMap::end()); }
        T* findPtr(Key key) {
                iterator it = baseMap::find(key);
                return (it == baseMap::end()) ? (T *)NULL : &((*it).second);
        }
};
```

Figura 50: Agregar palabra en archivo ls.h

Fuente y Elaboración: La Autora

Debido a que OTcl no funciona con la versión que viene instalado por defecto en la herramienta, se debe informar que se utiliza la versión GCC.

- Ingresamos al directorio de **ns-allinone-2.35**
- Seguidamente ingresamos a la carpeta **otcl-1.14**
- Buscamos el archivo **Makefile.in**
- En la línea 7 del archivo demos cambiar **@CC@** a **gcc-4.4.** Vease figura 51.

```
CC=             gcc-4.4
CFLAGS=         @CFLAGS@
RANLIB=         @RANLIB@
INSTALL=        @INSTALL@
```

Figura 51: Modificación en archivo Makefile.in

Fuente y Elaboración: La Autora

Se procese a instalar la herramienta NS2 desde el terminal. Se lo realiza mediante consola

- Se debe ingresar donde se encuentra la carpeta ns-allinone-2.35 con: cd/ns-allinone-2.35/
- sudo ./install

Se debe configurar las variables de entorno recién instaladas de la herramienta de simulación.

- En el terminal: sudo gedit ~/.bashrc
- Agregar al final del archivo, tener en cuenta la dirección de cada computador:

  cd# LD_LIBRARY_PATH

  OTCL_LIB=/home/priscita/ns-allinone-2.35/otcl-1.14

  NS2_LIB=/home/priscita/ns-allinone-2.35/lib

  X11_LIB=/usr/X11R6/lib

  USR_LOCAL_LIB=/usr/local/lib

  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:$X11_LIB:$USR_LOCAL_LIB

  # TCL_LIBRARY

  TCL_LIB=/home/priscita/ns-allinone-2.35/tcl8.5.10/library

  USR_LIB=/usr/lib

  export TCL_LIBRARY=$TCL_LIB:$USR_LIB

  # PATH

  XGRAPH=/home/priscita/ns-allinone-2.35/bin:/home/priscita/ns-allinone-2.35/tcl8.5.10/unix:/home/priscita/ns-allinone-2.35/tk8.5.10/unix

  #the above two lines beginning from xgraph and ending with unix should come on the same line

  NS=/home/priscita/ns-allinone-2.35/ns-2.35/

  NAM=/home/priscita/ns-allinone-2.35/nam-1.15/

  PATH=$PATH:$XGRAPH:$NS:$NAM
- Guardar y cerrar el archivo
- Recargar el archivo: source ~/.bashrc

Se debe validar si la herramienta ha sido correctamente instalada. La validación puede tomar algún tiempo. A la final de la validación se debe mostrar la siguiente información véase figura 52.

- Ingresamos a a carpeta ns-2.35: cd ns-2.35
- ./validate

Figura 52: Validación correcta

Fuente y Elaboración: La Autora

Si las variables de entorno han sido correctamente configuradas al ejecutar en el terminal **ns,** se muestra el signo %. Tal como se muestra en la figura 53

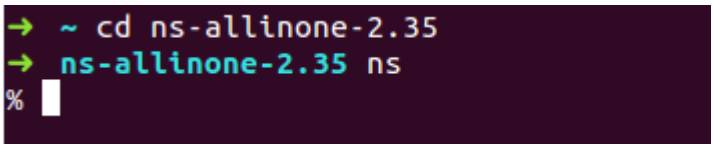


Figura 53: Herramienta instalada correctamente.

Fuente y Elaboración: La Autora

ANEXO 2. Inclusión de Protocolos en herramienta de simulación

- **ns-2.35/common/packet.h**

Se debe definir el método de acceso a la cabecera del paquete BCHP y CBRP, al final de todos los paquetes.

#define HDR_CBRP(p) (hdr_cbrp::access(p)) //added Pris

#define HDR_BCHP(p) (hdr_bchp::access(p)) // Added Pris

En el mismo archivo debemos incluir a los protocolos BCHP y CBRP en la lista de paquetes reconocidos por NS2 en la linea 201, antes de PT_NTYPE.

