VNIVERSIDAD NACIONAL
AVTONOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"INUNDACIÓN TOLERANTE A RETARDOS EN REDES AD HOC"

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

ISRAEL CENTENO RAMÍREZ

TUTOR:
DR. JAVIER GÓMEZ CASTELLANOS, FACULTAD DE INGENIERÍA

MÉXICO, D. F. ENERO 2015

AGRADECIMIENTOS

Agradezco al Consejo de Ciencia y Tecnología (CONACYT) y al Posgrado en Ciencia e Ingeniería de la Computación de la Universidad Nacional Autónoma de México (UNAM) por todo el apoyo económico, técnico y científico para el desarrollo de: “Inundación tolerante a retardos en redes ad hoc.”

Además quiero extender este agradecimiento para PAPPIT IN114813 de DAGAPA y CONACYT 105117 por su apoyo durante ej desarrollo de este trabajo de tesis.

ÍNDICE GENERAL

CAPÍTULO I ANTEPROYECTO DE INVESTIGACIÓN 5
1.1 INTRODUCCIÓN 5
1.2 JUSTIFICACIÓN Y DELIMITACIÓN 9
1.3 OBJETIVO 10
1.4 HIPÓTESIS 10
1.5 MÉTODO DE INVESTIGACIÓN 10
1.6 APORTES DE ESTE TRABAJO 11
CAPÍTULO II ESTADO DEL ARTE 12
2.1 TRABAJO RELACIONADO 12
2.2 INUNDACIÓN CIEGA 12
2.3 AODV 13
2.4 DSR 14
2.5 TORA 15
2.6 MPR 16
2.7 NARD 17
2.8 FRESH 19
2.9 TABLA DE COMPARACIÓN 22
CAPÍTULO III ALGORITMOS PARA FASE DE DESCUBRIMIENTO DE RUTAS 23
3.1 ALGORITMO A 23
3.2 ALGORITMO A* 27
3.3 ALGORITMO B 30
3.4 ALGORITMO B* 33
CAPÍTULO IV EVALUACIÓN DE DESEMPEÑO DE LOS ALGORITMOS 36
4.1 CAMBIOS AL SIMULADOR NS2 36
4.2 ESCENARIO DE LA RED 39
4.3 ALGORITMO A 39
4.4 ALGORITMO A* 41
4.5 ALGORITMO B 43
4.6 ALGORITMO B* 45
4.7 COMPARACIÓN DE NARD, FRESH Y ALGORITMO A* 49

CAPÍTULO V CONCLUSIONES 51
BIBLIOGRAFÍA Y REFERENCIAS 53

CAPÍTULO I ANTEPROYECTO DE INVESTIGACIÓN

En este capítulo se explicará en qué consiste este trabajo de tesis. Se dará una introducción que nos permitirá entender la necesidad de llevar a cabo este trabajo de investigación y posteriormente se describirá de manera detallada el planteamiento del problema, el objetivo a cumplir, la hipótesis, la delimitación, el trabajo relacionado, etc.

1.1 INTRODUCCIÓN

Un hecho muy importante en la vida de todas las personas es la necesidad de comunicarse. Durante épocas recientes, la creación de distintas tecnologías ha permitido el surgimiento de inventos como la radio, la televisión, redes mundiales de telefonía, la computación, internet, satélites de comunicaciones y el fenómeno que vivimos hoy en día con los dispositivos inalámbricos. El uso de dispositivos inalámbricos se ha expandido a lo ancho del mundo lo cual ha permitido que nuevas aplicaciones emerjan, dando como resultado que cada vez sea más común el uso de aplicaciones para enviar y recibir una gran cantidad de información.

Si bien originalmente todas estas tecnologías operaron de forma independiente, el progreso tecnológico ha acelerado la necesidad de que converjan. El resultado de esta convivencia es lo que se conoce como las redes de comunicaciones. Una red, de acuerdo a su definición más general, es un conjunto de elementos organizados que interactúan para lograr un determinado fin. Refiriéndonos en particular a las redes inalámbricas, éstas son aquellas que se componen de un conjunto de nodos que no utilizan una conexión alámbrica para comunicarse y compartir información.

Una subclasificación de las redes inalámbricas, a la que nos referiremos a lo largo de este trabajo, son las denominadas redes inalámbricas *ad hoc*, que tienen una particularidad muy importante; este tipo de redes se caracterizan por ser una colección de nodos inalámbricos con una administración descentralizada y que tienen capacidad de autoconfiguración.

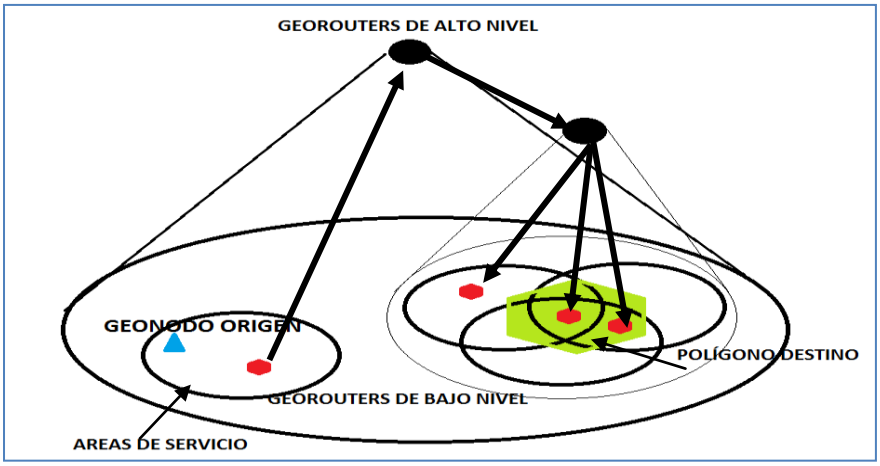
El término *ad hoc* viene del latín y significa literalmente “para esto”, se refiere a una solución específicamente elaborada para un problema o fin preciso y, por lo tanto, no generalizable, mientras que en comunicaciones el propósito de *ad hoc* es proporcionar flexi ili dad y autonom a aprovechando los principios de autoorganización.

A este tipo de redes inalámbricas también se les conoce como redes *MANET* **[15]** (redes *ad hoc* móviles) donde los nodos tienen la libertad de moverse y debido a que el rango de transmisión es limitado, en las redes *ad hoc*, los propios nodos deben participar en el proceso de encaminamiento, entregando paquetes a sus vecinos. Este tipo de encaminamiento es conocido de saltos múltiples (*multi-hop routing*). Podemos ubicar los orígenes de las redes *ad hoc* en los años setenta con proyectos como *PRnet* (*packet radio network*) **[14]** que era una red de saltos múltiples, donde los nodos coopera an en la retransmisión del tr fico para llegar a estaciones distantes que de otra manera estar an fuera de su alcance.

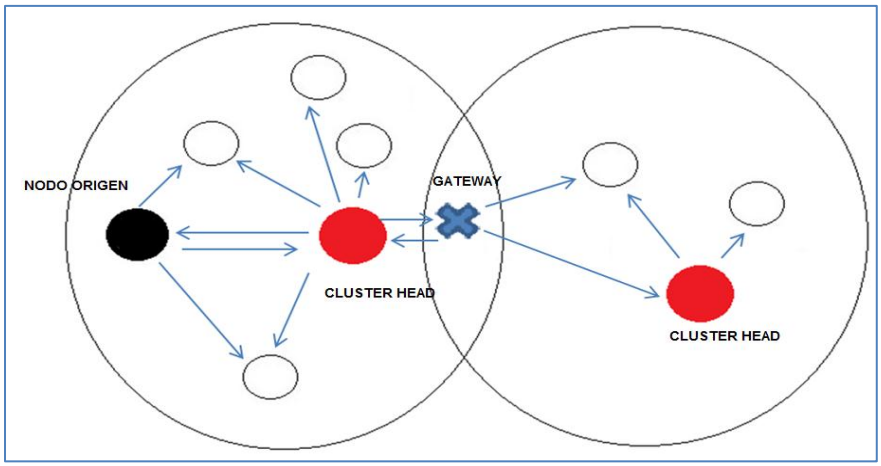
Debido a que las redes *ad hoc* presentan cam ios frecuentes en su topolog a de ido a su movilidad, estas caracter sticas impiden la utilización de protocolos de encaminamiento tradicionales. Esta situación obliga a buscar otro tipo de algoritmos que ofrezcan soluciones para pro lemas tales como topolog a din mica y ancho de banda de la red limitado.

A diferencia de las redes estáticas cableadas, donde los algoritmos de encaminamiento más usados usualmente están basados en el estado de los enlaces (*link-state routing protocol*) **[10][11]** o algoritmos basados en vectores de distancia (*distance-vector routing protocol*) **[10][11]**, en el caso de las redes *ad hoc* se pueden clasificar en tres grupos principales: planos (*flat*), jerárquicos (*hierarchical*) y de posición geográficamente asistidos (*geographic position assisted*). Los protocolos planos se refieren a aquellos en los que todos los nodos tienen el mismo comportamiento dentro de la red. En los protocolos jerárquicos algunos nodos tienen una responsabilidad diferente en el funcionamiento del protocolo de encaminamiento. Por último están los protocolos de posición geográfica asistida en donde los nodos pueden ser ayudados por algún dispositivo especial, como un dispositivo *GPS*) **[10][11]**.

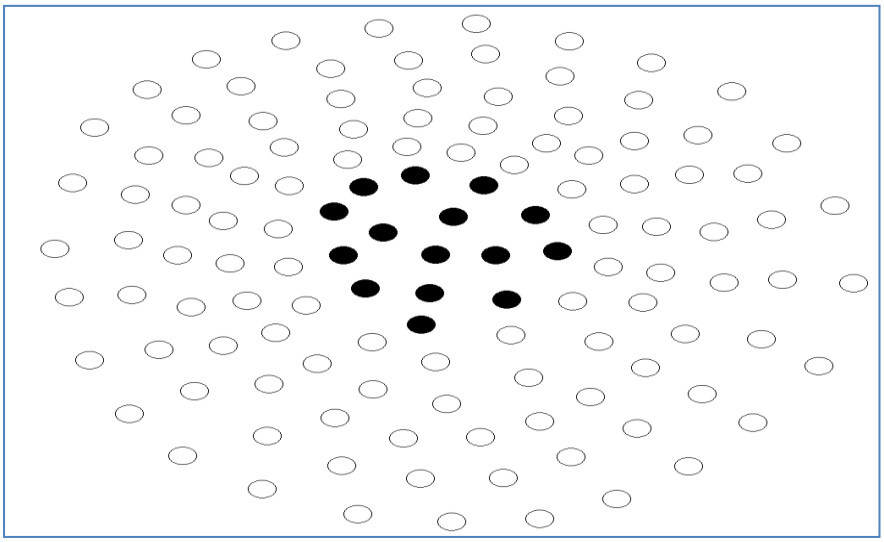
En la *Figura 1.0* se muestra un ejemplo de estos algoritmos. En primer lugar, tenemos el algoritmo de posición geográficamente asistido, donde se hace uso de *georouters* para encontrar al nodo destino. La segunda imagen muestra al algoritmo jerárquico donde se visualiza que en la red existen algunos nodos que tienen un papel diferente, algunos de ellos sí pueden reenviar paquetes y otros se limitan a recibirlos. Por último está una imagen que representa los algoritmos planos en donde un mensaje de inundación comienza a diseminarse, en este caso todos los nodos tienen el mismo rol en la red.



ALGORIMO DE POSICIÓN GEOGRÁFICAMENTE ASISTIDO



ALGORIMO JERÁRQUICO



ALGORIMO PLANO

FIGURA 1.0 DIAGRAMAS DE ALGORITMOS PARA REDES AD HOC

En este trabajo nos enfocamos en particular a los protocolos planos. De acuerdo al tipo de encaminamiento estos protocolos se dividen en tres tipos; proactivos, reactivos e híbridos **[12]**. Los protocolos proactivos procuran mantener la información de encaminamiento consistente y actualizada para todos los demás nodos de la red. Por lo tanto, si un nodo necesita una ruta la obtiene inmediatamente, ya que esta información se mantiene en tablas. También se les conoce como *table driven* **[12]**.

Los protocolos de encaminamiento reactivos descubren rutas únicamente cuando se necesitan. El proceso de descubrimiento de ruta es iniciado por el nodo origen y, si se encuentra una ruta hacia el destino deseado, ésta se mantiene hasta que el destino deje de ser accesible. A diferencia de los protocolos proactivos aquí no se genera una gran cantidad de tráfico para mantener las tablas actualizadas; pero aumenta el tiempo de retardo que le toma al nodo origen encontrar una ruta al nodo destino (tiempo de extremo a extremo (*end-to-end delay*)). La fase de descubrimiento de rutas en los protocolos de encaminamiento reactivos para redes *ad hoc* es un tema de interés primordial ya que, aunque hay muchas propuestas para realizarla. La técnica de búsqueda más empleada es la inundación ciega. Esta técnica consiste en la diseminación de paquetes de control en toda la red. En este protocolo un nodo denominado origen envía un mensaje con el objetivo de encontrar algún nodo en particular, el cual es recibido por todos sus vecinos que se encuentren en su rango de transmisión. Al recibir el mensaje, cada nodo vecino lo retransmite sólo una vez a su propio vecindario. Dicho proceso se repite hasta encontrar el nodo destino. El gran número de paquetes que se generan al utilizar este tipo de encaminamiento tiene el inconveniente de que puede incrementar significativamente el retardo de los paquetes transportados por la red, ya que los paquetes de control compiten con los paquetes de datos por el ancho de banda disponible.

La inundación ciega representa una solución simple y robusta para el descubrimiento de rutas, pero también implica un enorme costo en términos de ancho de banda y tiempo de extremo a extremo. Aunque los nodos sólo retransmiten el paquete de búsqueda una sola vez, en el caso de redes densas con muchos nodos pueden presentarse muchas colisiones en el envío de paquetes durante la transmisión entre nodos vecinos. Estas colisiones generan varias retransmisiones innecesarias que incrementan la ventana de contienda y la tasa de colisiones de la red, ocasionando lo que se conoce como el problema de la tormenta de mensajes difusión (*the broadcast storm problem*) **[13]**. En la actualidad la eficiencia en el uso de los recursos de cualquier red es fundamental para proporcionar un correcto desempeño de la misma. Debido a esto existen distintas propuestas que tratan de mejorar las deficiencias que presenta el algoritmo de *inundación ciega*, algunas de ellas *MPR* (*Multipoint Relays*) **[2]**,

FRESH (*FResher Encounter SearcH*) **[3]** y *NARD* (*Neighbor-Assisted Route Discovery Protocol*) **[4]** que estudiaremos en el siguiente capítulo.

En este trabajo de investigación, se proponen dos algoritmos de búsqueda, el primero busca disminuir la cantidad de mensajes enviados a través de la red utilizando dos tipos de inundaciones; la inundación rápida o sin retardo y la inundación lenta que introduce retardos en el proceso denominado reenvío (*forwarding*). La inundación lenta consiste en que el nodo origen inicia una inundación ciega en la red pero cada vez que un nodo recibe un mensaje que no está dirigido a él, en vez de reenviarlo inmediatamente deja pasar un pequeño intervalo de tiempo para que en caso de que el nodo destino sea encontrado, éste tenga tiempo de iniciar una segunda inundación. La segunda inundación es la que denominamos inundación rápida ya que no está afectada por el proceso de retardo para poder avisar que el nodo destino ya recibió el mensaje de búsqueda y así contener la primera inundación lenta en sólo una porción de la red opuesto a inundación ciega que siempre inunda toda la red.

El segundo algoritmo de búsqueda también busca reducir la señalización en la red; pero en este caso el nodo origen en vez de iniciar una inundación ciega, seleccionará a un solo nodo de su vecindario para enviarle el mensaje de búsqueda. Al recibir el mensaje el nodo seleccionado, si no es el nodo buscado, seleccionará a su vez un nodo de su vecindario de forma aleatoria para reenviarle el mensaje. Este proceso se repetirá hasta que se encuentre el nodo destino o hasta que el nodo que reciba el mensaje ya no tenga vecinos a quien reenviar el mensaje.

Este trabajo está estructurado en cinco capítulos. En el primer capítulo se dará una breve introducción de lo que se pretende realizar en esta investigación. En el segundo capítulo se mostrará el trabajo relacionado en protocolos de búsqueda en redes *ad hoc*. En el tercer capítulo se presentarán los algoritmos propuestos en esta investigación. En el cuarto capítulo se mostrarán los resultados obtenidos de los mismos. Finalmente en el quinto se presentan las conclusiones.

1.2 JUSTIFICACIÓN Y DELIMITACIÓN

La investigación en redes inalámbricas es un campo muy amplio y demandante, ya que aunque este tipo de redes ofrecen muchas ventajas, también presentan problemas que no ocurren en las redes alambradas como son: ofrecer un menor ancho de banda, el problema conocido como la terminal oculta **[17]**, variabilidad de la topología, etc. En el mundo existen diversos grupos de investigación intentado resolver y mejorar muchos de estos aspectos negativos que se presentan en este tipo de redes.

Aunque existen distintos tipos de redes inalámbricas, en este trabajo sólo se busca presentar propuestas para mejorar el proceso de encontrar rutas entre nodos, así como el efecto de estos algoritmos en el consumo de recursos en las redes inalámbricas *ad hoc.* También se pretende que el funcionamiento de los algoritmos creados puedan ser aplicados a las *DTN* (*Delay Tolerant Network*) redes tolerantes a retardos **[18]**.

1.3 OBJETIVO

El objetivo de este trabajo de tesis es proponer dos algoritmos de inundación tolerante a retardos en redes *ad hoc* y evaluar su desempeño en comparación con otros métodos existentes mediante el uso de un simulador de redes inalámbricas como *NS2*.

1.4 HIPÓTESIS

Es posible diseñar un algoritmo de búsqueda que pueda mejorar el desempeño en cuestión de señalización ofrecido por el algoritmo inundación ciega mediante la modificación del mecanismo de reenvío que utiliza cada nodo, con base en el concepto de redes tolerantes al retardo *DTN*.

1.5 MÉTODO DE INVESTIGACIÓN

Primeramente, se realizará una búsqueda de toda la información disponible sobre el simulador de redes *NS2* **[9]**, protocolos de encaminamiento y su desempeño, en particular para redes *ad hoc*. Ya que *NS2* no cuenta con una opción para manejar retardos independientes para cada nodo, se tendrá que modificar el código del simulador tanto en los módulos de *TCL* y *C++*.

Se implementará una red inalámbrica en el simulador *NS2* con una cantidad de nodos considerable (400 nodos) y con el origen del mensaje de inundación localizado en el centro de la red (aproximado) para para hacer mediciones de tiempo de descubrimiento de ruta y cantidad de mensajes de control enviados en la red hasta encontrar el destino.