```
// Bchp packet
static const packet_t PT_BCHP = 73;
    // CBRP packet
static const packet_t PT_CBRP = 74;
```

```
    // Protoname Packet   inserted by Rommel 26-sep-2016
static const packet_t PT_PROTONAME = 75;
      // insert new packet types here
static packet_t       PT_NTYPE = 76; // This MUST be the LAST one
```

De debe definir a BCHP y CBRP como protocolos de enrutamiento en la línea 276 antes de return ENRUTAMIENTO;

```
static packetClass classify(packet_t type) {
            if (type == PT_DSR ||
               type == PT_MESSAGE ||
               type == PT_TORA ||
               type == PT_PUMA ||
               type == PT_AODV ||
               type == PT_MDART ||
               type == PT_BCHP ||
               type == PT_CBRP  ||
               type == PT_PROTONAME)
```

Definir los protocolos implementados para identificar los protocolos en las trazas. En la linea 428 se pueden definir los nombres de los protocolos antes de name_[PT_NTYPE]= "undefined";

```
name_[PT_BCHP]= "BCHP";  // added pris
name_[PT_CBRP]= "CBRP";//added pris
name_[PT_PROTONAME]="PROTONAME";   // added pris
name_[PT_NTYPE]= "undefined";
```

- **ns-2.35/trace/cmu-trace.h**

  Declaración de la función de registro de trazas en la línea 164 antes de static PacketTracer *pktTrc_;

```
void    format_cbrp(Packet *p, int offset); // added pris
void    format_bchp(Packet *p, int offset); // added pris
void    format_protoname(Packet *p, int offset; // added pris
// This holds all the tracers added at run-time
static PacketTracer *pktTrc_;
```

- **ns-2.35/trace/cmu-trace.cc**

  Inclusión de las librerías del encabezado del paquete BCHP y CBRP

```
#include <aodv/aodv_packet.h> //AODV
#include <cbrp/hdr_cbrp.h>        //CBRP  added pris
#include <bchp/hdr_bchp.h> // BCHP added pris
#include <protoname/protoname_pkt.h>  // addede pris
```

  En el mismo archivo se realiza la implementación de la función de registro de trazas en la línea 970, después del registro de trazas de protocolo aodv. (para ambos protocolos, cambiar nombre de protocolo)

```
void
CMUTrace::format_cbrp(Packet *p, int offset)
```

```
{
        hdr_cbrp *cbrph = HDR_CBRP(p); //modified by zhaoqi 2007.5.18
      hdr_ip *iph = HDR_IP(p);
        sprintf(pt_->buffer() + offset,
                "[%d #%d %d->%d] [%d #%d %d %d %d->%d] [%d %d]
[%d %d %d %d->%d]",
            cbrph->route_request(),
            cbrph->rtreq_seq(),
            iph->saddr(),
            cbrph->request_destination(),

            cbrph->route_reply(),
            cbrph->rtreq_seq(),
            cbrph->route_reply_len(),
            // the dest of the src route
            cbrph->reply_addrs()[cbrph->route_reply_len()-1].addr,
            cbrph->reply_addrs()[0].addr,
            iph->daddr(),

            cbrph->route_shortened(),
            cbrph->route_repaired(),

            cbrph->route_error(),
            cbrph->num_route_errors(),
            cbrph->down_links()[cbrph->num_route_errors() - 1].tell_addr,
            cbrph->down_links()[cbrph->num_route_errors() - 1].from_addr,
            cbrph->down_links()[cbrph->num_route_errors() - 1].to_addr);
}
```

En el mismo archivo se realiza la obtención del desplazamiento del encabezado BCHP y CBRP con respecto a cada uno de los paquetes. Línea 1501, después del encabezado de AOMDV

```
case PT_CBRP:           //added pris
                format_cbrp(p, offset);
                break;
case PT_BCHP:    // added pris
                format_bchp(p, offset);
        break;
```

- ns-2.35/tcl/lib/ns-lib.tcl

Inicializar el Agente de enrutamiento en el entorno OTcl. Línea 638, después del agente AODV

```
 CBRP {
          $self at 0.0 "$node start-cbrp"
        # set ragent [$self create-cbrp-agent $node]
          }
BCHP {
          $self at 0.0 "$node start-bchp"
        #set ragent [$self create-bchp-agent $node]
        }
```

En el mismo archivo, permitir la coexistencia de BCHP y otros protocolos de enrutamiento en el caso de que exista una estación base.

Linea 713

```
# Attach agent
        if {$enrutamientoAgent_ != "DSR" && $enrutamientoAgent_ != "CBRP"
&& $enrutamientoAgent_ != "BCHP"} {
                $node attach $ragent [Node set rtagent_port_]
        }
```

Línea 740

```
if { [info exist wiredEnrutamiento_] && $wiredEnrutamiento_ == "ON" } {
                if { $enrutamientoAgent_ != "DSR" && $enrutamientoAgent_ !=
"CBRP" && $enrutamientoAgent_ != "BCHP"  } {
                        $node mip-call $ragent
                }
        }
```

Definir el nodo como tipo BCHP. Línea 803

```
if {$enrutamientoAgent_ == "DSR"} {
        set nodeclass [$self set-dsr-nodetype]
} elseif { $enrutamientoAgent_ == "CBRP" } {        # added pris
           set nodeclass CBRPNode
        } elseif { $enrutamientoAgent_ == "BCHP" } {       # added pris
           set nodeclass BCHPNode
        } else {
                set nodeclass Node/MobileNode
        }
        return [eval new $nodeclass $args]
}
```

Línea 898

```
# BCHP patch
Simulator instproc create-bchp-agent { node } {
        set ragent [new Agent/BCHP [$node node-addr]]
        $self at 0.0 "$ragent start"
        $node set ragent_ $ragent
        return $ragent
}

# CBRP patch
Simulator instproc create-cbrp-agent { node } {
        set ragent [new Agent/CBRP [$node node-addr]]
        $self at 0.0 "$ragent start"
        $node set ragent_ $ragent
        return $ragent
}
```

- ns-2.35/tcl/lib/ns-mobilenode.tcl

```
# Special processing for CBRP
set cbrponly [string first "CBRP" [$agent info class]]
if {$cbrponly != -1 } {
        $agent if-queue [$self set ifq_(0)]   ;# ifq between LL and MAC
}
# Added rovitor 30/06/2016
# Special processing for BCHP
set bchponly [string first "BCHP" [$agent info class]]
if {$bchponly != -1 } {
        $agent if-queue [$self set ifq_(0)]   ;# ifq between LL and MAC
}
```

Implementación de las funciones en OTcl para instanciar y relacionar el agente y el protocolo de enrutamiento en el simulador. Al final del archivo.

```
# added by zhaoqi 2007.5.30
Class CBRPNode -superclass Node/MobileNode

CBRPNode instproc init {args} {
   $self instvar cbrp_agent_ dmux_ entry_point_ address_
   set ns_ [Simulator instance]
   eval $self next $args       ;# parent class constructor

   if {$dmux_ == "" } {
                # Use the default mash and shift
                set dmux_ [new Classifier/Port]
        }

   set cbrp_agent_ [new Agent/CBRP]
   $cbrp_agent_ ip-addr $address_
   $cbrp_agent_ set myaddr_ $address_
   $cbrp_agent_ node $self
   $self addr $address_
   puts "Creating node $address_";

   if { [Simulator set RouterTrace_] == "ON" } {
        # Recv Target
        set rcvT [$self mobility-trace Recv "RTR"]
      set namfp [$ns_ get-nam-traceall]
                if {  $namfp != "" } {
                        $rcvT namattach $namfp
                }
        $rcvT target $cbrp_agent_
        set entry_point_ $rcvT
   } else {
        # Recv Target
        set entry_point_ $cbrp_agent_
   }

   $self set ragent_ $cbrp_agent_

   $cbrp_agent_ target $dmux_
```

```
    # packets to the DSR port should be dropped, since we've
    # already handled them in the DSRAgent at the entry.
    set nullAgent_ [$ns_ set nullAgent_]
    $dmux_ install [Node set rtagent_port_] $nullAgent_
    # $dmux_ install $opt(rt_port) $cbrp_agent_

    # CBRPNodes don't use the IP addr classifier.  The DSRAgent should
    # be the entry point
    $self instvar classifier_
    set classifier_ "CBRPNode made illegal use of classifier_"
    return $self

}

CBRPNode instproc start-cbrp {} {
    $self instvar cbrp_agent_

    $cbrp_agent_ startcbrp
    # if {$opt(cc) == "on"}
    # checkcache $cbrp_agent_
}

CBRPNode instproc entry {} {
        $self instvar entry_point_
        return $entry_point_
}

CBRPNode instproc add-interface args {
# args are expected to be of the form
# $chan $prop $tracefd $opt(ll) $opt(mac)

    eval $self next $args

    $self instvar cbrp_agent_ ll_ mac_ ifq_

    set ns_ [Simulator instance]
    $cbrp_agent_ mac-addr [$mac_(0) id]

    if { [Simulator set RouterTrace_] == "ON" } {
        # Send Target
        set sndT [$self mobility-trace Send "RTR"]
       set namfp [$ns_ get-nam-traceall]
                if {$namfp != "" } {
                        $sndT namattach $namfp
                }
        $sndT target $ll_(0)
        $cbrp_agent_ add-ll $sndT $ifq_(0)
    } else {
        # Send Target
        $cbrp_agent_ add-ll $ll_(0) $ifq_(0)
    }

    # setup promiscuous tap into mac layer
    $cbrp_agent_ install-tap $mac_(0)
```

```
}

CBRPNode instproc reset args {
   $self instvar cbrp_agent_
   eval $self next $args