Los algoritmos propuestos se programarán en código *tcl* y los resultados del desempeño se tomarán de la información generados por el simulador y lo mostrado por la interfaz *NAM*. Se generarán tablas y gráficas que permitan hacer comparaciones de desempeño entre los algoritmos propuestos y el algoritmo de *inundación ciega*. Al final, y con toda la información recabada, se realizarán las conclusiones correspondientes.

El tipo de investigación que se llevará a cabo será de tipo documental, descriptiva y correlacional, porque en un principio se tiene que consultar una gran cantidad de información relacionada con las propiedades más importantes de las redes *ad hoc*, algoritmos, nuestras propuestas a implementar y su estrecha correlación para que en su momento se pueda hacer una correcta descripción, explicación, evaluación y comparación de este trabajo.

1.6 APORTES DE ESTE TRABAJO

Como resultado de este trabajo de tesis, se ofrecen algunos algoritmos que puedan ser empleados como mecanismos de búsqueda en la fase de descubrimiento de rutas de distintos protocolos de encaminamiento planos para redes ad hoc. Su objetivo es proporcionar una mayor eficiencia para dicha fase en comparación con lo logrado por el algoritmo de inundación ciega.

CAPÍTULO II ESTADO DEL ARTE

En este capítulo se describirá con mayor detalle el trabajo relacionado que se mencionó en el capítulo anterior. Al final se mostrará una tabla comparativa que muestre de manera general las ventajas y desventadas de cada algoritmo mencionado.

2.1 TRABAJO RELACIONADO

Es muy importante conocer los diferentes trabajos que buscan aportar soluciones a las deficiencias que presentan los algoritmos de búsqueda de rutas de las redes inalámbricas *ad hoc*. Como veremos, cada uno de ellos tiene fortalezas y debilidades con respecto a los otros. En el capítulo anterior, se mencionaron algunos algoritmos como *inundación ciega*, *NARD*, *AODV*, *FRESH*, etc., porque conceptualmente comparten algunas características con los algoritmos que se proponen y explican en el siguiente capítulo. Después de ofrecer una explicación de cada uno de ellos, se creará una tabla comparativa señalando sus características más importantes.

2.2 INUNDACIÓN CIEGA

El algoritmo de *inundación ciega* es el mecanismo de búsqueda más utilizado en la práctica por su robustez y simplicidad, ya que garantiza que los paquetes serán entregados (si es que el destino es realmente alcanzable), aunque utilizarlo conlleva algunos inconvenientes. Como se explicó en el capítulo anterior, esta técnica consiste en la diseminación de paquetes de control en toda la red; cuando un nodo recibe un paquete de búsqueda por primera vez lo retransmite. El gran número de paquetes que se generan al utilizar este tipo de búsqueda de nodos tiene el inconveniente de que, en condiciones de sobrecarga de la red, puede incrementar sensiblemente el retardo de los paquetes transportados por ella. Esto se debe a que los paquetes de control compiten con los paquetes de datos por el ancho de banda disponible. El algoritmo de *inundación ciega* es fácil de implementar ya que no existe la necesidad de distribuir gran cantidad de información por la red, ni realizar cálculos complejos de ruta.

Aunque en el algoritmo de *inundación ciega* los nodos sólo retransmiten el paquete de búsqueda una sola vez, en el caso de redes con una alta concentración de nodos pueden existir varias retransmisiones innecesarias que incrementen el tiempo de contienda y la tasa de colisiones de la red, ocasionando lo que se conoce como el problema de la tormenta de mensajes de difusión (*the broadcast storm problem*).

2.3 AODV

El protocolo *AODV (Ad-Hoc On-Demand Distance Vector*) **[7]**, como su nombre lo indica, tiene un funcionamiento bajo demanda, donde los nodos únicamente ocupan las rutas que están usando actualmente o que han utilizado recientemente. Las rutas utilizadas por este algoritmo están libres de ciclos (*loops*). Este protocolo utiliza un proceso de descubrimiento de rutas a través de mensajes de difusión, donde hay tres tipos de mensajes; *Route Request* (*RREQ*), *Route Reply* (*RREP*) y *Route Error* (*REER*) y se diseminan por la red usando el protocolo *UDP*. Cuando un nodo quiere establecer una conexión con otro nodo destino, primero verifica que no haya alguna ruta válida hacia ese nodo, si tiene una ruta simplemente la utiliza y no se inicia *AODV*. En caso contrario, comienza un proceso llamado “Descu rimiento de camino” (*path discovery*). Durante este proceso se envía un mensaje *RREQ* a sus vecinos, si éstos tienen una ruta a dicho destino responden al nodo origen con un mensaje *RREP*, si no, entonces también reenvían el mensaje *RREQ* en modo difusión después de incrementar un contador llamado conteo de saltos (*hop count*).

Durante el proceso de descubrimiento de ruta, cada vez que un nodo reenvía el mensaje *RREQ* también debe almacenar la información del nodo vecino del que recibió el mensaje *RREQ* para que automáticamente se vaya estableciendo la ruta inversa hacia el nodo origen y, una vez que sea encontrado el nodo destino, éste pueda enviar un mensaje *unicast* de respuesta utilizando esa ruta inversa.

Una vez que el nodo origen recibió el primer mensaje *RREP*, éste comienza a transmitir los datos al nodo destino. Si posteriormente recibe una mejor ruta que requiera una menor cantidad de saltos para llegar al nodo destino, no hay problema en actualizar su información de encaminamiento.

El nodo origen puede distinguir las rutas más recientes mediante un par metro llamado “*Destination Sequence Number*”. Como se mencionó anteriormente, cuando un nodo intermedio recibe un mensaje *RREQ*, se reenvía o prepara un mensaje *RREP*. La validez de esa ruta proporcionada por el nodo intermedio es determinada al comparar el número de secuencia en el nodo intermedio con el del paquete *RREQ*. En este algoritmo existen distintos *timers* que se utilizan para ir eliminando la información considerada obsoleta como son: los caminos inversos (*reverse paths*), los caminos de avance (*forward paths*), los elementos de las tablas de rutas que no hayan sido utilizadas en determinado tiempo, etc.

Como se mencionó anteriormente, este protocolo tiene las ventajas de contar con rutas bajo demanda, números de secuencia para encontrar las rutas más actuales al destino y que el tiempo de retraso durante el proceso de conexión

sea poco. Aunque por otro lado, como desventajas podemos mencionar que los nodos intermedios pueden llevar a ofrecer rutas inconsistentes si el número de secuencia es viejo, la capacidad de producir múltiples mensajes de *RREP* en respuesta a un sólo *RREQ* generando sobrecarga en la red y el consumo innecesario de ancho de banda debido al envío periódico de señales de control (*beacons)*.

2.4 DSR

El protocolo *DSR* (*Dynamic Source Routing*) **[8]** es un protocolo de encaminamiento que utiliza un mecanismo llamado *source routing*, en el cual cada nodo es capaz de obtener la dirección de los nodos intermedios entre el nodo origen y el nodo destino. La información de estas direcciones es incluida en la cabecera del mensaje. Aunque *DSR* tiene la desventaja de generar sobrecarga en el rendimiento de la red cuando la ruta entre el origen y el destino es muy larga o si se utilizan direcciones *IPv6*, se pueden vislumbrar algunos puntos positivos a su favor. Por ejemplo, no se producen ciclos en las rutas, es posible el uso de enlaces unidireccionales, los nodos intermedios no necesitan mantener tablas de encaminamiento actualizadas ya que toda la información se encuentra en el nodo iniciador (origen). También, y debido a la posible existencia de más de una ruta a un nodo destino, puede haber un mejor balance de carga.

DSR utiliza dos mecanismos, *Route Discovery* y *Route Maintenance*, donde el primero se encarga de obtener todos los nodos intermedios por los cuales tiene que pasar el paquete para llegar al nodo destino y el segundo se encarga de detectar cuando una ruta hacia un determinado nodo ha dejado de ser útil debido a posibles cambios en la topología de la red. Como lo sugiere el nombre del protocolo, ambos mecanismos funcionan bajo demanda.

Cuando un nodo origen o iniciador requiere enviar datos a un nodo destino, primero verifica en su "*route cache*" para ver si tiene alguna ruta para llegar a ese destino, si la tiene la utiliza, de lo contrario se inicializa el mecanismo *Route Discovery*. De la misma manera que en *AODV*, el nodo origen transmite un mensaje *RREQ* pero con un identificador único "*request ID*" que permitirá determinar cuándo se recibe más de una vez el mismo mensaje. El *RREQ* también contendrá la información del nodo origen y el nodo destino que se desea encontrar y la lista de los nodos por los cuales el mensaje ha ido pasando. Cada vez que un nodo recibe el mensaje *RREQ*, si no es el nodo destino y recibe por primera vez el mensaje, él mismo se agrega a la lista de nodos por los cuales ha pasado el mensaje y lo retransmite.

En caso de que el nodo que recibe el mensaje *RREQ* sea el nodo destino, éste le contestará al nodo origen o iniciador con un mensaje *RREP* con la lista de todos los nodos intermedios por donde pasó el mensaje en modo *unicast*. Una vez que el nodo origen haya recibido el mensaje de respuesta, copiará la ruta en una cache llamada "*route cache*".

Cuando un nodo no obtenga respuesta después de haber retransmitido un mensaje varias veces, éste debe enviar un mensaje de error *REER* al nodo origen para avisarle que el destino es inalcanzable. El nodo origen verificará si tiene otra ruta disponible de lo contrario deberá inicializar el mecanismo de *Route Discovery* una vez más.

2.5 TORA

El protocolo *TORA* (*Temporally Ordered Routing Algorithm*) **[12]** es un algoritmo de encainamiento en demanda, adaptativo y eficiente, propuesto para redes muy dinámicas. Tiene la característica de encontrar muchas rutas de un nodo origen a otro destino utilizando mensajes de control. Estos mensajes están localizados en una pequeña porción de nodos cerca de donde ocurrió el cambio topológico. Para lograr ese objetivo, los nodos mantienen información acerca de los nodos adyacentes.

Las tres funciones básicas de *TORA* son: *Route creation*, *Route Maintenance* y *Route erasure*. Para iniciar una ruta (y también para el proceso de mantenimiento), el nodo envía un paquete en modo difusión llamado *QUERY* (*QRY*) a sus vecinos, este paquete es reenviado a través de la red hasta alcanzar su destino o algún nodo intermedio que tenga la ruta a dicho destino. Una vez encontrado el nodo destino, éste responde al nodo origen con un mensaje llamado *UPDATE (UPD)*. Finalmente existe un tipo de mensaje llamado *CLEAR (CLR) que* se utiliza en modo difusión para eliminar las rutas inválidas cuando un nodo es inaccesible.

Una característica muy importante de este protocolo es que tiene la posibilidad de mantener múltiples rutas a un destino de tal manera que los cambios topológicos no requieren una importante reacción. El protocolo funciona sólo cuando todas las rutas a un destino están perdidas. En el evento de una partición de la red el protocolo es capaz de detectar esa partición o eliminar todas las rutas inválidas. Todo el proceso de eliminar las rutas inválidas, buscar nuevas alternativas para llegar a un destino y construir las nuevas rutas se realiza en un sólo paso del protocolo a diferencia de *AODV* o *DSR*, donde se tiene que realizar *route error*, *route request* y *route reply*.

Aunque *TORA* tiene distintas bondades como los múltiples caminos creados, sólo buscar rutas cuando es necesario o que se comporte bien en redes densas, su uso es limitado ya que *DSR* y *AODV* lo superan en desempeño.

2.6 MPR

MPR (*Multipoint Relays*) **[2]** es un esquema utilizado en el protocolo proactivo *OLSR* (*Optimized Link State Routing*) que tiene algunas características que vale la pena señalar. El objetivo de *MPR* es reducir la inundación de paquetes de difusión en la red, al minimizar las retransmisiones duplicadas localmente. Cada nodo selecciona algunos nodos de su vecindario llamados *Multipoint Relays* (*MPR*) para que sólo ellos retrasmitan los paquetes de difusión y no el vecindario completo. La identificación del vecindario de cada nodo se realiza mediante el envío periódico de mensajes *hello.* También cada nodo obtiene información topológica de la red a través del envío de mensajes de control llamados TC (topology control). En la *Figura 2.1* los nodos MPR están ilustrados con una *X*. Este procedimiento tiene la finalidad de permitir a los vecinos que no pertenecen al grupo de los *MPR* tener la posibilidad de leer los mensajes pero no retransmitirlos, y así prevenir lo que se conoce como el problema de la tormenta de mensajes de difusión (*the broadcast storm*) que explicamos en el capítulo anterior.

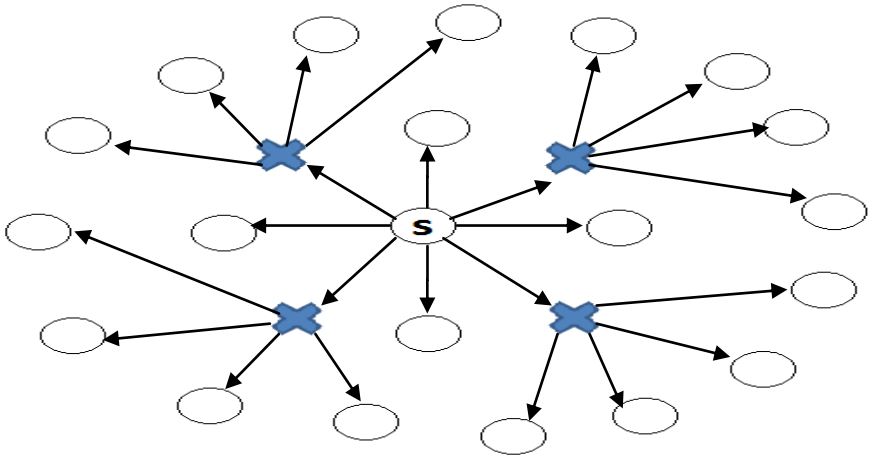


FIGURA 2.1 GRUPO DE MPRs

La tarea de cada nodo que retrasmite el mensaje será elegir los nodos vecinos que formarán parte del grupo de *MPR,* que garanticen que todos los nodos que se encuentren a dos saltos de distancia reciban el paquete como se muestra en la *Figura 2.1*. *MPR* sigue una regla simple, un nodo MPR retransmite un paquete de difusión si y sólo si fue recibido por primera vez de un nodo *MPR*. La mayor ganancia obtenida por la introducción de *MPR* es que mientras más pequeño sea el conjunto de *MPR*, menor será el número de retransmisiones de

paquetes. La información del vecindario y la topología de la red es usada por el protocolo OSLR para que cada nodo pueda calcular las rutas a cualquier destino conocido mediante el uso del algoritmo del camino más corto de Dijkstra.

2.7 NARD

El protocolo *NARD* (*Neighbor-assisted route discovery protocol*) **[4]** es un protocolo eficiente para el descubrimiento de rutas enfocado principalmente para redes grandes, en donde el algoritmo de inundación por fuerza bruta no es una solución práctica. El objetivo de *NARD* es reducir significativamente el número de paquetes de control diseminados en la red y así liberar ancho de banda. En *NARD* un nodo fuente inunda una porción limitada de la red buscando no sólo el nodo destino, sino también información relacionada a otros nodos (llamado vecindario destino) que estuvieron cerca del nodo destino recientemente. Esos nodos son utilizados como nuevos puntos de referencia para iniciar una segunda inundación limitada. Como sólo dos porciones de la red son inundadas con paquetes de control, esta técnica puede reducir la señalización significativamente en comparación con otros algoritmos.

La operación de *NARD* se compone de tres etapas llamadas *neighbor-discovery phase*, *neighbor-search phase* y *target-search phase*. El principal objetivo de la fase *neighbor-discovery phase* es encontrar la identidad de los vecinos que se encuentran a un salto de distancia y almacenar esa información en tablas llamadas tablas de vecinos (*neighbor tables*). Esa información será compartida con otros nodos cada vez que se establezca una conexión del tipo *TCP* o *UDP*. El funcionamiento de esta fase está ilustrado en *la Figura 2.2*.

En la fase *neighbor-search phase* cuando un nodo intenta transmitir un paquete a otro nodo, primero revisa si ya tiene una ruta válida para ese nodo, si la tiene entonces envía el paquete al siguiente salto, de lo contrario envía un mensaje *RREQ* que incluye las direcciones *IP* del nodo origen y el nodo destino. Esta búsqueda inicial es diseminada por los vecinos que se encuentran a *n* saltos de distancia del nodo origen. El funcionamiento de esta fase está ilustrado en la *Figura 2.3*.

En *NARD* no sólo se busca la dirección del nodo destino sino también información reciente de vecinos del nodo destino. Cuando los nodos reciben el *RREQ*, revisan su tabla de vecinos y aquellas tablas que hayan adquirido durante conexiones previas con otros nodos. En caso de que encuentre una ruta válida al nodo destino o antiguos vecinos, le contestan al nodo origen con un mensaje *unicast RREP*.

En caso de que no se obtenga información de una primera búsqueda porque tal vez el valor de *n* saltos fue pequeño, se debe probar con un valor mayor

de *n*, lo que aumentaría la probabilidad de éxito pero podría generar mayor señalización y retrasos que si se utilizara inundación por fuerza bruta.

En la tercera fase llamada *Target-search phase*, una vez que el nodo origen recibió las respuestas de sus vecinos, envía paquetes en modo *unicast*, llamados paquetes *SEARCH* a los posibles vecinos del nodo destino. Una vez que estos nodos reciben esos paquetes, verifican si tienen información sobre el nodo destino y, si no es así, envían un mensaje *RREQ* a una porción limitada de la red a *k* nodos de distancia. Si el nodo destino recibe el mensaje *RREQ*, le contesta al nodo vecino con un mensaje *RREP* y éste a su vez reenvía el mensaje al nodo origen. Esto lo podemos ver en la *Figura 2.4*.

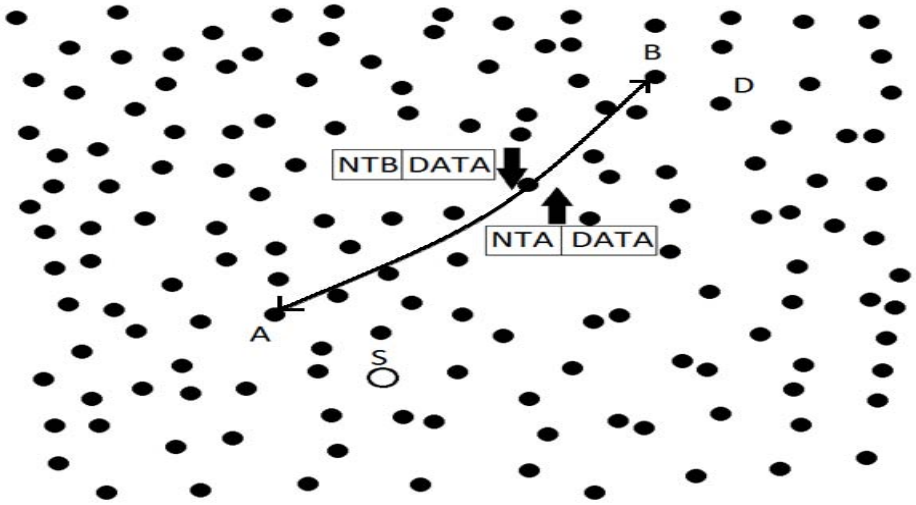


FIGURA 2.2 PRIMERA FASE NEIGHBOR-DISCOVERY PHASE. NTA Y NTB SON LAS TABLAS DE VECINOS TRANSMITIDAS EN ESTA FASE.