   $cbrp_agent_ reset
}

#

# added by rovitor 18/02/2011
Class BCHPNode -superclass Node/MobileNode

BCHPNode instproc init {args} {
   $self instvar bchp_agent_ dmux_ entry_point_ address_
   set ns_ [Simulator instance]
   eval $self next $args       ;# parent class constructor

   if {$dmux_ == "" } {
                # Use the default mash and shift
                set dmux_ [new Classifier/Port]
        }

   set bchp_agent_ [new Agent/BCHP]
   $bchp_agent_ ip-addr $address_
   $bchp_agent_ set myaddr_ $address_
   $bchp_agent_ node $self
   $self addr $address_
   puts "Creating node $address_";

   if { [Simulator set RouterTrace_] == "ON" } {
        # Recv Target
        set rcvT [$self mobility-trace Recv "RTR"]
      set namfp [$ns_ get-nam-traceall]
                if {  $namfp != "" } {
                        $rcvT namattach $namfp
                }
        $rcvT target $bchp_agent_
        set entry_point_ $rcvT
   } else {
        # Recv Target
        set entry_point_ $bchp_agent_
   }


   $self set ragent_ $bchp_agent_

   $bchp_agent_ target $dmux_

   # packets to the DSR port should be dropped, since we've
   # already handled them in the DSRAgent at the entry.
   set nullAgent_ [$ns_ set nullAgent_]
```

```
    $dmux_ install [Node set rtagent_port_] $nullAgent_
    # $dmux_ install $opt(rt_port) $bchp_agent_

    # BCHPNodes don't use the IP addr classifier.  The DSRAgent should
    # be the entry point
    $self instvar classifier_
    set classifier_ "BCHPNode made illegal use of classifier_"
    return $self

}

BCHPNode instproc start-bchp {} {
    $self instvar bchp_agent_

    $bchp_agent_ startbchp
    # if {$opt(cc) == "on"}
    # checkcache $bchp_agent_
}

BCHPNode instproc entry {} {
        $self instvar entry_point_
        return $entry_point_
}


BCHPNode instproc add-interface args {
# args are expected to be of the form
# $chan $prop $tracefd $opt(ll) $opt(mac)

    eval $self next $args

    $self instvar bchp_agent_ ll_ mac_ ifq_

    set ns_ [Simulator instance]
    $bchp_agent_ mac-addr [$mac_(0) id]

    if { [Simulator set RouterTrace_] == "ON" } {
        # Send Target
        set sndT [$self mobility-trace Send "RTR"]
       set namfp [$ns_ get-nam-traceall]
                if {$namfp != "" } {
                        $sndT namattach $namfp
                }
        $sndT target $ll_(0)
        $bchp_agent_ add-ll $sndT $ifq_(0)
    } else {
        # Send Target
        $bchp_agent_ add-ll $ll_(0) $ifq_(0)
    }

    # setup promiscuous tap into mac layer
    $bchp_agent_ install-tap $mac_(0)

}
```

```
BCHPNode instproc reset args {
   $self instvar bchp_agent_
   eval $self next $args

   $bchp_agent_ reset
}
```

- ns-2.35/Makefile

**-I./protoname **
**-I./bchp **
-I./cbrp

Incluir el código fuente para que se compile con el simulador.

**bchp/bchpagent.o bchp/bchpntable.o bchp/hdr_bchp.o **
**protoname/Protoname.o protoname/protoname_rtable.o **
**cbrp/cbrpagent.o cbrp/ntable.o cbrp/hdr_cbrp.o **

ANEXO 3. Configuración de la topología

- **setdest**

Esta herramienta ayuda a realizar la movilidad de los nodos, además de desplazar de un lugar a otro dependiendo de una velocidad máxima y de una pausa promedio entre los movimientos. El siguiente comando se lo debe realizar dentro de: /home/priscita/ns-allinone-2.35/ns-2.35/indep-utils/cmu-scen-gen/setdest

./setdest -n 80 -p 2.0 -M 10.0 -t 150 -x 1000 -y 1000 > scen-80-test

Donde:

-n: cantidad de nodos que se va a utilizar

-p: pausa promedio entre los movimientos

-M: velocidad máxima de movimiento de nodos

-t: Tiempo de simulación

-x: límite de la topología en eje x

-y: límite de la topología en eje x´

scen-80-test: nombre del archivo(puede variar)

- **cbrgen.tcl**

Esta herramienta ayuda a la creación de conexiones entre los nodos, se puede especificar el número máximo de conexiones. Además se especifica qué tipo de conexiones se realizarán. Se lo debe realizar en la carpeta: / ns-allinone-2.35/ns-2.35/indep-utils/cmu-scen-gen

ns cbrgen.tcl -type cbr -nn 80 -seed 1.0 -mc 40 -rate 4.0 > cbr-80-test

Donde:

-type: tipo de conexion CBR o TCP

-nn: cantidad de nodos

-seed: valor de semilla

-mc: máximo de conexionoes

-rate: tasa

cbr-80-test: nombre del archivo(puede variar)

ANEXO 4. Archivo OTcl usado en la simulación

Funciones principales de archivo de simulación

A continuación se escriben las líneas de código más importantes en el archivo de situación.

El siguiente código se puede utilizar para la creación de escenarios con diferentes nodos. La primera parte del código se especifica las opciones, donde se definen los valores correspondientes a cada una de las variables como por ejemplo: los nodos a utilizar, el tipo de canal, modelo de antena, tiempo de simulación, tipo de protocolo de enrutamiento (BCHP y CBRP), entre otros.