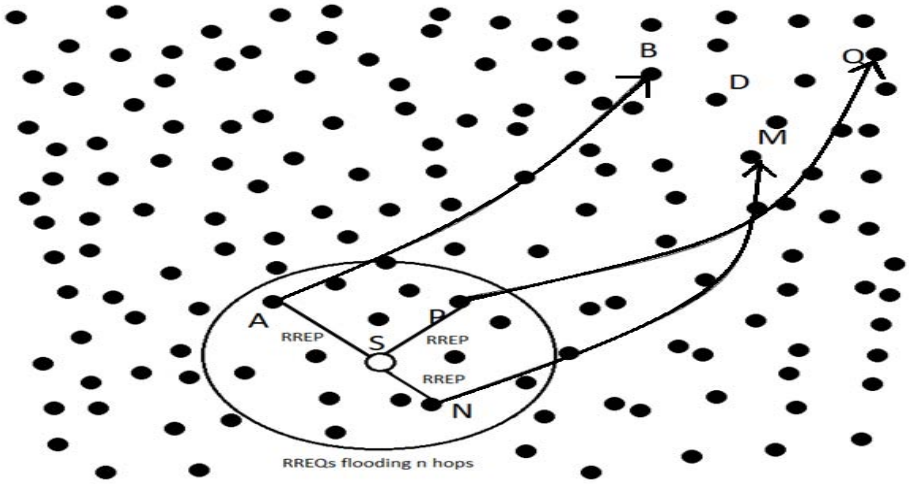


FIGURA 2.3 SEGUNDA FASE NEIGHBOR-SEARCH PHASE

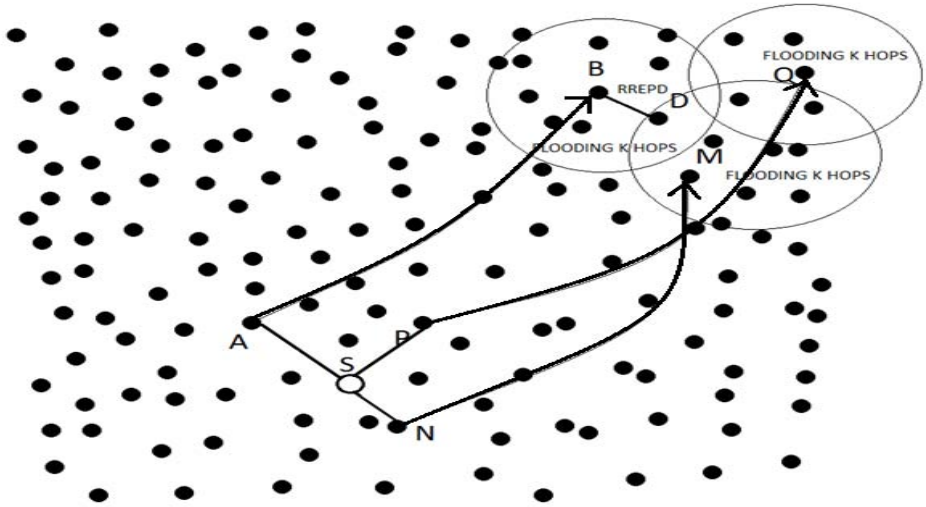


FIGURA 2.4 TERCERA FASE TARGET-SEARCH PHASE

2.8 FRESH

El algoritmo denominado *FRESH* (*FResher Encounter SearcH*) **[3]**, es un algoritmo simple que busca disminuir la señalización en la red bajo el concepto de enfocar la búsqueda en la dirección adecuada en la red. Para lograr ese objetivo *FRESH* guarda la información de los encuentros más recientes de cada nodo y en vez de buscar por el nodo directamente, el nodo origen busca en una porción limitada de la red a cualquier nodo que haya encontrado al nodo destino más recientemente que el mismo nodo origen. En *FRESH* los nodos tienen una tabla que contiene los encuentros más recientes con otros nodos, un encuentro ocurre cuando dos nodos son vecinos con una separación de un salto de distancia. El tiempo transcurrido desde el encuentro más reciente que han tenido dos nodos es lo que se llama edad de encuentro (*encounter age*), y sirve para diferenciar la antigüedad de los encuentros. Los encuentros pueden detectarse al tener la interfaz del nodo en modo promiscuo y así escuchar cualquier paquete de datos o directamente enviar mensajes hola (*hello*) a nodos vecinos.

En *FRESH* el nodo origen busca a un nodo ancla (*anchor*) de tal manera que la antigüedad del encuentro del nodo ancla con el nodo destino sea menor que el que tiene al nodo origen. A su vez, el nodo ancla busca a su alrededor por algún nodo que haya tenido un encuentro todavía más reciente con el nodo buscado. Este procedimiento se itera hasta que el nodo destino sea encontrado como se muestra en la *Figura 2.5*.

El desempeño de este esquema dependerá del proceso de movilidad de los nodos y del tráfico generado. Las rutas obtenidas por este algoritmo están libres de ciclos (*loops*) y este algoritmo no requiere conocimiento global ya que cada búsqueda está definida en términos de las tablas locales de cada nodo.

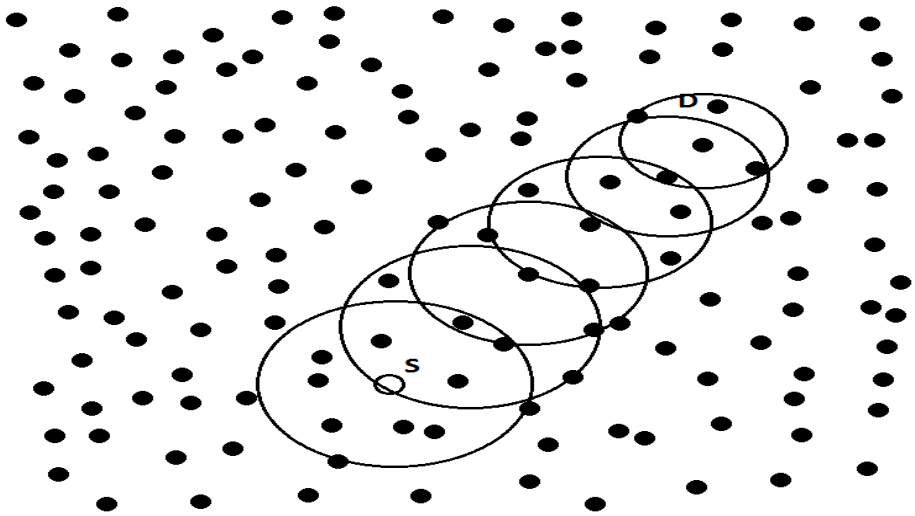


FIGURA 2.5 ALGORITMO FRESH

Como mencionamos en el capítulo anterior, en este trabajo nos enfocamos en los protocolos de encaminamiento planos, en particular la subdivisión de protocolos reactivos donde se encuentran clasificados *AODV*, *DSR* y *TORA*. Debido a que estos protocolos son los más utilizados y comúnmente utilizan inundación ciega como mecanismo de búsqueda en su fase de descubrimiento de rutas, han surgido distintas propuestas que pretenden ofrecer una mayor eficiencia para dicha fase.

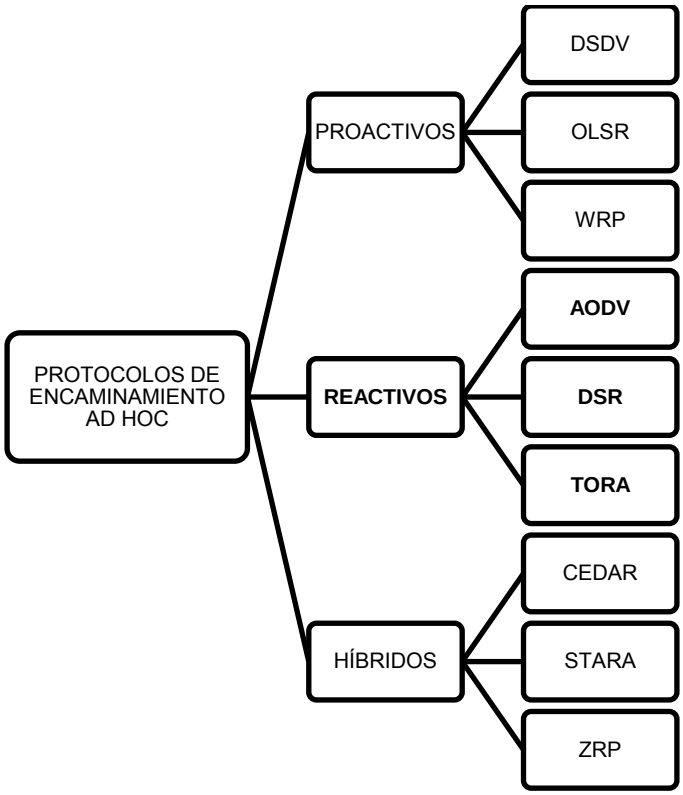


FIGURA 2.6 PROTOCOLOS DE ENCAMINAMIENTO AD HOC

Aquí se describieron los casos de *FRESH* y *NARD* porque estos algoritmos al igual que dos de los propuestos en esta tesis, buscan disminuir el número de paquetes de control diseminados en la red modificando la manera en que se comportan las inundaciones. Mencionamos a *MPR* porque aunque es un mecanismo que está inmerso en el protocolo proactivo *OSLR*, su funcionamiento tiene coincidencias con los dos últimos algoritmos descritos en este trabajo. Dichas similitudes se encuentran en la manera en que se busca disminuir la inundación de paquetes de difusión en la red al minimizar las retransmisiones duplicadas localmente enviando sólo a nodos específicos del vecindario de los nodos que transmiten.

2.9 TABLA DE COMPARACIÓN

	INUNDACIÓN FUERZA BRUTA **[11] [12]**	AODV **[7]**	DSR **[8]**	TORA **[12]**	MPR **[2]**	NARD **[4]**	FRESH **[3]**
VENTAJAS	Siempre encuentra el nodo destino si éste está disponible. Es simple, no hay conocimiento global de la red.	Rutas libres de ciclos. Número de secuencia para rutas.	Rutas libres de ciclos. No usa tablas de encaminamiento. Posible uso de balance de carga.	Ideal para redes densas. Mensajes de control donde ocurrió el cambio en la red. Múltiples rutas a un destino.	Sólo disemina paquetes de difusión por los nodos *MPR*. Disminución de señalización. Útil para redes grandes.	Especialmente útil para redes grandes. Sólo inunda porciones específicas de la red.	También inunda porciones específicas de la red y reduce señalización.
DESVENTAJAS	Mucha señalización. Malo para redes grandes. Ineficiente.	Múltiples respuestas a sólo un mensaje RREQ. Posibles inconsistencias en nodos internos. Puede generar sobrecarga en la red y consumo innecesario de ancho de banda de la red.	Sobrecarga en la red. Grandes cabeceras en mensajes. No adecuado para grandes redes.	Peor desempeño que AODV y DSR.	Jerarquización de nodos. Actualmente usado más en protocolos proactivos.	Esencial encontrar valores correctos de n y k para evitar mucha señalización.	La cantidad de señalización depende el proceso de movilidad de los nodos. Los nodos deben almacenar información.

CAPÍTULO III ALGORITMOS PARA FASE DE DESCUBRIMIENTO DE RUTAS

En este capítulo se describirán y explicarán de manera detallada el funcionamiento de los algoritmos propuestos. También se presentará el pseudocódigo de cada uno de ellos.

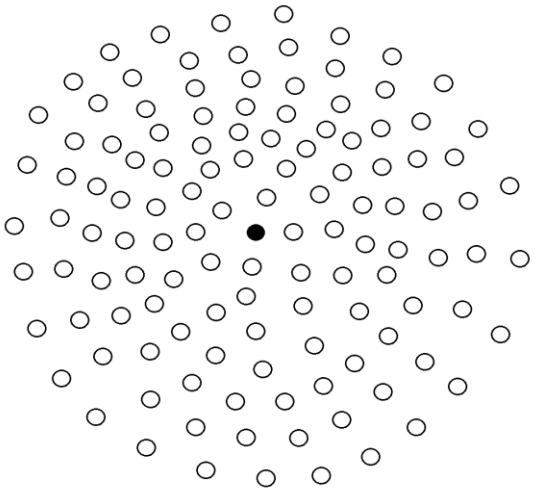
3.1 ALGORITMO A

El primer algoritmo que presentaremos es el que hemos denominado Algoritmo *A*. La idea de este algoritmo es iniciar una *inundación ciega* que no disemine mensajes inmediatamente por toda la red, sino que los mensajes, una vez que se sean recibidos por los nodos, los contengan durante un determinado tiempo. En caso de que uno de esos nodos sea el nodo destino, el nodo destino será capaz de iniciar una segunda *inundación ciega* sin retardo que pueda contener la primera *inundación ciega* con retardo y así buscar disminuir la señalización en la red. El Algoritmo *A* funciona de la siguiente manera:
El nodo origen envía un mensaje de inundación a sus vecinos buscando al nodo destino.

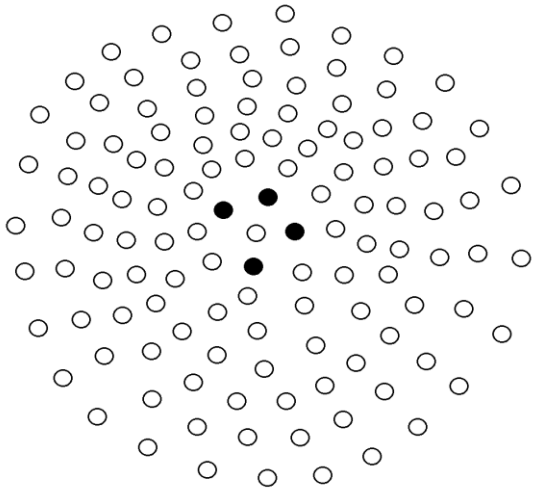
- Si el nodo vecino no es el destino y recibe el mensaje de inundación con retardo (RREQ), éste lo reenvía después de un retardo de Δt segundos.
- Si el nodo vecino es el destino del mensaje de inundación con retardo (RREQ); éste contesta con un segundo mensaje de inundación sin retardo (RREP) dirigido al nodo origen.
    - Si el nodo vecino recibe el mensaje de inundación sin retardo (RREP) después de haber recibido el mensaje de inundación con retardo (RREQ), reenvía inmediatamente el mensaje RREP.
    - Si el nodo vecino no ha recibido previamente el mensaje de inundación con retardo (RREQ) y recibe el mensaje de inundación sin retardo (RREP), no reenvía el mensaje RREP.
    - Si el nodo vecino ya recibió previamente el mensaje de inundación sin retardo (RREP) y recibe el mensaje de inundación con retardo (RREQ), no reenvía el mensaje RREQ.

La *Figura 3.0* ilustra el comportamiento del algoritmo *A* de una manera gráfica. En primer lugar el nodo origen envía un mensaje de búsqueda mediante una *inundación ciega*, estos paquetes son ilustrados con ● Cada vez que los nodos vecinos reciben dicho mensaje lo retienen durante un intervalo de tiempo antes de reenviarlo. Cuando el nodo destino recibe el mensaje de búsqueda, éste le contesta al nodo origen con una nueva *inundación ciega* donde los nodos que lo reciben *no* lo retienen como en la primera inundación, permitiendo que la segunda inundación alcance y contenga a la primera inundación en la red. Los nodos que participan en la segunda inundación se muestran como✖.

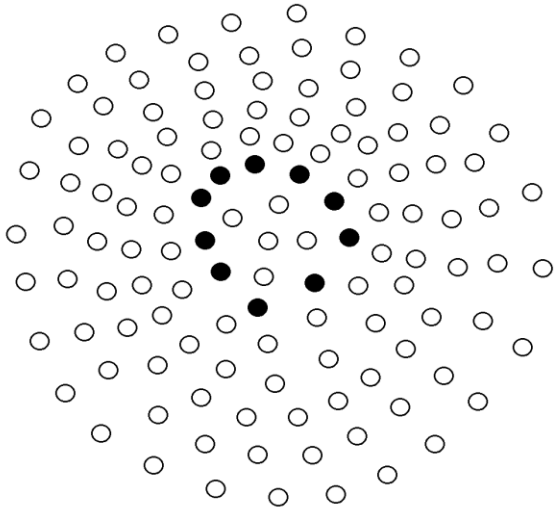
En la *Figura 3.0* se puede notar que, debido a los retardos insertados en el proceso de reenvío (*forwarding*) de cada nodo, el avance de la propagación de la primera inundación se lleva a cabo por saltos. Visualmente en el simulador cada vez que se envía el mensaje se observa como si se propagara en forma de ondas como se muestra en la *Figura 3.0*.



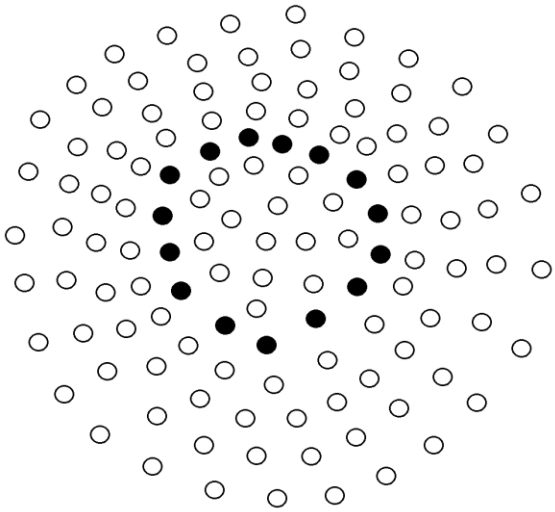
t1 NODO ORIGEN (INUNDACIÓN LENTA)



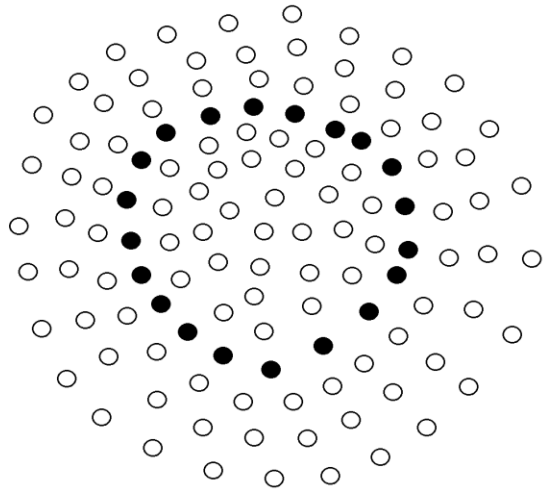
t2 PRIMER SALTO (INUNDACIÓN LENTA)



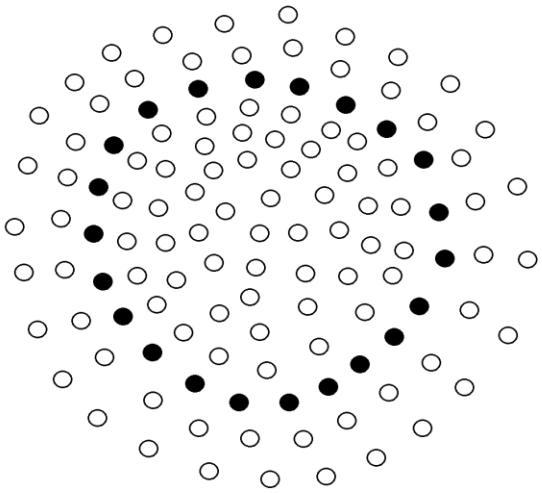
t3 SEGUNDO SALTO (INUNDACIÓN LENTA)



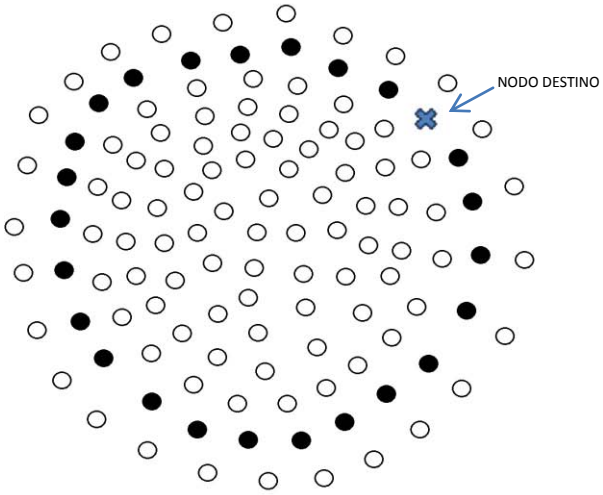
t4 TERCER SALTO (INUNDACIÓN LENTA)



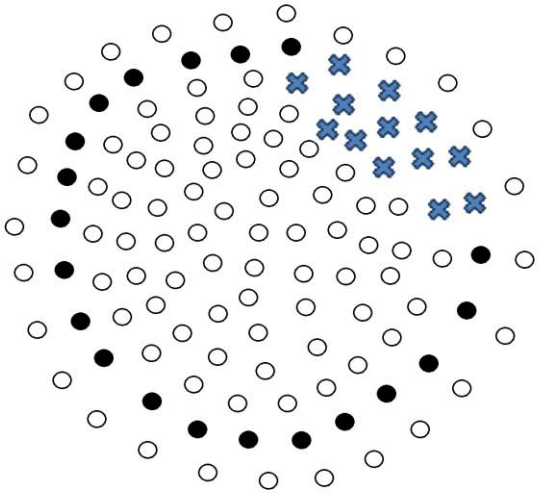
t5 CUARTO SALTO (INUNDACIÓN LENTA)



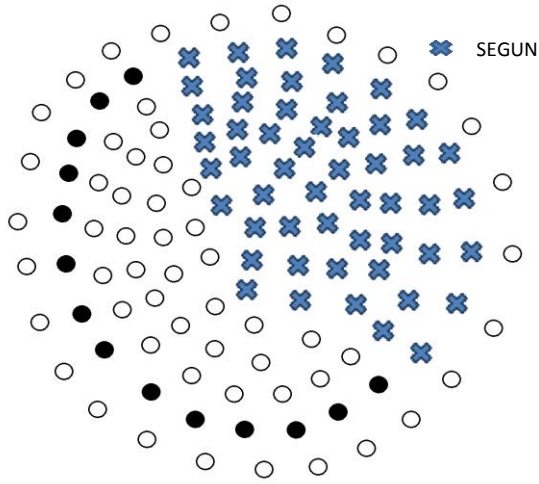
t6 QUITNO SALTO (INUNDACIÓN LENTA)



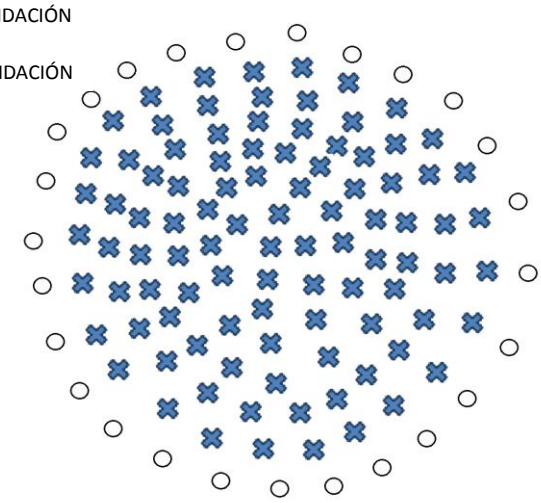
t7 SEXTO SALTO (INUNDACIÓN LENTA)



t8 INICIO SEGUNDA INUNDACIÓN (INUNDACIÓN RÁPIDA)



t9 DISEMINACIÓN DE LA INUNDACIÓN RÁPIDA



t10 FIN DE LA INUNDACIÓN RÁPIDA

FIGURA 3.0 FUNCIONAMIENTO DEL ALGORITMO A

En el algoritmo *A* se especifica que se debe dejar un retardo con una duración de Δt segundos cada vez que los nodos reciben los mensajes de búsqueda. Dicho retardo debe durar lo suficiente para que el nodo destino pueda iniciar una segunda *inundación ciega* sin retardo capaz de abarcar toda el área cubierta por la primera *inundación ciega* para detener su propagación. En el tiempo nueve de la *Figura 3.0* se ilustra como la segunda inundación abarca toda el área cubierta por la primera inundación para evitar que ésta se siga propagando en la red.

Al programar este algoritmo en el simulador podemos ver como inicialmente la propagación de los mensajes está bastante controlada, ya que los únicos nodos que reenvían el mensaje son los que se encuentran a la misma cantidad de saltos de distancia del nodo origen, como se muestra en los primeros seis tiempos de la *Figura 3.0*. El caso contrario ocurre cuando la *inundación ciega* se disemina libremente como se muestra en los t8, t9 y t10 de la *Figura 3.0*.

También, y gracias a los retardos insertados, notamos a cuantos saltos de distancia se encuentra cada nodo con respecto al nodo origen. Esto nos permite visualizar que mientras el nodo destino se encuentre lejos del nodo origen, la suma total de los mensajes enviados por la primera *inundación ciega* con retardo más los generados por la respuesta de la segunda *inundación ciega sin retardo* será una cantidad de mensajes alta, incrementando la señalización en la red. Como consecuencia es de esperar que este algoritomo funcione mejor en escenarios donde el nodo destino se encuentre cerca del nodo origen.

PSEUDOCÓDIGO ALGORITMO A

Origen envía mensaje:

- **IF** (nodo != destino 1 **AND** mensaje recibido= 1) **THEN**
  registrar mensaje 1
  retardo = X
  reenviar
- **IF** (nodo = destino 1 **AND** mensaje recibido= 1) **THEN**
  registrar mensaje 1
  retardo = 0
  Envía mensaje 2
  - **IF** (nodo != destino 2 **AND** mensaje recibido = 2 **AND** mensaje 1 visto) **THEN**
    registrar mensaje 2
    retardo = 0
    reenviar
  - **IF** (nodo != destino 2 **AND** mensaje recibido = 2) **THEN**
    registrar mensaje 2
  - **IF** (nodo != destino 2 **AND** mensaje recibido = 1 **AND** mensaje 2 visto) **THEN**
    registrar mensaje 1

3.2 ALGORITMO A*

Durante el desarrollo del algoritmo *A* pudimos identificar una posible oportunidad de mejora cuando notamos que cada vez que el nodo destino se encuentra en una ubicación alejada del nodo origen, la cantidad de mensajes totales enviados es mayor a que si sólo utilizáramos una simple *inundación ciega* sin retardos.

Si de alguna manera encontráramos el límite en la red a partir de donde ya no es ventajoso el uso del algoritmo *A*, podríamos tener una menor cantidad de señalización en la red. Esto lo lograríamos si hacemos que la diseminación de los mensajes se comporte como una *inundación ciega* normal cuando los nodos destino se encuentran más allá de ese límite. A esta variación del algoritmo *A* la hemos denominado algoritmo *A**.

En la definición de este algoritmo *A** nos hemos referido a este límite en la red con el nombre de salto *X,* ya que el mensaje de búsqueda se va propagando a través de saltos. Las modificaciones que se realizaron al algoritmo *A* son las siguientes.

- Si el nodo vecino no es el destino, se encuentra a una distancia menor al del salto *X* y además recibe el mensaje de inundación con retardo, entonces éste lo reenvía después de un retardo de Δt segundos.
- Si el nodo vecino es el destino del mensaje de inundación con retardo y además se encuentra a una distancia menor al del salto *X*, entonces éste contesta con un mensaje de inundación sin retardo dirigido al nodo origen.
- Si el nodo vecino no es el destino, se encuentra a una distancia mayor al del salto *X* y además recibe el mensaje de inundación con retardo, entonces éste lo reenvía después de un retardo de Δt segundos.
- Si el nodo vecino es el destino del mensaje de inundación con retardo y además se encuentra a una distancia mayor al del salto *X*, entonces éste contesta con un mensaje directo al nodo origen (*unicast*) ocupando la ruta inversa que siguió el primer mensaje para llegar al nodo destino.
- Si el nodo vecino es el destino del segundo mensaje de búsqueda no hace nada.

En la *Figura 3.1* se muestra como en la red existe un límite imaginario, representado por un círculo que abarca una gran cantidad de nodos en la red, que se debe tomar en cuenta para determinar qué tipo de comportamiento tendrá el algoritmo dependiendo de la ubicación del nodo destino.

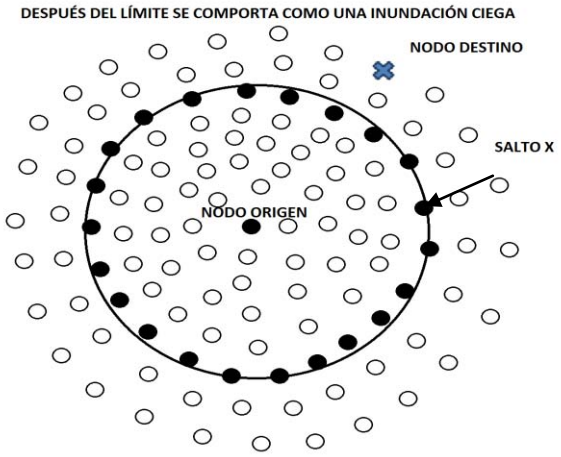
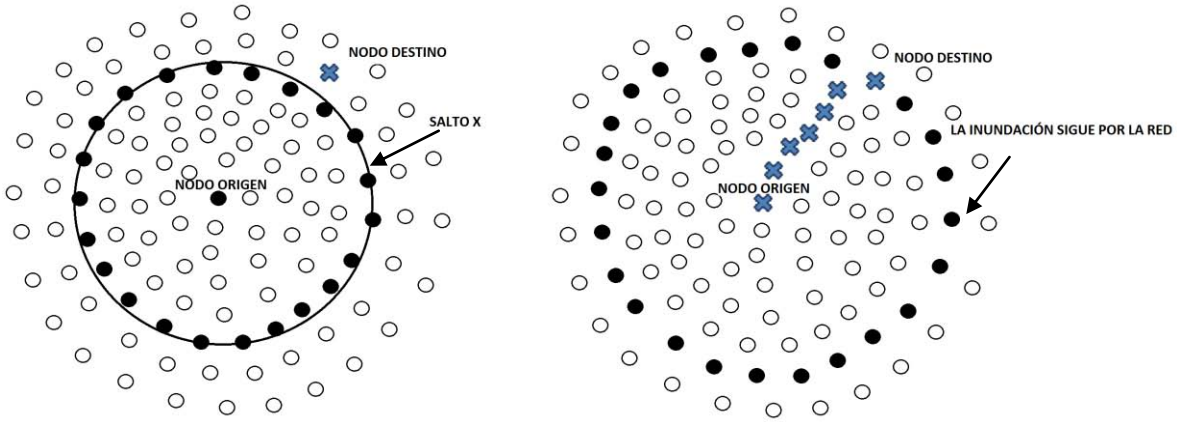


FIGURA 3.1 COMPORTAMIENTO GRÁFICO DEL ALGORITMO DESPUÉS DEL LÍMITE IMAGINARIO EN LA RED

La *Figura 3.2* es una vista del algoritmo *A** en ejecución, donde el nodo destino se encuentra más allá de ese límite imaginario en la red y para no generar tanta señalización, simplemente se deja diseminar la *inundación ciega* iniciada por el nodo origen por toda la red.



t1 INUNDACIÓN ALCANZA EL SALTO X t2 NODO DESTINO ENCONTRADO MÁS ALLÁ DEL SALTO X

FIGURA 3.2 COMPORTAMIENTO DEL ALGORITMO CUANDO EL NODO DESTINO ESTÁ MÁS ALLÁ DEL LIMITE

PSEUDOCÓDIGO ALGORITMO A*

Origen envía mensaje:

- **IF** (nodo != destino 1 **AND** mensaje recibido = 1 **AND** salto < #) **THEN**
  registrar mensaje 1
  retardo = X
  reenviar
- **IF** (nodo = destino 1 **AND** mensaje recibido= 1 **AND** salto < #) **THEN**
  registrar mensaje 1
  retardo = 0
  Envía mensaje 2
  - **IF** (nodo != destino 2 **AND** mensaje recibido = 2 **AND** mensaje 1 visto) **THEN**
    registrar mensaje 2
    retardo = 0
    reenviar
  - **IF** (nodo != destino 2 **AND** mensaje recibido = 2) **THEN**
    registrar mensaje 2
  - **IF** (nodo != destino 2 **AND** mensaje recibido = 1 **AND** mensaje 2 visto) **THEN**
    registrar mensaje 1
- **IF** (nodo != destino 1 **AND** mensaje recibido= 1 **AND** salto > #) **THEN**
  registrar mensaje 1
  retardo = X
  reenviar
- **IF** (nodo = destino 1 **AND** mensaje recibido= 1 **AND** salto > #) **THEN**
  registrar mensaje 1
  contestar a origen con ***unicast***

3.3 ALGORITMO B

El tercer algoritmo llamado Algoritmo *B*, es una idea diferente a los dos algoritmos anteriores. Aquí el nodo origen selecciona de manera aleatoria a sólo uno de los nodos que se encuentran en su vecindario y le envía un mensaje de búsqueda. Cuando el nodo vecino recibe dicho mensaje también selecciona de forma aleatoria un nodo de su vecindario, pero antes de enviarle el mensaje, verifica que ese nodo no haya enviado el mensaje previamente y si no es así se lo reenvía. El proceso se repite hasta encontrar al nodo destino o cuando el nodo que reciba el mensaje ya no tenga nodos vecinos a quien reenviar el mensaje. En la *Figura 3.3* se muestra cómo la ejecución del algoritmo termina cuando el nodo destino (nodo tres) es encontrado y éste envía su mensaje de respuesta.

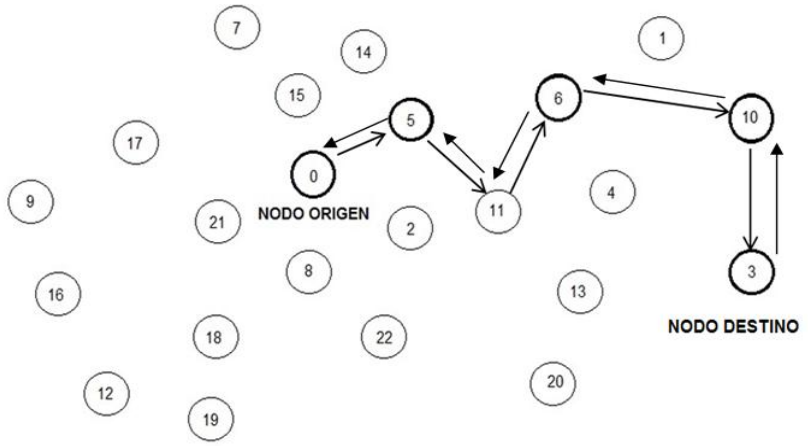


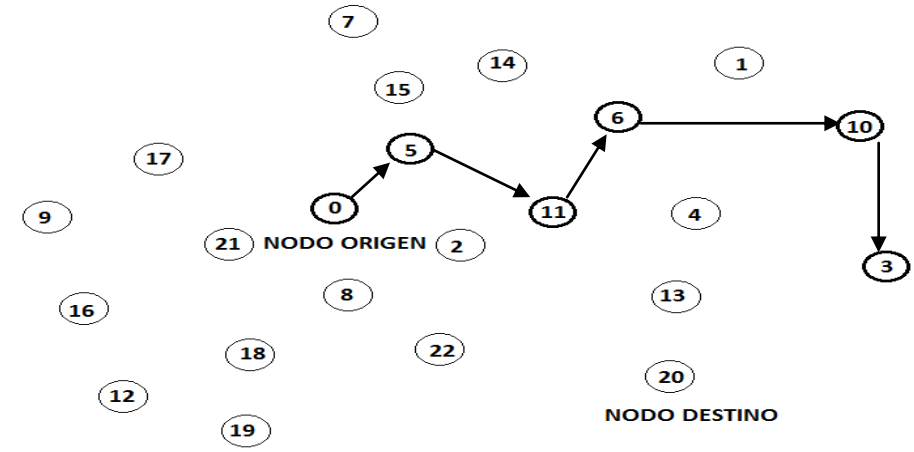
FIGURA 3.3 COMPORTAMIENTO GRÁFICO DEL ALGORITMO B

El funcionamiento del algoritmo *B* se describe con mayor detalle a continuación.