```
set val(chan)           Channel/WirelessChannel ;#Channel Type
set val(prop)           Propagation/TwoRayGround ;# radio-propagation model
set val(netif)          Phy/WirelessPhy ;# network interface type
set val(mac)            Mac/802_11 ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue ;# interface queue type
set val(ll)             LL ;# link layer type
set val(ant)            Antenna/OmniAntenna ;# antenna model
set val(ifqlen)         100 ;# max packet in ifq
```

```
set val(nn)             20 ;# number of mobilenodes
set val(rp)             BCHP ;# enrutamiento protocol
set val(x)              500
set val(y)              500
set val(conexiones)     "scen-tcp" ;

set val(scenario) "scen25total" ;
set val(stop) 100 ;
# Opciones de energía
set val(engmodel) EnergyModel
set val(txPower) 3 ;# transmitting power in mW
set val(rxPower) 2 ;# recving power in mW
set val(sensePower) 1; ;# sensing power in mW
set val(idlePower) 1 ;# idle power in mW
set val(initeng) 200 ;# Initial energy in Joules
=============================================================
# Programa Principal
# Initialize the SharedMedia interface with parameters to make
# it work like the 2.4GMHz Lucent Orinoco WaveLAN DSSS radio interface (11
Mb/s and 170 m)
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 5.011872e-12
Phy/WirelessPhy set RXThresh_ 1.02054e-10
Phy/WirelessPhy set Rb_ 11*1e6
Phy/WirelessPhy set Pt_ 0.031622777
Phy/WirelessPhy set freq_ 2.472e9
Phy/WirelessPhy set L_ 1.0
# Ajuste del ancho de banda en 11 Mbps
Mac/802_11 set dataRate_ 11Mb
Mac/802_11 set basicRate 11Mb
# Inicialización de variables globales
set ns_ [new Simulator]
set tracefd [open bchpe25tcp.tr w]
$ns_ trace-all $tracefd
```

```
$ns_ use-newtrace
set namtrace [open bchpe25tcp.nam w]
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
# Configurando la topografía
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
# Create God
create-god $val(nn)}
122
# New API to config node:
# 1. Create channel (or multiple-channels);
# 2. Specify channel in node-config (instead of channelType);
# 3. Create nodes for simulations.
# Create channel #1 and #2
set chan_1_ [new $val(chan)]
set chan_2_ [new $val(chan)]
# configure node, please note the change below.
$ns_ node-config -adhocEnrutamiento $val(rp) \ -llType $val(ll) \ -macType $val(mac) \ -ifqType $val(ifq) \ -ifqLen $val(ifqlen) \ -antType $val(ant) \ -propType $val(prop) \ -phyType $val(netif) \ -topoInstance $topo \ -agentTrace ON \ -routerTrace ON \ -macTrace OFF \ -movementTrace OFF \ -channel $chan_1_ \ -energyModel $val(engmodel)\ -rxPower $val(rxPower) \ -txPower $val(txPower) \ -sensePower $val(sensePower) \ -idlePower $val(idlePower) \ -initialEnergy $val(initeng)
set node_(0) [$ns_ node]
$node_(0) color red
$ns_ at 0.0 "$node_(0) color red"
$node_(0) label â€œSAâ€
for {set i 1} {$i < $val(nn)} {incr i} {
set node_($i) [$ns_ node]
$node_($i) random-motion 0;# enable random motion
}
puts "Loading scenario file..."
source $val(scenario)
# Configurando el fllujo de tráfico entre nodos
# TCP connections between node_(0) and node_(1)
```

```
123
puts "Loading connection pattern..."
source $val(conexiones)
# Dice a los nodos el final de la simulación
for {set i 0} {$i < $val(nn) } {incr i} {
$ns_ at $val(stop) "$node_($i) reset";
}
$ns_ at $val(stop) "stop"
$ns_ at $val(stop).01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
global ns_ tracefd
$ns_ flush-trace
close $tracefd
}
puts "Starting Simulation..."
$ns_ run
```

ANEXO 5. Archivos awk

Cada uno de los siguientes archivos ayuda a obtener resultados de rendimiento, pérdida de paquetes, jitter, paquetes enviados/paquetes recibidos. Se describe cada una de las partes de cada código.

Rendimiento

```
BEGIN {
# Las instrucciones contenidas dentro de BEGIN sólo se ejecutan una vez
# Se abre el fichero de salida
salida2="./Graf_troughput.txt"
# Se inicializan las variables globales
mayor_id=-1
bytes_env_0=0
```