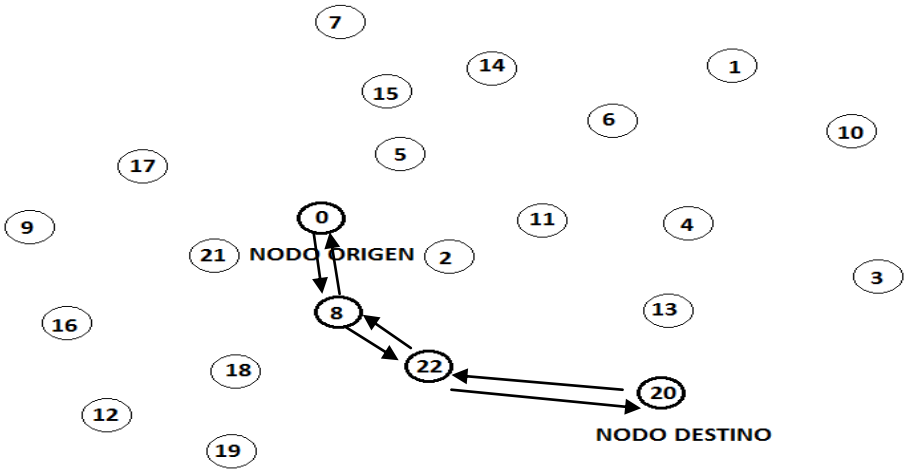
El nodo origen envía un mensaje dirigido a sólo uno de sus vecinos

- Si el nodo vecino no es el destino, este a su vez reenvía el mensaje de búsqueda a uno de los nodos que se encuentran en su vecindario (seleccionándolo de manera aleatoria), y que no haya sido seleccionado previamente.
- Si el nodo vecino es el destino, este envía un mensaje de regreso directo hasta el nodo origen (*unicast*) ocupando la ruta inversa que siguió el primer mensaje para llegar al nodo destino.

Si después de un retardo Δt no se recibió el mensaje de respuesta, el nodo origen envía otro mensaje de búsqueda a otro nodo de su vecindario al que no le haya enviado mensaje de búsqueda previamente como se muestra en la *Figura 3.4*.



t1 PRIMER MENSAJE NO ENCUENTRA AL DESTINO



t2 SEGUNDO MENSAJE SÍ ENCUENTRA AL DESTINO

FIGURA 3.4 COMPORTAMIENTO GRÁFICO DEL ALGORITMO B

El retardo Δt que espera el nodo origen para enviar otro mensaje de búsqueda debe ser suficiente para que el mensaje previo haga su recorrido. Cabe señalar que los recorridos que se realizan en el algoritmo *B* son parecidos a viajar por una rama de un árbol centrado en el nodo origen.

En el algoritmo *B,* cada nodo que recibe el mensaje de búsqueda sólo puede seleccionar a uno de los nodos que se encuentran en su vecindario con el objetivo de disminuir el número de mensajes de búsqueda diseminados por la red. También el algoritmo *B* tiene la restricción de no poder reenviar el mensaje a algún nodo que ya lo haya recibido previamente, eso provoca que la propagación se vea como aparece en la *Figura 3.5*.

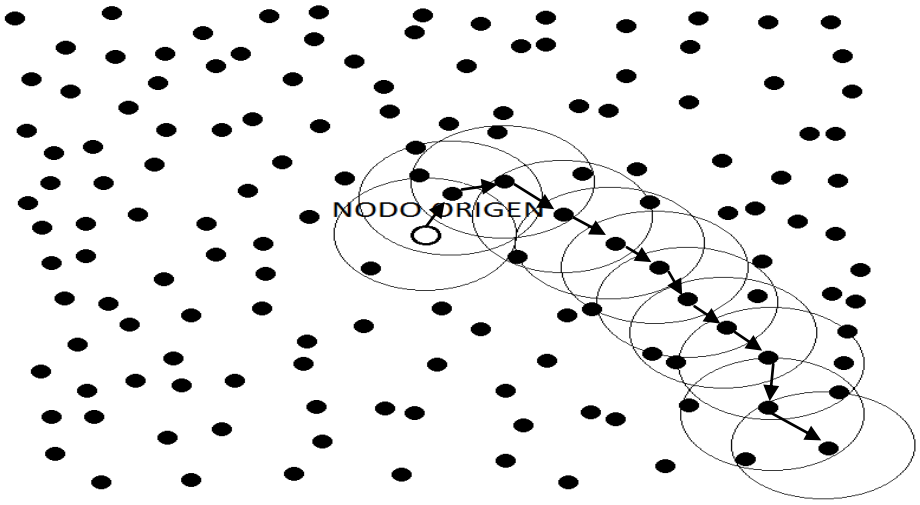


FIGURA 3.5 EJEMPLO DE UNA RUTA DE PROPAGACIÓN DEL MENSAJE

Durante las simulaciones encontramos un problema, debido a que la dirección que toma la propagación del mensaje es totalmente aleatoria, no se puede controlar a qué nodos se les envía el mensaje de búsqueda. Alguno de estos recorridos puede generar un cerco alrededor del nodo origen que posteriormente haga que el nodo destino ya no pueda enviar más mensajes para encontrar al destino, más allá del cerco. Este problema está ilustrado en la *Figura 3.6*.

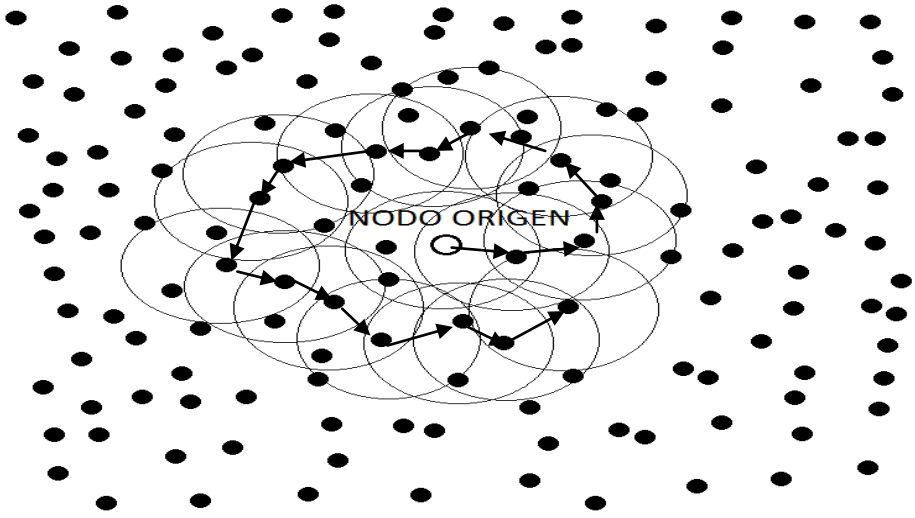


FIGURA 3.6 CERCO AL NODO ORIGEN

En los casos donde después de realizar toda la ejecución del algoritmo *B* no se encuentra al nodo destino, lo que se hace es iniciar una *inundación ciega*. Eso asegurará encontrar el nodo destino pero generará una cantidad mayor de señalización en la red.

PSEUDOCÓDIGO ALGORITMO B

- **IF**(origen tiene vecinos por enviar **AND** no ha recibido respuesta **AND** Tiempo evento = X)
  Origen envía mensaje a vecino:
  IF (nodo != destino **AND** nodo tenga vecinos para enviar) **THEN**
  registrar remitente
  seleccionar aleatoriamente un vecino no visitado previamente
  reenviar
  IF (nodo = destino) **THEN**
  registrar remitente
  contestar con **unicast** a origen
- **IF**(origen no tiene vecinos por enviar **AND** no ha recibido respuesta **AND** Tiempo evento = X)
  Origen inicia **flooding**

3.4 ALGORITMO B*

Durante el desarrollo y la simulación del algoritmo *B* nos dimos cuenta de dos cosas; cuando el nodo destino fue hallado, la cantidad de mensajes enviados en la red fue pequeña, pero cuando no se encontró el nodo destino y se tuvo que iniciar una *inundación ciega* el número de mensajes enviados se incrementaba mucho. La principal razón por la cual el algoritmo *B* no encontraba al nodo destino es que se generaban los problemas ilustrados en la *Figura 3.6*, donde se creaba un cerco alrededor del nodo origen que le impedía seguir enviando mensajes de búsqueda a través de la red. Pensando en cómo solucionar ese problema, surgió la idea del algoritmo *B*.*

El algoritmo *B** es una variación del algoritmo *B* que cambia principalmente una característica del algoritmo *B*. Cuando los nodos vecinos reciban el mensaje de búsqueda, en vez de sólo seleccionar un nodo de su vecindario para disminuir el número de mensajes de búsqueda que se diseminan en la red, se seleccionarán dos nodos de su vecindario que no hayan recibo el mensaje de búsqueda previamente. El objetivo de este cambio es que el alcance de la búsqueda en la red sea mucho mayor, la propagación del mensaje se realice más rápido y se trate de evitar el problema del cerco alrededor del nodo origen. Las modificaciones que se realizaron al algoritmo original son las siguientes.

- Si el nodo vecino no es el destino, éste a su vez reenvía el mensaje a dos de los nodos que se encuentran en su vecindario (seleccionándolo de manera aleatoria), y que no hayan sido seleccionados previamente.

Si después de un retardo Δt no se recibió el mensaje de respuesta por parte del nodo destino, el nodo origen envía otro mensaje dirigido a otros dos nodos de su vecindario a los que no les haya enviado un mensaje previamente.

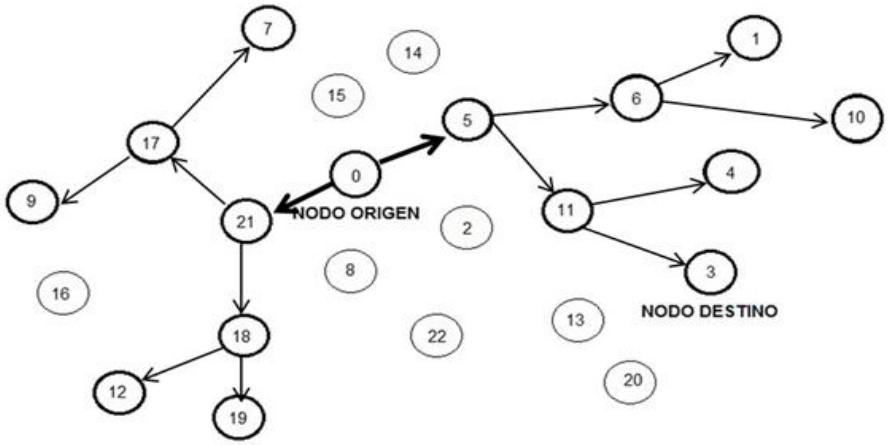


FIGURA 3.7 PROPAGACIÓN UTILIZANDO DOS VECINOS

Al irse seleccionando dos vecinos de forma aleatoria en vez de uno en el algoritmo *B**, y al tener también la restricción de no entregar el mensaje a algún nodo que haya sido visitado previamente, la propagación que genera abarca un área más grande en la red, como se muestra en la *Figura 3.7*, incrementando la posibilidad de encontrar el nodo destino.

En la *Figura 3.8*, podemos ver que el comportamiento del algoritmo *B** se asemeja mucho al de una *inundación ciega*, en el algoritmo *B** la diseminación de los mensajes alcanza a todos los nodos de la red.

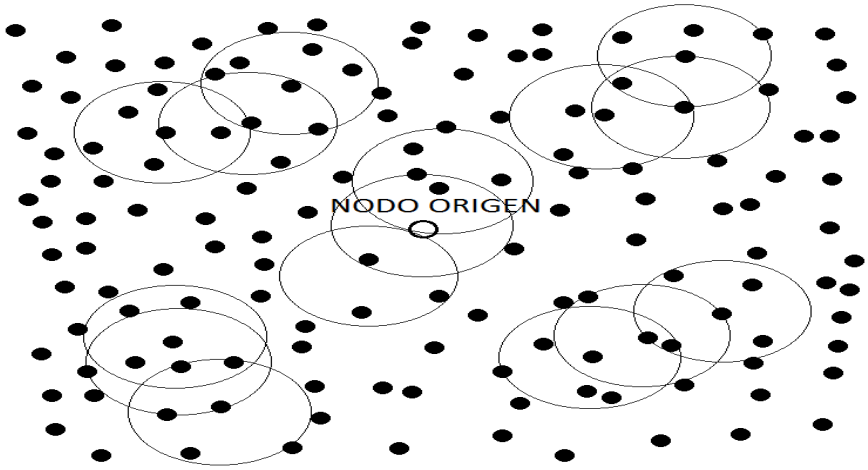


FIGURA 3.8 DISEMINACIÓN PARECIDA A INUNDACIÓN CIEGA

PSEUDOCÓDIGO ALGORITMO B*

IF (origen tiene vecinos por enviar **AND** no ha recibido respuesta **AND** Tiempo evento = X)
Origen envía mensaje a dos vecinos:
IF (nodo != destino **AND** nodo tenga vecinos para enviar) **THEN**
registrar remitente
seleccionar aleatoriamente dos vecinos no visitados previamente
reenviar
IF (nodo = destino) **THEN**
registrar remitente
contestar con **unicast** a origen
IF (origen no tiene vecinos por enviar **AND** no ha recibido respuesta **AND** Tiempo evento = X)
Origen inicia **flooding**

Como el algoritmo B* utiliza dos nodos vecinos en vez de uno, el alcance de la búsqueda en la red fue mucho mayor, la propagación del mensaje fue más rápida que los algoritmos previos y en ninguna de las simulaciones realizadas se presentó el problema del cerco alrededor del nodo origen. Aunque su desempeño fue parecido al de la inundación ciega, tiene la desventaja de no poder asegurar que siempre entregará el mensaje al nodo destino.

CAPÍTULO IV EVALUACIÓN DE DESEMPEÑO DE LOS ALGORITMOS

En este capítulo se iniciará describiendo de manera breve las modificaciones realizadas al simulador de redes *NS2* **[6]**. Se describirán los resultados obtenidos del simulador sobre el funcionamiento de los cuatro algoritmos propuestos en el capítulo anterior y al final se mostrará una gráfica que compare los resultados obtenidos de su desempeño en cuestión de señalización y tiempos de búsqueda.

4.1 CAMBIOS AL SIMULADOR NS2

Antes de comenzar a describir los resultados de los algoritmos que componen este trabajo es importante explicar algunas modificaciones que fueron necesarias realizar al simulador de redes *NS2*. El simulador *NS2* fue diseñado bajo el paradigma de programación orientado a objetos. Para poder realizar algún cambio al comportamiento de éste, es necesario tener una buena idea de cómo están íntimamente relacionadas las clases que los componen. Por ejemplo en *NS2* interactúan dos lenguajes de programación diferentes; por un lado tenemos a *C++* en el cual está programado el comportamiento interno del simulador y *TCL*, que es un lenguaje de scripting en el que los usuarios pueden ocupar las características del simulador.

Una de las ideas más importantes en uno de nuestros algoritmos es que los nodos tengan la capacidad de retener un paquete durante un determinado tiempo, pero que también puedan prescindir de dicho retardo durante la misma simulación. Para eso tuvimos que investigar qué parte de la estructura del nodo debíamos modificar para poder insertar los retardos.

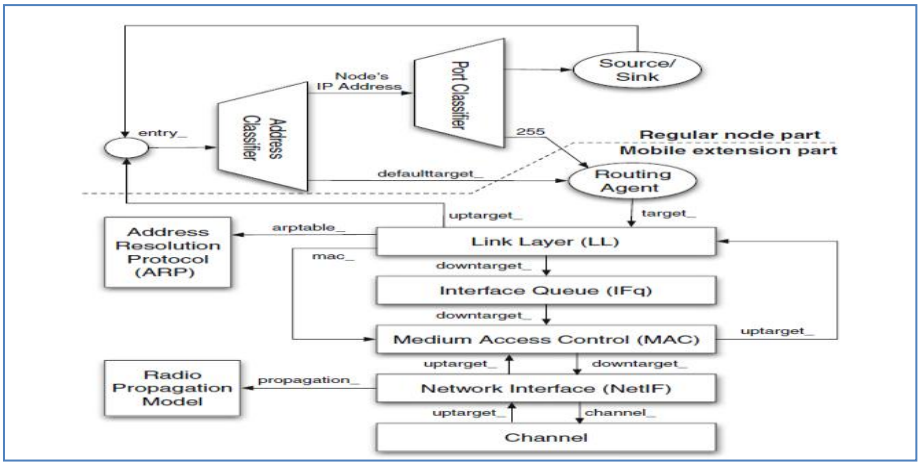


FIGURA 4.0 ESTRUCTURA INTERNA DE UN NODO INALÁMBRICO EN NS2. IMAGEN TOMADA TEERAWAT ISSARIYAKUL'S PERSONAL HOME PAGE [41]

Internamente los nodos inalámbricos tienen una estructura como la que se muestra en la *Figura 4.0*. La parte del nodo inalámbrico que modificamos fue la clase *Link Layer (LL)*. Esta clase hereda características de una clase llamada *Link Delay* como el método *recv,* vital para poder insertar el retardo porque interactúa con dos métodos que manejan la información y los tiempos internamente en el nodo; *sendDown* y *sendUp*. Si quisiéramos hacer lo mismo pero en una red alambrada, la clase que tendríamos que modificar es *Link Delay*.

Una vez identificado el lugar en donde se deben realizar los cambios en *C++*, nos encontramos con un problema, si insertamos el retardo en la clase *LL* se modifica el comportamiento general de todos los nodos y tendrán que esperar un retardo antes de poder transmitir aún cuando se trata de datos, aunque no siempre sea necesario. Entonces ¿Cómo le decimos al simulador cuándo utilizar el retardo y cuándo no? Desde el lado de *TCL* no hay ninguna opción que nos permita interactuar directamente con *C++*. Por eso, la primera idea que intentamos llevar a cabo fue clonar la clase *LL* para que el nodo tuviera la posibilidad de ocupar más de un tipo de capa de enlace, pero la configuración de un nodo inalámbrico se realiza al inicio de la simulación, y no se pude cambiar en cualquier momento durante la simulación(o se ocupa una u otra pero no existe la posibilidad en la arquitectura actual para indicar más de una).

```
$ns node-config -adhocRouting AODV \
     -llType LL \
     -macType Mac/802_11 \
     -ifqType Queue/DropTail/PriQueue \
     -ifqLen 50 \
     -antType Antenna/OmniAntenna \
     -propType Propagation/TwoRayGround \
     -phyType Phy/WirelessPhy \
     -channelType Channel/WirelessChannel \
     -topoInstance $topo \
     -agentTrace ON \
     -routerTrace ON \
     -macTrace OFF \
     -movementTrace OFF
```

FIGURA 4.1 EJEMPLO DE UNA CONFIGURACIÓN TÍPICA DE NODOS INALÁMBRICOS

Como se muestra en la *Figura 4.1*, el script de configuración de los nodos inalámbricos es una interfaz bastante bien definida pero nada flexible. Sería ideal tener la posibilidad de indicar más de un tipo de *LinkLayer* pero tratar de llevar esa idea a cabo nos llevaría mucho más tiempo y modificar tanto el código del simulador que tendríamos una nueva arquitectura. Aunque son pocos, existen

algunos artículos en donde han intentado hacer más flexible al simulador *NS2* creando más de una interfaz para cada nodo.