```
rec_1=0
env_0=0
drop=0
bytes_rec_11=0
 }
{
# Las instrucciones contenidas dentro de este apartado principal se ejecutan una vez
# para cada fila del fichero
# Se asignan variables a las columnas o campos a utilizar del fichero de trazas
accion=$1
nodo_actual=$9
bytes_paquete=$37
id_paquete=$41
nodo_destino=$7
prot_mes=$35
trace_level=$19
if(accion!="SFESTs" && accion!="SFs" && trace_level=="AGT"){
# Se analizan los paquetes enviados
if ( accion == "s"){
    if (id_paquete > mayor_id){ # Se miran los paquetes no analizados previamente
        mayor_id=id_paquete
#       if ( prot_mes == "cbr"){ # Sólo interesan los paquetes CBR
            if ( nodo_actual == 0){
                # Se comprueba si el paquete lo envía el nodo fuente
                    bytes_env_0 = bytes_env_0 + bytes_paquete
                    # Se suman los bytes CBR enviados por el nodo 0 o fuente
```

```
            env_0++
              # Se incrementa el número de paquetes CBR enviados por el nodo 0
     }
   }
}
# Se analizan los paquetes recibidos
if ( accion == "r"){
   if (nodo_actual == nodo_destino){
      if (nodo_actual == 0){
         bytes_rec_1 = bytes_rec_1 + bytes_paquete - 20
         # Se suman los bytes CBR recibidos por el nodo 0 o destino, se descuenta
         # la cabecera IP de 20 bytes
         rec_1++
         # Se incrementa el número de paquetes CBR recibidos por el nodo 1
       }
    }
}
if ( accion == "d"){
   if (nodo_actual == nodo_destino){
      if (nodo_actual == 0){
         # Se suman los bytes CBR recibidos por el nodo 0 o destino, se descuenta
         # la cabecera IP de 20 bytes
         drop++
         # Se incrementa el número de paquetes CBR recibidos por el nodo 1
       }
    }
```

```
}

}}


END{

# Las instrucciones contenidas dentro de END sólo se ejecutan una vez

# Se guardan los resultados obtenidos en el fichero de salida

printf(": ") >> salida2

printf("Paquetes recibidos  %i ", rec_1) > salida2

printf("\nPaquetes enviados  %i ", env_0) > salida2

printf("\nPaquetes borrados  %i ", drop) > salida2

#printf("Throughput1 %i ", rec_1/env_0) > salida2

printf("\nBytes recibidos  %f ", bytes_rec_1) > salida2

printf("\nBytes enviados  %i ", bytes_env_0) > salida2

#printf("Throughput2  %f \n", bytes_rec_1/bytes_env_0) > salida2

#printf("Throughput2  %f \n", bytes_rec_1/bytes_env_0) > salida2

# Aunque se calculan dos throughputs, uno a partir de los paquetes y otro a partir de

# los bytes (descontando en este último caso los bytes de las cabeceras IP), en este

# proyecto todos los datos obtenidos del throughput usados son a partir del número de

# paquetes. El segundo throughput no se ha usado para extraer los resultados pero se

# mantiene aquí como dato extra.

# Se cierra el fichero de salida

close(salida2)

}
```

Paquetes perdidos

```
# Para ejecutar awk -f thisfile trazafile listadodearchivos
```

```
# ARGCV[1] trazafile ARGV2[2] listadodearchivos
# Para obtener el listado de archivos
# find . -type f | grep \.tr$ > listadodearchivos
BEGIN {
# Las instrucciones contenidas dentro de BEGIN sólo se ejecutan una vez
# Se abre el fichero de salida
salida2="./Graf_dropped.txt"
# Se inicializan las variables globales
total=0
end=0
col=0
dup=0
err=0
ret=0
sta=0
bsy=0
nrte=0
loop=0
ttl=0
tout=0
cbk=0
ifq=0
arp=0
out=0
file=ARGV[1]
 }
```

```
{
# Las instrucciones contenidas dentro de este apartado principal se ejecutan una vez
# para cada fila del fichero
# Se asignan variables a las columnas o campos a utilizar del fichero de trazas
accion=$1
nodo_actual=$9
bytes_paquete=$37
id_paquete=$41
nodo_destino=$7
prot_mes=$35
trace_level=$19
borrado=$21
if ( accion == "d"){
        total++
   if (borrado == "END") {
            end++
   }
   if (borrado == "COL") {
            col++
   }
   if (borrado == "DUP") {
            dup++
   }
   if (borrado == "ERR") {
            err++
   }
   if (borrado == "RET") {
            ret++
   }
   if (borrado == "STA") {
```