Crear una nueva arquitectura del simulador va más allá de la idea original de esta tesis, así que para poder hacer las modificaciones necesarias debemos encontrar un parámetro utilizado en los scripts de *TCL* que en algún momento, incluso de forma indirecta, llegue a tener contacto con la clase *LL*. Para eso fue necesario ver todas las dependencias que tiene la clase *ll.cc*. La única manera que nos permitió manipular información de forma indirecta desde los scripts de *TCL* hasta la clase *LL* fue crear una instancia del *header cmn* dentro de su método llamado *sendUP*. Al hacer eso, podemos utilizar información del mensaje que ha sido configurada desde el script de *TCL* como el *uid*, *ptype*, *direction* o *size*, y así aplicar retardo o quitarlo antes de que se manipule el mensaje internamente en los nodos.

```
237
238 void LL::sendUp(Packet* p)
239 {
240         hdr_cmn *ch2 = HDR_CMN(p); //este es mi codigo
241         double dExtra = 0.15;
242         Scheduler& s = Scheduler::instance();
243         if (hdr_cmn::access(p)->error() > 0)
244                 drop(p);
245         else
246                 if(ch2->size() != 200 )
247                   {
248                       dExtra = 0;
249                   }
250                 cerr << "Valoractual de  uid " << ch2->uid() << endl;
251                 cerr << "Valoractual de  ptype " << ch2->ptype() << endl;
252                 cerr << "Valoractual de  size " << ch2->size() << endl;
253                 cerr << "Valoractual de  direction " << ch2->direction() << endl;
254                 s.schedule(uptarget_, p, delay_ + dExtra );
255 }
256
257 inline void LL::hdr_dst(Packet *, int)
258 {}
```

C++ Ancho de la tabulación: 8

FIGURA 4.2 MÉTODO SENUP DE LA CLASE LL

La *Figura 4.2* muestra las modificaciones realizadas en el archivo *ll.c* del simulador *NS2* para manipular los parámetros *uid*, *ptype*, *direction* o *size*. Una vez hechas dichas modificaciones al simulador, el siguiente paso fue crear una configuración que ocuparíamos para poner en práctica todos los algoritmos que programamos.

4.2 ESCENARIO DE LA RED

Para todos los algoritmos que programamos utilizamos la misma configuración para la red inalámbrica; los escenarios se componen de un conjunto de 400 nodos que están distribuidos de forma aleatoria.

TABLA 4.0 CARACTERÍSTICAS DE LA RED

Número de nodos	400
Dimensión de la red	X=3000 m, Y=3000 m
Rango de Tx de nodos	250 m
Rango de sensado	550 m
Posición de los nodos	Aleatorio

4.3 ALGORITMO A

Para analizar el desempeño de este algoritmo *A* y verificar si al realizar las modificaciones mejoramos el desempeño del algoritmo con respecto al algoritmo de *inundación ciega*, se realizaron veinte simulaciones para cada una de las cuales seleccionamos un nodo de forma aleatoria como nodo destino, mientras que el nodo origen siempre se escogió entre alguno de los nodos localizados en el centro de la red. Contamos la cantidad de mensajes que fueron necesarios para encontrar el nodo destino, también medimos el tiempo que el algoritmo *A* tardó en encontrar al nodo destino y lo comparamos con el algoritmo de *inundación ciega*, los resultados son los siguientes:

ID NODO DESTINO	NODOS	ENCONTRADO	MENSAJES	TIEMPO (S)	INUNDACIÓN CIEGA MENSAJES	INUNDACIÓN CIEGA TIEMPO (S)
359	1	SÍ	589	1.39	399	0.109
395	2	SÍ	119	0.47	399	0.05
7	3	SÍ	423	1.07	399	0.074
296	4	SÍ	331	0.916	399	0.053
63	5	SÍ	792	2.294	399	0.114
306	6	SÍ	119	0.476	399	0.03
11	7	SÍ	255	0.764	399	0.039
380	8	SÍ	325	0.928	399	0.109
31	9	SÍ	523	1.22	399	0.065
52	10	SÍ	190	0.609	399	0.046
367	11	SÍ	119	0.476	399	0.051
143	12	SÍ	525	1.223	399	0.065
50	13	SÍ	435	1.085	399	0.061
123	14	SÍ	325	0.921	399	0.031
225	15	SÍ	321	0.925	399	0.065
65	16	SÍ	17	0.152	399	0.003
244	17	SÍ	185	0.623	399	0.075
181	18	SÍ	17	0.152	399	0.003
328	19	SÍ	119	0.476	399	0.051
21	20	SÍ	269	0.782	399	0.072

El promedio de mensajes enviados durante las veinte simulaciones fue de 299.9 mensajes, lo que es bastante bueno en comparación del promedio de la *inundación ciega* que es de 399, hay una ganancia del *25%*.

PROMEDIO MENSAJES ALGORITMO A	PROMEDIO TIEMPO ALGORITMO A (S)
299.9	0.8476

Al contrario del caso de los mensajes enviados, el promedio del tiempo que tarda este algoritmo en encontrar al nodo destino es superior en comparación a la *inundación ciega* debido a los retardos insertados.

PROMEDIO MENSAJES INUNDACIÓN	PROMEDIO TIEMPO INUNDACIÓN (S)
399	0.0583

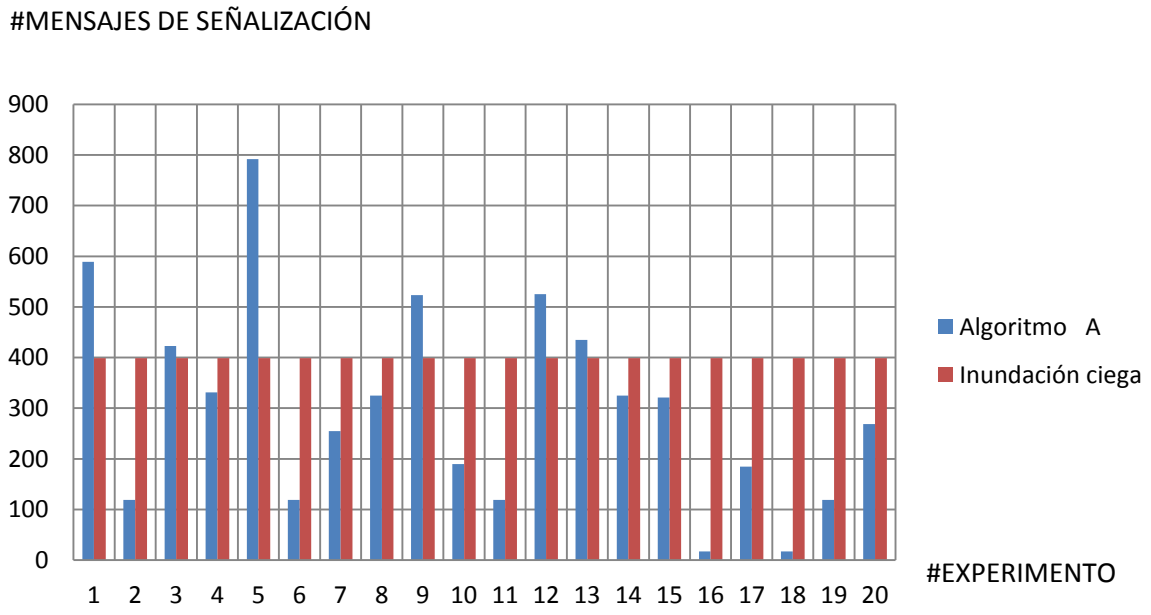


FIGURA 4.3 COMPARACIÓN DEL NÚMERO DE MENSAJES ENVIADOS POR INUNDACIÓN CIEGA Y ALGORITMO *A*

La disminución en el número de mensajes de señalización por parte del algoritmo *A* es entendible si tomamos en cuenta que se pueden dar casos en los que los nodos destino se encuentren a una distancia corta con respecto al nodo origen, como fue en la simulación número dieciséis o la dieciocho que se muestra en la *Figura 4.3*. Aunque también hay casos como el de la simulación número uno o cinco en donde la cantidad de mensajes enviados superan por mucho al algoritmo de *inundación ciega*. En general el promedio de los mensajes enviado por este algoritmo *A* es favorable.

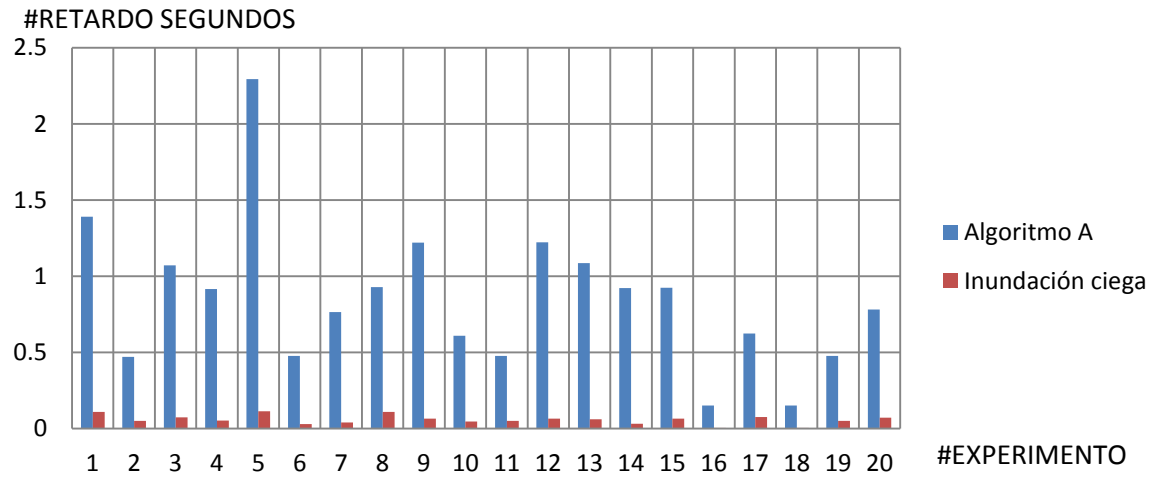


FIGURA 4.4 COMPARACIÓN DEL RETARDO DE INUNDACIÓN CIEGA Y ALGORITMO *A*

Como podemos ver en la *Figura 4.4*, el tiempo que tomó cada simulación para encontrar al nodo destino es mayor a la del algoritmo de *inundación ciega*, y esto tiene que ver con el tiempo de retardo que cada nodo aplica antes de reenviar el mensaje de la primera inundación (expansión).

4.4 ALGORITMO A*

Para el análisis de este algoritmo también se realizaron veinte simulaciones y se utilizaron los mismos nodos destino del experimento anterior. Contamos la cantidad de mensajes necesarios para encontrar el nodo destino, el tiempo que se tardó en encontrar al nodo destino y lo comparamos con el algoritmo de *inundación ciega*. Los resultados fueron los siguientes:

ID NODO DESTINO	NODOS	ENCONTRADO	MENSAJES	TIEMPO (S)	INUNDACIÓN CIEGA MENSAJES	INUNDACIÓN CIEGA TIEMPO (S)
359	1	SÍ	408	1.39	399	0.109
395	2	SÍ	119	0.47	399	0.05
7	3	SÍ	406	1.07	399	0.074
296	4	SÍ	331	0.916	399	0.053
63	5	SÍ	414	2.294	399	0.114
306	6	SÍ	119	0.476	399	0.03
11	7	SÍ	255	0.764	399	0.039
380	8	SÍ	325	0.928	399	0.109
31	9	SÍ	407	1.22	399	0.065
52	10	SÍ	190	0.609	399	0.046
367	11	SÍ	119	0.476	399	0.051
143	12	SÍ	407	1.223	399	0.065
50	13	SÍ	406	1.085	399	0.061
123	14	SÍ	325	0.921	399	0.031
225	15	SÍ	321	0.925	399	0.065
65	16	SÍ	17	0.152	399	0.003
244	17	SÍ	185	0.623	399	0.075
181	18	SÍ	17	0.152	399	0.003
328	19	SÍ	119	0.476	399	0.051
21	20	SÍ	269	0.782	399	0.072

En todas las simulaciones realizadas se encontró al nodo destino, al compararlo con el desempeño del algoritmo *A* podemos apreciar que el promedio de mensajes enviados fue menor pero el tiempo en encontrar cada nodo fue el mismo, así que el promedio es idéntico.

PROMEDIO MENSAJES ALGORITMO A*	PROMEDIO TIEMPO ALGORITMO A* (S)
257.95	0.8476

Al compararlo con el algoritmo de *inundación ciega* como se muestra en las *Figuras 4.5* y *4.6*, vemos que la única mejora es la reducción en cantidad de mensajes enviados, el promedio de tiempo que tarda en encontrar a los nodos destino (tiempo *end-to-end*) sigue siendo mucho mejor en *inundación ciega*.

PROMEDIO MENSAJES INUNDACIÓN	PROMEDIO TIEMPO INUNDACIÓN (S)
399	0.0583

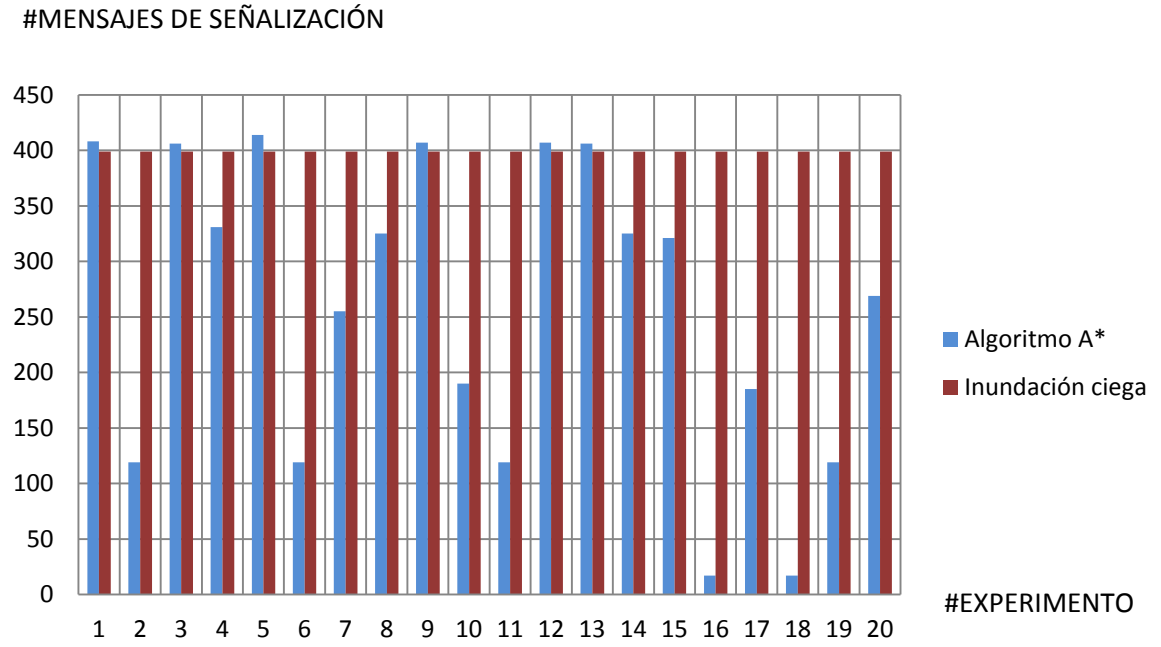


FIGURA 4.5 COMPARACIÓN DEL NÚMERO DE MENSAJES ENVIADOS POR INUNDACIÓN CIEGA Y ALGORITMO *A

A diferencia del algoritmo *A*, para encontrar los nodos más lejanos del nodo origen en el algoritmo *A**, el número de mensajes enviados fue casi idéntico al que se requieren en *inundación ciega*. En los experimentos uno, tres, cinco, nueve, doce y trece de la *figura 4.5* es donde se mejora la señalización en la red.

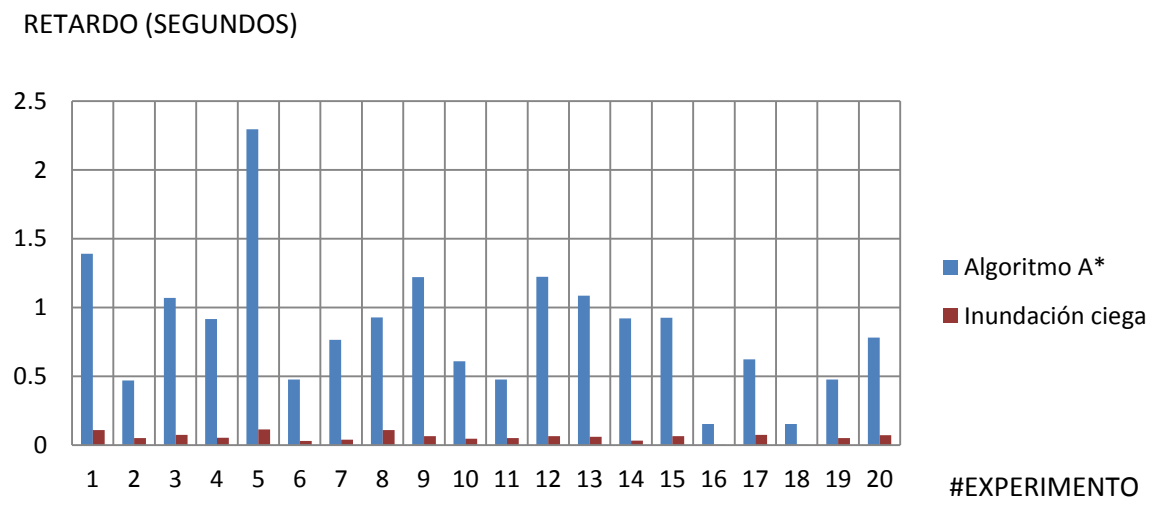


FIGURA 4.6 COMPARACIÓN DEL RETARDO DE INUNDACIÓN CIEGA Y ALGORITMO *A

Con lo que respecta al tiempo en encontrar los nodos, el comportamiento del algoritmo *A** es idéntico al del primer algoritmo, por lo que *inundación ciega* es mejor en este aspecto.

4.5 ALGORITMO B

En el caso del algoritmo *B* realizamos un total de veinte simulaciones y seleccionamos el mismo nodo destino para cada una de ellas, debido a que en este algoritmo las rutas generadas por cada simulación para encontrar el mismo nodo destino son diferentes. Contamos la cantidad de mensajes totales para encontrar el nodo destino y el tiempo que el algoritmo *B* tardó en encontrarlo y lo comparamos con el algoritmo de *inundación ciega*, los resultados fueron los siguientes:

ID NODO DESTINO	RUTAS	ENCONTRADO 1° VUELTA	ENCONTRADO 2° VUELTA	MENSAJES + INUNDACIÓN	TIEMPO + INUNDACIÓN	INUNDACIÓN CIEGA MENSAJES	INUNDACIÓN CIEGA TIEMPO (S)
244	1	NO	SÍ	530	2.8833	399	0.075
244	2	SÍ	SÍ	170	2.926	399	0.075
244	3	SÍ	SÍ	112	1.693	399	0.075
244	4	SÍ	SÍ	160	2.87	399	0.075
244	5	NO	SÍ	541	2.8833	399	0.075
244	6	NO	SÍ	520	2.8833	399	0.075
244	7	NO	SÍ	524	2.8833	399	0.075
244	8	SÍ	SÍ	155	2.065	399	0.075
244	9	NO	SÍ	515	2.8833	399	0.075
244	10	NO	SÍ	522	2.8833	399	0.075
244	11	NO	SÍ	593	2.8833	399	0.075
244	12	SÍ	SÍ	69	1.635	399	0.075
244	13	NO	SÍ	546	2.8833	399	0.075
244	14	SÍ	SÍ	61	1.67	399	0.075
244	15	SÍ	SÍ	56	0.134	399	0.075
244	16	NO	SÍ	522	2.8833	399	0.075
244	17	SÍ	SÍ	71	1.639	399	0.075
244	18	SÍ	SÍ	18	0.041	399	0.075
244	19	NO	SÍ	517	2.8833	399	0.075
244	20	SÍ	SÍ	55	2.02	399	0.075

En este caso no todas las simulaciones realizadas encontraron al nodo destino durante la ejecución del algoritmo *B*, en la mitad de los experimentos tuvimos que realizar posteriormente una *inundación ciega*, para encontrar el nodo destino. Aunque el promedio de mensajes enviados fue menor que en el caso de *inundación ciega* como se muestra en la *Figura 4.7*, su desempeño fue menos favorable que los dos algoritmos anteriores *A* y *A**.