```
                sta++
        }
        if (borrado == "BSY") {
                bsy++
        }
        if (borrado == "NRTE") {
                nrte++
        }
        if (borrado == "LOOP") {
                loop++
        }
        if (borrado == "TTL") {
                ttl++
        }
        if (borrado == "TOUT") {
                tout++
        }
        if (borrado == "CBK") {
                cbk++
        }
        if (borrado == "IFQ") {
                ifq++
        }
        if (borrado == "ARP") {
                arp++
        }
        if (borrado == "OUT") {
                out++
        }
#    if (nodo_actual == nodo_destino){
#        if (nodo_actual == 0){
            # Se suman los bytes CBR recibidos por el nodo 0 o destino, se descuenta
```

```
            # la cabecera IP de 20 bytes
            # Se incrementa el número de paquetes CBR recibidos por el nodo 1
#         }}
}
}

END{
# Las instrucciones contenidas dentro de END sólo se ejecutan una vez
# Se guardan los resultados obtenidos en el fichero de salida
printf("%s|%i|%i|%i|%i|%i|%i|%i|%i|%i|%i|%i|%i|%i|%i|%i \n",
file,total,end,col,dup,err,ret,sta,bsy,nrte,loop,ttl,tout,cbk,ifq,arp,out) >> salida2
# Aunque se calculan dos throughputs, uno a partir de los paquetes y otro a partir de
# los bytes (descontando en este último caso los bytes de las cabeceras IP), en este
# proyecto todos los datos obtenidos del throughput usados son a partir del número de
# paquetes. El segundo throughput no se ha usado para extraer los resultados pero se
# mantiene aquí como dato extra.
# Se cierra el fichero de salida

close(salida2)
}
```

Jitter

```
#This program is used to calculate the jitters for CBR
# jitter =((recvtime(j)-sendtime(j))-(recvtime(i)-sendtime(i)))/(j-i),  j > i

BEGIN {
```

```
# Initialization
     highest_packet_id = 0;
     salidajitter="./Graf_jitter";
     salidae2edelay="./Graf_e2edelay";
     flowfile="./flowfile";
     split(ARGV[1],a,"/");
     tmp=a[4];
   split(tmp,a,".");
     printf("archivo %s; a[4] %s; b[1] %s", ARGV[1], tmp, a[1])
    graphjitter=sprintf("%s-%s.txt",salidajitter,a[1]);
   graphe2edelay=sprintf("%s-%s.txt",salidae2edelay,a[1]);
#     printf("nuevo archivo: %s, %s\n", salidajitter, salidae2edelay)
     allsend=0;
     allreceive=0;
     allAGT=0;
     allRTR=0;
     allrecAGT=0;
     allrecRTR=0;
}
{
   action = $1;
   time = $3;
   from = $5;
   to = $7;
   type = $35;
   pktsize = $37;
```

```
  flow_id = $39;
  level_trace = $19;
  src = $31;
  dst = $33;
  seq_no = $47;
  packet_id = $41;

# printf("flow_id id: %i, packet_id, %i \n %s \n", flow_id,packet_id, $0);
  if (flow_id > 0) {
 #     printf("flow_id id: %i, packet_id, %i \n %s \n", flow_id,packet_id, $0);
  }
    if (action == "s") {
       allsend++;
#       By protocolo Overhead
       if (level_trace=="AGT") {
          allAGT++;
        }
       if (level_trace=="RTR") {
           allRTR++
        }
  }
  if (action == "r") {
      allreceive++
       if (level_trace=="AGT") {
          allrecAGT++;
        }
```

```
        if (level_trace=="RTR") {
            allrecRTR++
         }
   }
   if ( packet_id > highest_packet_id ) {
          highest_packet_id = packet_id;
       }
#Record the transmission time
   if ( start_time[packet_id] == 0 )  {
          # Record the sequence number
          pkt_seqno[packet_id] = seq_no;
          start_time[packet_id] = time;
   }
#Record the receiving time for CBR (flow_id=2)
   if ( flow_id == 0 && action != "d" ) {
      if ( action == "r" ) {
            end_time[packet_id] = time;
      }
    } else {
      end_time[packet_id] = -1;
   }
}
END {
        last_seqno = 0;
        last_delay = 0;
        seqno_diff = 0;
```

```
        jitter_average = 0;
      e2edelay_average = 0;


  for ( packet_id = 0; packet_id <= highest_packet_id; packet_id++ ) {
    start = start_time[packet_id];
    end = end_time[packet_id];
    packet_duration = end - start;

#    printf("flow_id id: %i, packet_id, %i \n %s \n", flow_id,packet_id, $0);
    if ( start < end ) {
          e2edelay_average = e2edelay_average + packet_duration;
          seqno_diff = pkt_seqno[packet_id] - last_seqno;
          delay_diff = packet_duration - last_delay;
          if (seqno_diff <= 0 || delay_diff <= 0) {
          if (seqno_diff == 0 ) {
                jitter =0;
          } else {
               jitter = delay_diff/seqno_diff;
                jitter_average = jitter_average + jitter;
          }
  #  Print jitter graf for each trace file
          printf("%i %f %f\n", packet_id, start, abs(jitter)) >> graphjitter;
  #  Print End to end dely for each trace file
          printf("%i %f %f\n",  packet_id, start, packet_duration) >> graphe2edelay;
          last_seqno = pkt_seqno[packet_id];
```

```
            last_delay = packet_duration;
      }
   }
#  printf("flow_id id: %i, packet_id, %i \n %s \n", flow_id,packet_id, $0);
#    printf("File|Delay Average|Jitter
Average|AllSend|SendAGT|SendRTR|AllReceive|recAGT|recRTR");
}
      if (packet_id == 0 ) {
         temp = e2edelay_average/1
         temp1 =  jitter_average/1
      } else {
          temp = e2edelay_average/packet_id
          temp1 =  jitter_average/packet_id
      }

        printf("%s|%f|%f|%d|%d|%d|%d|%d|%d\n", ARGV[1], temp , temp1,allsend,allAGT,
allRTR, allreceive,allrecAGT, allrecRTR) >> salidae2edelay;
   close(salidae2edelay);
   close(graphjitter);
   close(graphe2edelay);
}
function abs(value)
{
  return (value<0?-value:value);
}
```

Paquetes enviados/recibidos (rateAGTsendingpackets)

```
BEGIN {
# Este script obtiene la tasa de envio de paquetes solo para AGT, no toma en cuenta
# las trazas de enrutamiento (RTR) ni las trazas (MAC)
#  Se divide los paquetes enviados para los paquetes recibidos
# Las instrucciones contenidas dentro de BEGIN sólo se ejecutan una vez
# Se abre el fichero de salida
    salida2="./AGTratesending"
    split(ARGV[1],a,"/");
    tmp=a[4];
    split(tmp,a,".");
    printf("archivo %s; a[4] %s; b[1] %s", ARGV[1], tmp, a[1])
    Nombre=a[1];

# Se inicializan las variables globales
mayor_id=-1
bytes_env_0=0
rec_1=0
env_0=0
drop=0
bytes_rec_11=0
packets_overhead=0
bytes_overhead_noiphead=0
bytes_overhead_full=0
 }
{
```

```
# Las instrucciones contenidas dentro de este apartado principal se ejecutan una vez
# para cada fila del fichero
# Se asignan variables a las columnas o campos a utilizar del fichero de trazas
accion=$1
nodo_actual=$9
bytes_paquete=$37
id_paquete=$41
nodo_destino=$7
prot_mes=$35
trace_level=$19
if(trace_level=="AGT"){
  # Se analizan los paquetes enviados
  if ( accion == "s"){
      if (id_paquete > mayor_id){ # Se miran los paquetes no analizados previamente
          mayor_id=id_paquete
                  # Se comprueba si el paquete lo envía el nodo fuente
                    bytes_env_0 = bytes_env_0 + bytes_paquete
                  # Se suman los bytes CBR enviados por el nodo 0 o fuente
                 env_0++
                # Se incrementa el número de paquetes CBR enviados por el
nodo 0
      }
   } else if ( accion == "r") {        # Se analizan los paquetes recibidos
          bytes_rec_1 = bytes_rec_1 + bytes_paquete - 20;
          # Se suman los bytes CBR recibidos por el destino, se descuenta
          # la cabecera IP de 20 bytes
```

```
                rec_1++
                # Se incrementa el número de paquetes CBR recibidos por los nodos
            }
        }
        # Se determina cual es la cantidad de paquetes a nivel de capa de red se han usado en la simulacion
        if ((accion == "s" || accion == "f") && (trace_level == "RTR"))
        {
            packets_overhead = packets_overhead + 1;
            bytes_overhead_full = bytes_overhead_full + bytes_paquete;
            bytes_overhead_noiphead = bytes_overhead_noiphead + bytes_paquete  - 20;
        }
}

END{
# Las instrucciones contenidas dentro de END sólo se ejecutan una vez
# Se guardan los resultados obtenidos en el fichero de salida
# Imprime nombre del fichero, paquetesrecibibos, enviados, recibidos/enviados, bytesenviados, bytesrecibidos,bytesrecibidos/bytesenviados, paqutessoverhead, bytes_overhead full, bytes overhead sin cabecera IP
#printf("Traza|Paq rec|Paq env| Pac rec sobre Pac env|bytes rec|bytes env| bytes rec sobre bytes env|overhead full|overhead sin cabecera IP")
printf("%s|%i|%i|%f|%i|%i|%f|%i|%i|%i\n ", Nombre, rec_1, env_0, rec_1/env_0, bytes_rec_1, bytes_env_0,
bytes_rec_1/bytes_env_0,packets_overhead,bytes_overhead_full,bytes_overhead_noiphead ) >> salida2
close(salida2)
}
```