PROMEDIO MENSAJES ALGORITMO B	PROMEDIO TIEMPO ALGORITMO B (S)
312.85	2.2763

Con respecto el promedio de tiempo que tarda en encontrar a los nodos destino (tiempo *end-to-end*), este algoritmo tarda más tiempo en comparación de los algoritmos *A*, *A** e *inundación ciega*.

PROMEDIO MENSAJES INUNDACIÓN	PROMEDIO TIEMPO INUNDACIÓN (S)
399	0.075

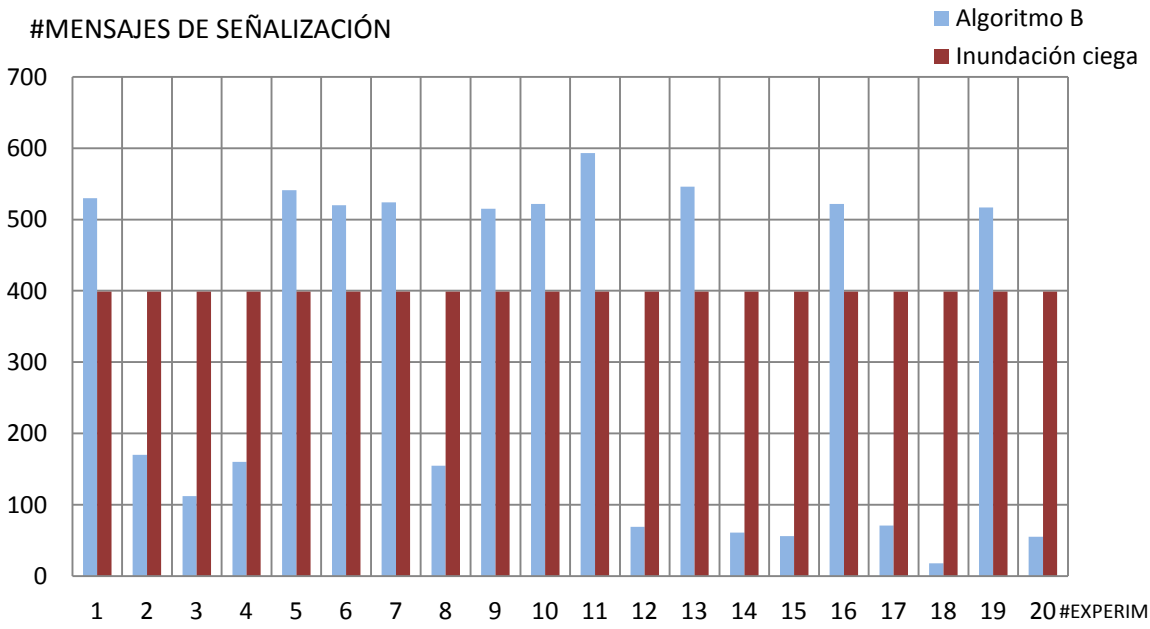


FIGURA 4.7 COMPARACIÓN DEL NÚMERO DE MENSAJES ENVIADOS POR INUNDACIÓN CIEGA Y ALGORITMO *B*

Aunque en la mitad de los casos no se encontró el nodo destino en este algoritmo y se tuvo que realizar una *inundación ciega*, lo cual aumenta considerablemente la cantidad de mensajes de señalización y el tiempo de búsqueda, el promedio general de los veinte experimentos es todavía menor que en el caso de un *inundación ciega*. Esto se debe a que en los casos donde sí se encontró el nodo destino la cantidad de mensajes enviados fue muy pequeña.

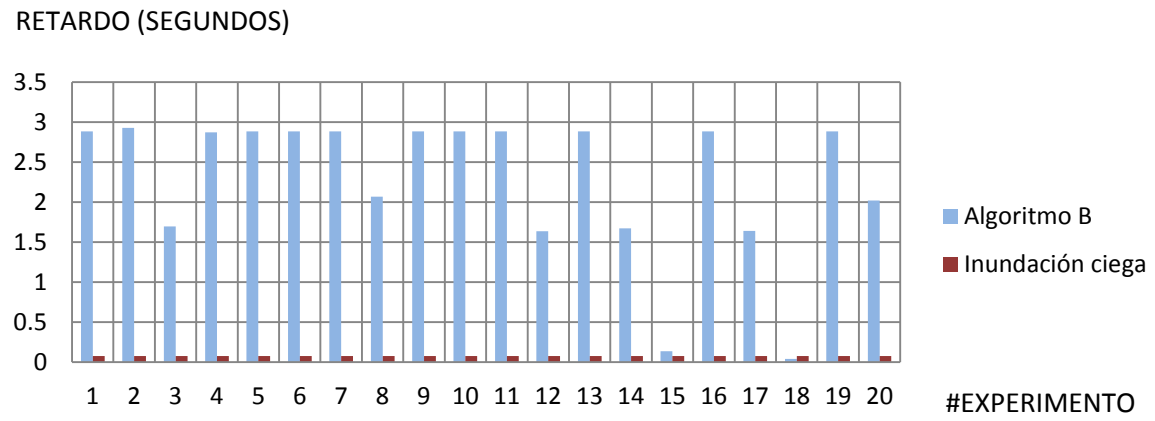


FIGURA 4.8 COMPARACIÓN DEL RETARDO DE INUNDACIÓN CIEGA Y ALGORITMO *B*

Como se muestra en la *Figura 4.8,* el desempeño de este algoritmo con respecto al tiempo que tarda en encontrar a los nodos destino (tiempo extremo a extremo) es inferior. Esto se debe a que forzosamente hay un lapso de tiempo necesario entre cada envío del nodo origen a sus vecinos y si ocurren casos en los que no se encuentra el nodo destino, se debe agregar el tiempo en que el *inundación ciega* tarda en encontrar a cada uno de esos nodos.

4.6 ALGORITMO B*

En el caso de este algoritmo también seleccionamos un nodo origen ubicado aproximadamente en el centro de la red, realizamos un total de veinte simulaciones y seleccionamos sólo un nodo destino como en el algoritmo *B*. Contamos la cantidad de mensajes totales para encontrar el nodo destino, el tiempo que le tomó al algoritmo *B* en encontrarlo y lo comparamos con el algoritmo de *inundación ciega*, los resultados fueron los siguientes:

ID NODO DESTINO	RUTAS	ENCONTRADO	MENSAJES	TIEMPO (S)	INUNDACIÓN CIEGA MENSAJES	INUNDACIÓN CIEGA TIEMPO (S)
244	1	SÍ	408	0.1841	399	0.075
244	2	SÍ	396	0.239	399	0.075
244	3	SÍ	334	0.15	399	0.075
244	4	SÍ	366	0.16	399	0.075
244	5	SÍ	391	0.042	399	0.075
244	6	SÍ	381	0.1616	399	0.075
244	7	SÍ	398	0.158	399	0.075
244	8	SÍ	392	0.079	399	0.075
244	9	SÍ	399	0.144	399	0.075
244	10	SÍ	398	0.1744	399	0.075
244	11	SÍ	396	0.109	399	0.075
244	12	SÍ	371	0.117	399	0.075
244	13	SÍ	400	0.126	399	0.075
244	14	SÍ	385	0.088	399	0.075
244	15	SÍ	397	0.148	399	0.075
244	16	SÍ	399	0.145	399	0.075
244	17	SÍ	387	0.061	399	0.075
244	18	SÍ	388	0.121	399	0.075
244	19	SÍ	392	0.084	399	0.075
244	20	SÍ	390	0.085	399	0.075

En este caso y a diferencia del algoritmo *B*, todas las simulaciones realizadas encontraron al nodo destino durante la ejecución del algoritmo *B**, aunque el promedio de mensajes enviados fue casi igual que los que ocuparía *inundación ciega*. Su desempeño fue el menos favorable de todos los algoritmos propuestos; *A*, *A** y *B*.

PROMEDIO MENSAJES ALGORITMO B*	PROMEDIO TIEMPO ALGORITMO B* (S)
388.4	0.128805

El tiempo promedio que tarda en encontrar a los nodos destino (tiempo *end-to-end*) en este algoritmo si bien es un poco mayor que en la *inundación ciega*, es mucho mejor que *A*, *A** y *B*.

PROMEDIO MENSAJES INUNDACIÓN	PROMEDIO TIEMPO INUNDACIÓN (S)
399	0.075

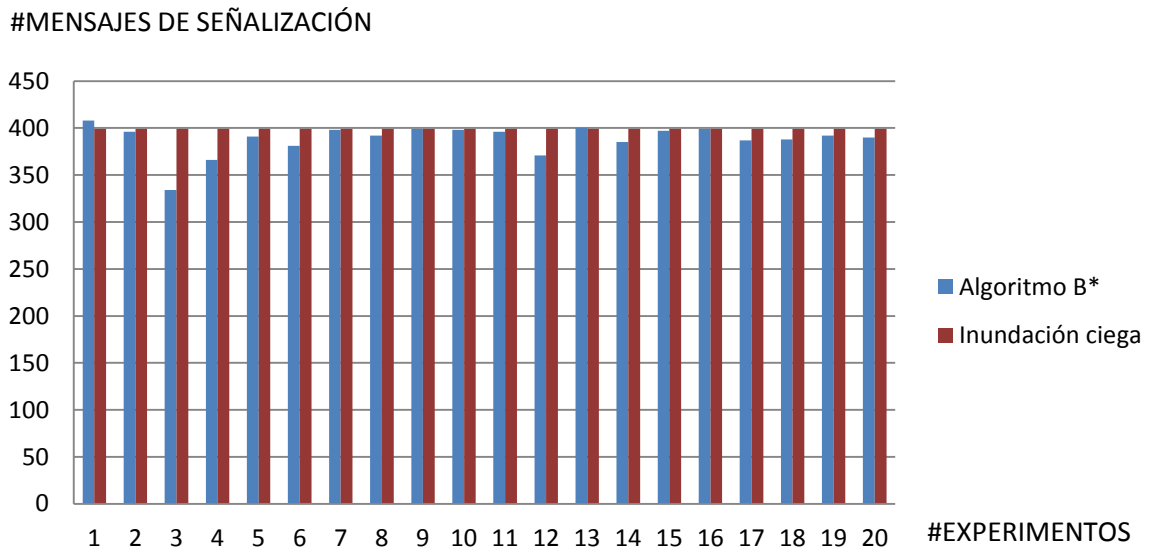


FIGURA 4.9 COMPARACIÓN DEL NÚMERO DE MENSAJES ENVIADOS POR INUNDACIÓN CIEGA Y ALGORITMO *B**

Como podemos visualizar en la *Figura 4.9*, la cantidad de mensajes enviados en cada simulación del algoritmo en *B** es similar al del *inundación ciega*. En todos los casos el nodo origen sólo tuvo que enviar un mensaje a uno de sus vecinos para alcanzar al nodo destino. Si recordamos, su comportamiento de propagación se asemeja bastante al algoritmo de *inundación ciega*.

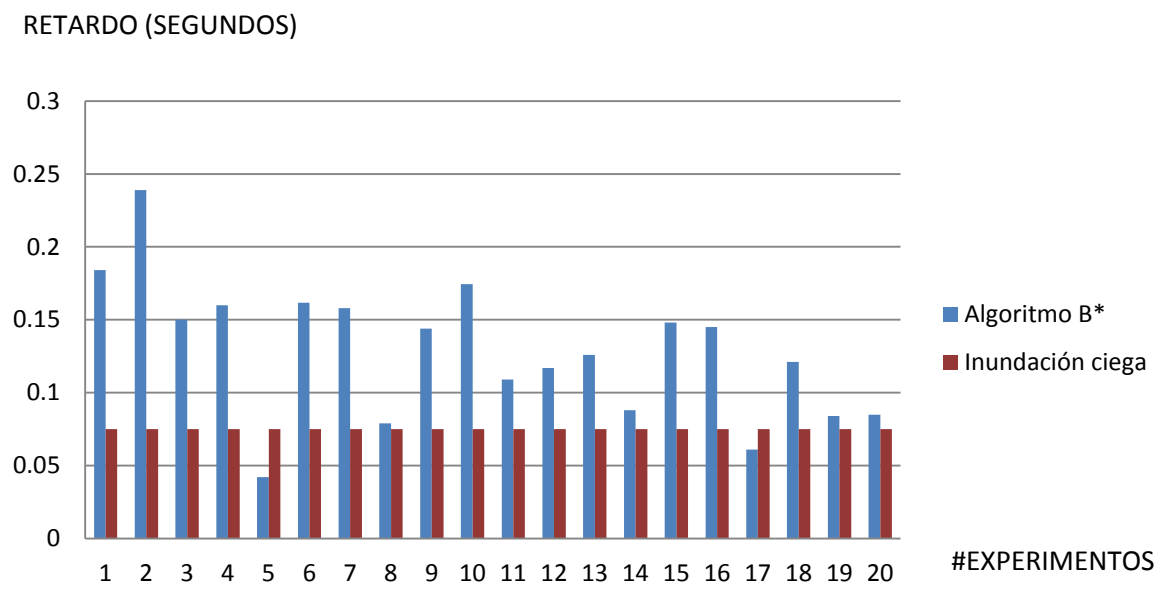


FIGURA 4.10 COMPARACIÓN DEL RETARDO DE INUNDACIÓN CIEGA Y ALGORITMO *B*

Como se muestra *en la Figura 4.10*, el tiempo promedio que tarda en encontrar a los nodos destino (tiempo extremo a extremo) este algoritmo es un poco mayor que el de *inundación ciega*, incluso en algunos casos lo mejora y supera por un gran margen a los algoritmos *A*, *A** y *B*. Esto se debe a que en todas las simulaciones el algoritmo encuentra al nodo destino con el primer mensaje enviado. No hay necesidad de esperar los Δt que ocurren en el algoritmo *B* cuando el nodo origen espera el mensaje de respuesta del nodo destino para detener la búsqueda y ya no enviar más mensajes de búsqueda a otro de sus vecinos.

Finalmente, a manera de resumen en las *Figuras 4.11* y *4.12* se muestra una gráfica comparativa entre los algoritmos *A*, *A**, *B*, *B** e *inundación ciega*. En la *Figura 4.11* podemos observar que el algoritmo *A** es el que obtuvo el mejor promedio de menos mensajes enviados a través de la red. Por otro lado en la *Figura 4.12* podemos ver que el algoritmo que obtuvo el menor tiempo promedio para encontrar a los nodos destino *(*tiempo extremo a extremo) fue el algoritmo de *inundación ciega*.

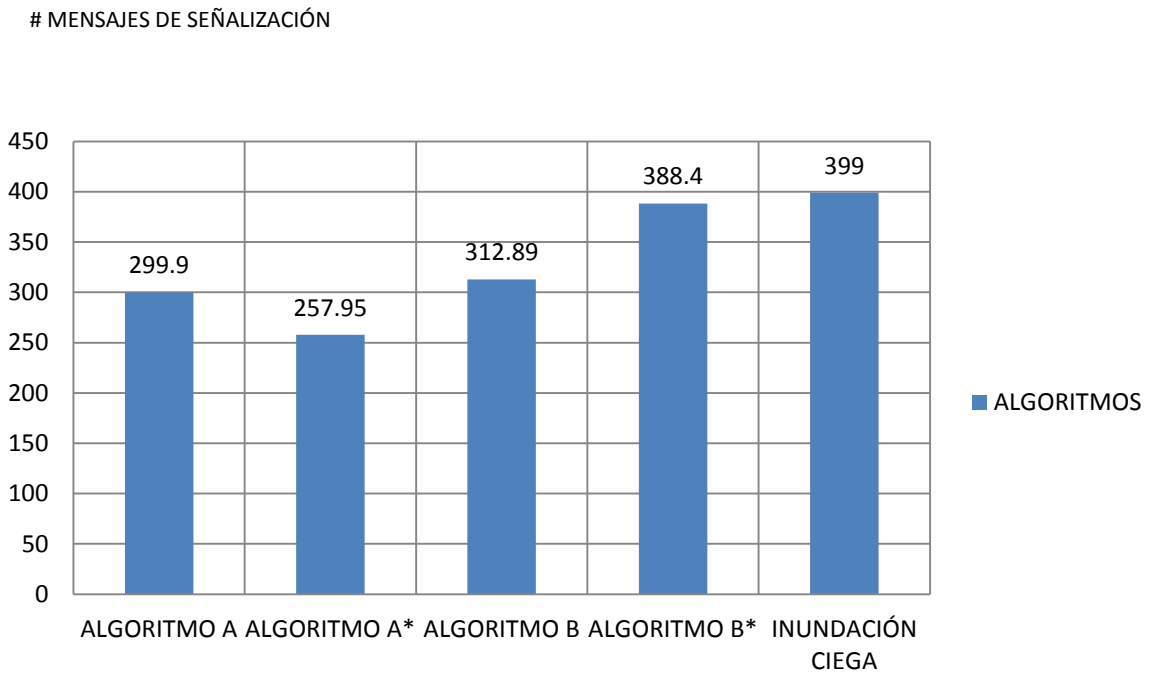


FIGURA 4.11 COMPARACIÓN DE PROMEDIO DE MENSAJES ENVIADOS POR LOS CUATRO ALGORITMOS

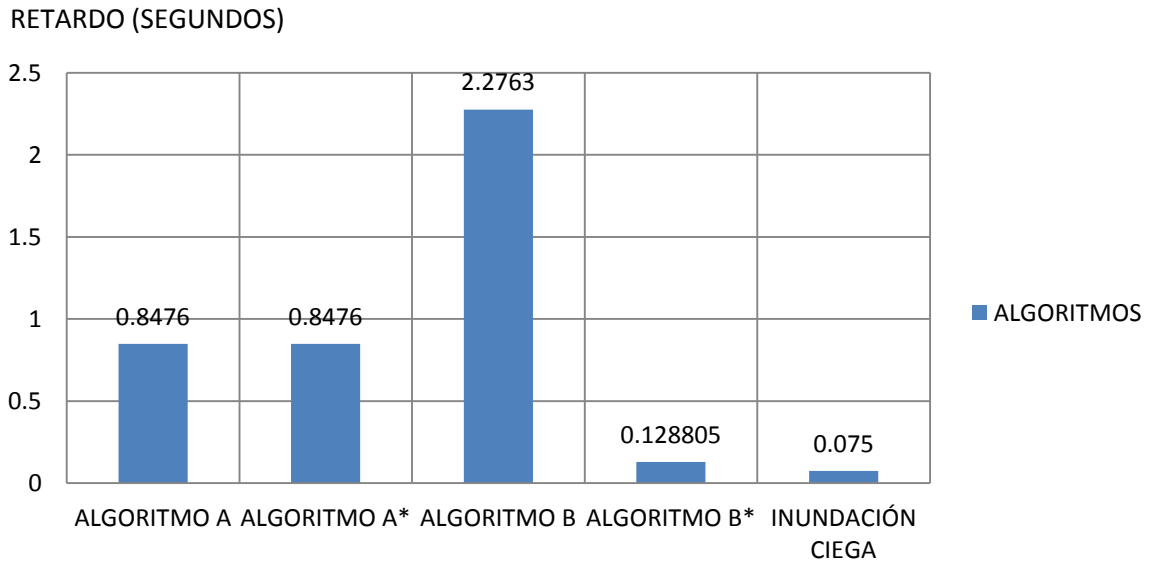


FIGURA 4.12 COMPARACIÓN DE PROMEDIO DE MENSAJES ENVIADOS POR LOS CUATRO ALGORITMOS

4.7 COMPARACIÓN DE NARD, FRESH Y ALGORITMO A*

También comparamos el desempeño del algoritmo *A**, que fue el que tuvo el mejor desempeño de los cuatro algoritmos que diseñamos, con los algoritmos *NARD* y *FRESH*, descritos en el capítulo dos. Para hacer esto utilizamos la información de las gráficas mostradas en **[4]** cuando el nodo destino se encuentra a una distancia de entre cinco y hasta diez saltos de distancia con respecto al nodo origen.

En la *Figura 4.13* podemos apreciar que cuando el valor *k* (número de saltos de distancia del nodo destino) fue constante, independientemente del número de saltos de distancia a los que se encontraba el nodo destino, *NARD* generó menos señalización que *FRESH* y que el algoritmo A*. Por otro lado en la *Figura 4.14* se muestra que cuando el valor de *n* (número de saltos de distancia del nodo origen) fue el que se mantuvo constante, el algoritmo *A** tuvo un mejor desempeño que *NARD* cuando el nodo destino se encontraba a cinco y seis saltos de distancia del nodo origen. En los demás casos *NARD* generó menos señalización que el algoritmo *A** aunque la diferencia no fue tan grande. En cualquiera de los escenarios *FRESH* generó más señalización que los otros dos algoritmos.

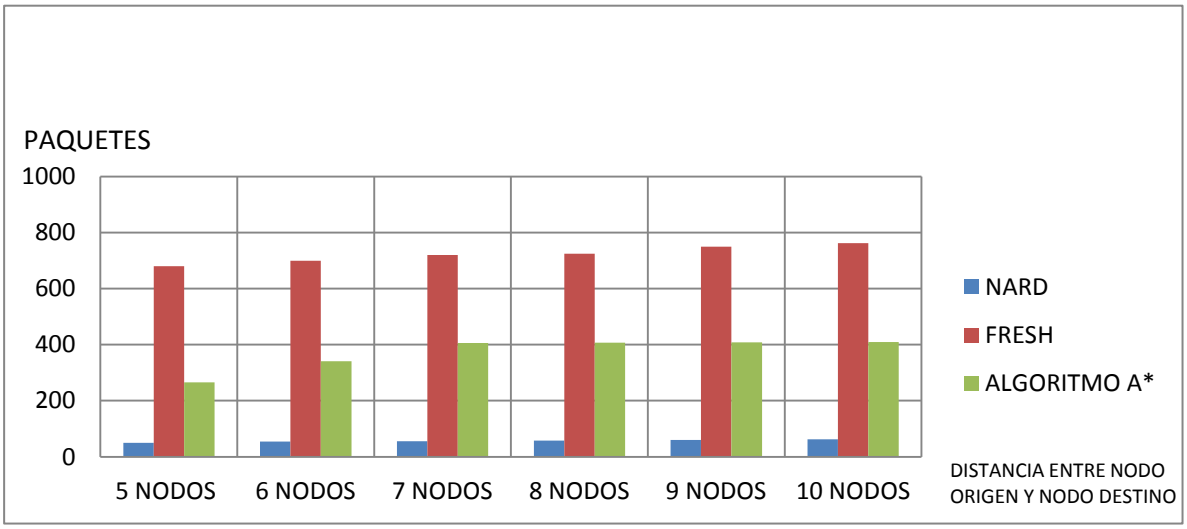


FIGURA 4.13 FIGURA COMPARACIÓN NARD (K CONSTANTE), FRESH Y ALGORITMO A*

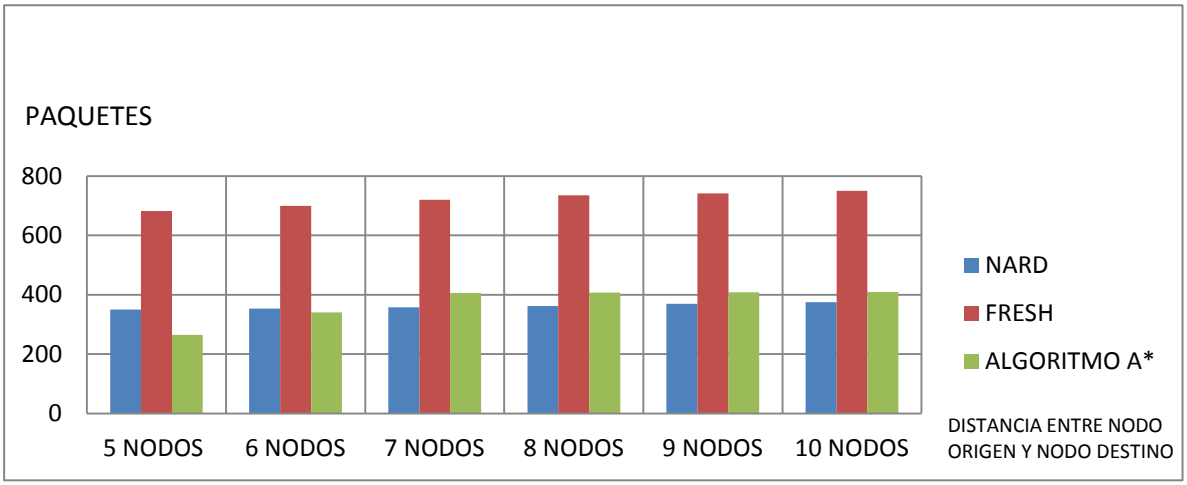


FIGURA 4.14 COMPARACIÓN NARD (N CONSTANTE), FRESH Y ALGORITMO A*

CAPÍTULO V CONCLUSIONES

La fase de descubrimiento de rutas en los protocolos de encaminamiento reactivos para redes *ad hoc* es un tema de gran interés porque, aunque hay distintas propuestas de mejora para la búsqueda de rutas en la red, la técnica de *inundación ciega* sigue siendo la más utilizada en la práctica. La ventaja de usar el algoritmo de *inundación ciega* es que nos provee de una solución simple y robusta para el descubrimiento de rutas, pero también implica un enorme costo en términos de ancho de banda y tiempo extremo a extremo. El gran número de paquetes que se generan al utilizar este tipo de encaminamiento tiene el inconveniente de que puede incrementar sensiblemente el retardo de los paquetes transportados por ella. Debido a esto, en este trabajo se presentaron cuatro algoritmos que buscan aminorar las desventajas del uso del algoritmo de *inundación ciega*.

Durante el desarrollo y explicación de los cuatro algoritmos que componen este trabajo de investigación podemos pensar que si clasificáramos los cuatro algoritmos de acuerdo a su desempeño en cuestión de mensajes de señalización enviados por la red, el tiempo que le lleva al nodo origen encontrar el nodo destino y los requerimientos del algoritmo, lo haríamos de la siguiente manera.

En último lugar se encuentra el Algoritmo *B**, ya que el número de mensajes enviados en promedio es muy similar al del caso del algoritmo de *inundación ciega*, apenas hay una ganancia del *2.65 %.* Aunque el desempeño de su tiempo promedio que tarda en encontrar a los nodos destino fue el mejor de los cuatro algoritmos y sólo un poco más tardado que la *inundación ciega*.

En tercer lugar se encuentra el Algoritmo *B* debido a la reducción del número de mensajes enviados en promedio en comparación con el algoritmo de *inundación ciega* que fue del *21.59 %.* El problema con este algoritmo fue el bajo desempeño del tiempo promedio que tarda en encontrar a los nodos destino (tiempo extremo a extremo), ya que de los cuatro algoritmos en promedio tuvo el mayor retardo al encontrar una ruta entre los nodos origen y destino.

El segundo lugar lo ocupa el Algoritmo *A* debido a que el número de mensajes enviados en promedio fue un *25 %* menor en comparación con el algoritmo de *inundación ciega* y su tiempo de descubrimiento de ruta entre los nodos origen y destino fue el segundo mejor. Este algoritmo también tiene la ventaja de que es bastante simple y robusto ya que no presenta muchas diferencias con el algoritmo de *inundación ciega*.

Finalmente, el algoritmo que tuvo de forma general el mejor desempeño de los cuatro fue el Algoritmo *A**. Aunque el Algoritmo *A* se comporta bastante bien, durante su desarrollo visualizamos que se podría mejorar con el concepto del límite en la red que describimos en el capítulo tres, llevándonos a la creación del

algoritmo *A**. Esa idea ayudó automáticamente a que el algoritmo *A** pudiera reducir la señalización en la red aún más, en promedio se redujo un *35 %* en comparación con el algoritmo de *inundación ciega* y su tiempo de retardo entre los nodos origen y destino fue el segundo mejor junto con el algoritmo *A*. La única desventaja que podríamos encontrar en el algoritmo *A** es que, para poner en práctica el concepto del límite de propagación de la inundación inicial en la red, hay que tener un conocimiento previo del tamaño de la red.

En el caso de los algoritmos *A* y *A** son algoritmos sencillos y robustos porque sólo incorporan ligeras modificaciones con respecto al algoritmo de *inundación ciega*.

Con respecto a la comparación entre los algoritmos *NARD*, *FRESH* y *A** podemos concluir que *NARD* fue el que mostró el mejor desempeño de los tres algoritmos excepto cuando el valor de *n* se mantuvo constante. En los casos donde el nodo destino se encontraba a cinco y seis saltos de distancia del nodo origen, algoritmo *A** generó menos señalización que *NARD*. Esto se pudo dar porque el valor de *n* controla la primera inundación en *NARD* y probablemente el valor utilizado para *n* era muy grande en los casos donde el nodo destino estaba a tan solo cinco o seis saltos de distancia. Ese hecho pudo provocar que se generara más señalización de lo necesario. De forma general podemos decir que el algoritmo *NARD* fue el que tuvo mejor desempeño, después *A** y al último lugar *FRESH*.

Al analizar las pruebas realizadas a los algoritmos propuestos en esa tesis podemos observar que su tiempo de descubrimiento de ruta de extremo a extremo fue mayor que el utilizado por el algoritmo de *inundación ciega.* Por el otro lado*,* la cantidad de mensajes enviados fue menor, en particular el algoritmo A* tuvo una disminución de un 35% con respecto al algortimo de inundacón ciega. Esto es muy importante porque mientras menos mensajes envíe un nodo, mayor será el ahorro de energía y dismiuira la tormenta de mensajes de difusión durante el proceso e busqueda de nodos. Es posible que en escenarios donde muchos nodos realicen procesos de busqueda al mismo tiempo se logre, inclusive, reducir el tiempo de descubimiento de ruta de extremo a extremo que si se utilizara inundación ciega.

BIBLIOGRAFÍA Y REFERENCIAS

[1] Taek Jin Kwon, Mario Gerla, “Efficient Flooding in Ad hoc Networks using On-Demand (Passive) Cluster Formation”, ACM SIGCOMM Computer Communication Review ,2002.

[2] B. Mans and N. Shrestha, “Performance Evaluation of Approximation Algorithms for Multipoint Relay Selection” in Proceedings of the 3rd Annual Mediterranean Ad Hoc Networking Workshop, 2004.

[3] Dubois-Ferrie, H., Grossglauser, M., & Vetterli, M. “Age matters: efficient route discovery in mo ile ad hoc networks using encounter ages” in Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, 2003.

[4] Gomez, J., Rangel, V., Lopez-Guerrero, M., Pascoe, M. “NARD: Neigh or-assisted route discovery in MANETs”. Springer Science+Business Media, LLC, 2011.

[5] Surendra H , Raut Hemant ,P. Am ulgekar “ Proactive and Reactive Routing Protocols in Multi hop Mo ile Adhoc Network” International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, pp:152 -157, ISSN: 2277-1286, abril 2013.

[6] Issariyakul, T., Hossain, E. Introduction to Network Simulator NS2. Springer, 2nd ed. 2012, 512 p.

[7] Perkins, Belding-Royer: Ad hoc On-Demand Distance Vector (AODV) Routing, 2003.

[8] David B. Johnson, David A. Maltz, and Josh Broch, "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks", in Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5, pp. 139-172, Addison-Wesley, 2001.

[9] The Network Simulator – NS-2; Disponible en la dirección web: http://www.isi.edu/nsnam/ns/.

[10] Petteri Kuosmanen: Classification of Ad Hoc Routing Protocols http://keskus.hut.fi/opetus/s38030/k02/papers/12-Petteri.pdf

[11] Ad-Hoc Networks-Technologies and protocols. Springer, 2005.

[12] M., Marrone L., Diaz J., Barbieri A. Comparación de protocolos de encaminamiento en redes ad hoc. http://www.linti.info.unlp. edu.ar

[13] On the broadcast storm problem in ad hoc wireless networks

OK Tonguz, N Wisitpongphan, JS Parikh, F Bai, P Mudalige, VK Sadekar Broadband Communications, Networks and Systems. BROADNETS, 3rd International Conference on, 2006.

[14] John Jubin and Janet D. Tornow, The DARPA Packet Radio Network Protocols, Proceedings of the IEEE, 75:21-32, enero 1987.

[15] L Hogie, P Bouvry, F Guinand. An overview of MANET simulation. Electronic notes in theoretical computer science 150, 81-101.

[16] Marc Greis; “Tutorial for Network Simulator NS”; enero 2014; Disponible en la dirección web: http://www.isi.edu/nsnam/ns/tutorial/index.html.

[17] Jayasuriya, Aruna; Hidden vs exposed terminal problem in ad hoc networks. Australian Telecommunication Networks & Applications Conference (ATNAC) 2004, pp. 52-59.

[18] Harras K. A., Almeroth K. C., Belding-Royer E. M. Delay tolerant mobile networks (DTMNs): controlled flooding in sparse mobile networks. Proceeding NETWORKING'05 Proceedings of the 4th IFIP-TC6 international conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communication Systems. Pages 1180-1192.

[19] R. S. Mangrulkar, Atique M. Article: Performance Evaluation of Flooding Based Delay Tolerant Routing Protocols. International Journal of Computer Applications NCETCSIT(1):35-40, febrero 2012.

[20] Aguero R., Perez J. “Adding Multiple Interface Support in NS-2”; enero 2007.

[21] Hal Abelson, Philip Greenspun, Lydia Sandon; “Tcl for We Nerds”; julio 2014; Disponible en la dirección web: http://philip.greenspun.com/tcl/

[22] Kurkowski S., Camp T., Colagrosso M; “MANET Simulation Studies: The Incredibles”; ACM SIGMOBILE Mobile Computing and Communications Review, 2005.

[23] Ros F., Ruiz P; Implementing a New Manet Unicast Routing Protocol in NS2; diciembre 2004.

[24] Sadkhan S., Mohammed S; DESIGN OF WIRELESS NETWORK BASED ON NS2; Journal of Global Research in Computer Science; diciembre 2012.

[25] Wu Z.; Network Simulator 2 for Wireless: My Experience; Disponible en la dirección: http://www.winlab.rutgers.edu/~zhibinwu/html/network_simulator_2.html.

[26] Chung J., Claypool M; NS by Example; Disponible en la dirección web: http://nile.wpi.edu/NS/

[27] Tcl/Tk Quick Reference Guide; Disponible en la dirección web: http://www.slac.stanford.edu/~raines/tkref.html

[28] OTcl Tutorial (Berkeley Version); Disponible en la dirección web: http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl

[29] OTcl Tutorial (MIT Version); Disponible en la dirección web: ftp://ftp.tns.lcs.mit.edu/pub/otcl/README.html

[30] XGraph; Disponible en la dirección web: http://jean-luc.ncsa.uiuc.edu/Codes/xgraph

[31] Network Animator (NAM); Disponible en la dirección web: http://www.isi.edu/nsnam/nam/

[32] Altman E., Jimenez T. NS for beginners: Disponible en: http://www.sopinria.

fr/mistral/personnel/Eitan.Altman/COURS-NS/n3.pdf

[33] Paquereau L., Helvik B. "A module-based wireless node for NS-2"; Proceeding WNS2 '06 Proceeding from the 2006 workshop on ns-2: the IP network simulator Article No. 4.

[34] Patil. S; "HOMOGENEOUS MULTI-INTERFACE MOBILE NODE SUPPORT IN NS2"; International Journal of Communication Network and Security (IJCNS) ISSN: 2231-1882 Volume. 1 Issue. 4.

[35] Yue Wang; "A Tutorial of 802.11 Implementation in ns-2"; Mo iTec La , CUHK

[36] Heidrich W., Slusallek P; "Automatic Generation of Tcl Bindings for C and C++ Libraries". Proceeding TCLTK '98 Proceedings of the 3rd Annual USENIX Workshop on Tcl/Tk - Volume 3 páginas 10-10, 1995.

[37] Brent Welch B; "Practical Programming in Tcl and Tk" 4th Edition; junio 20, 2003.

[38] Discrete-event simulation software; Disponible en la dirección web: http://www.topology.org/soft/sim.html

[39] Fall K., Varadhan K., Huang P; UCB ns-2 tutorial workshop, junio 1999.

[40] NS2.26SourcesOriginal Documentation; Disponible en la dirección web: http://www.netlab.ecnu.edu.cn/software/ns-class/index.htm

[41] Teerawat Issariyakul's Personal Home Page; Disponible en la dirección web: http://www.ece.ubc.ca/~teerawat/NS2.htm

[42] NS2 ULTIMATE; Disponible en la dirección web: http://ns2ultimate.tumblr.com/