UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de un sistema basado en microcontrolador ARM Cortex-A

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Javier Moraga Matoque

Tutor: Juan Vicente Capella Hernández

Tutor Experimental: José Navarro Alabarta

2015/2016

Resumen

El concepto de redes vehiculares va cogiendo fuerza en los últimos tiempos debido a su potencial para mejorar la seguridad y la fiabilidad en la carretera. La creación de un sistema eficiente de comunicación entre vehículos, que minimice las pérdidas de paquetes y los entregue con retardos que no supongan problemas, es crucial en este tipo de redes.

En este documento se exponen los protocolos más adecuados para redes vehiculares y se implementa uno basándonos en geo-localización de nodos y un sistema de lógica difusa a partir de cálculos basados en las coordenadas obtenidas.

El protocolo estudiado e implementado en este proyecto sigue esa filosofía, llevándose a cabo dicha implementación sobre dispositivos basados en un microcontrolador Cortex-A. Estos dispositivos comunicándose entre ellos en modo "ad-hoc" son capaces de intercambiar la información necesaria que permita conocer parámetros significativos de nodos cercanos para usarlos como métrica de enrutamiento.

Este tipo de protocolos son peculiares ya que los diferentes dispositivos que forman parte de la red crean y destruyen enlaces de forma muy dinámica, por ello se hace indispensable que sus funciones estén muy adaptadas a la movilidad.

Palabras clave: Raspberry, microcontrolador, VANET, enrutamiento, protocolo.

Abstract

Vehicular networks are getting stronger over the last years due to their potential to improve security and reliability to vehicles driving over highways. The creation of an efficient communication system between vehicles (V2V communications), that minimizes packet-loss and deliveries them with delays that doesn't suppose a problem, is crucial in this kind of networks.

In this document the most adequate VANET (vehicular network) protocols are exposed. I make an implementation of one protocol similar to those that uses "fuzzy logic" to make routing decisions based on calculations using nodes positions.

The studied and implemented protocol follows that philosophy by performing the actual implementation over devices that are based on Cortex-A microcontrollers (Raspberry Pi devices are used), These devices communicating between them in ad-hoc mode are capable of exchanging the needed information that allows us to obtain significant parameters of close nodes in order to use them as routing metric.

This kind of protocols are very singular, different devices that are part of the network create and destroy links in a dynamic way, for this reason its compulsory that its features are adapted to mobility.

Keywords : Raspberry, microcontroller, VANET, routing, protocol.

Tabla de contenidos

1. Introducción

Las redes vehiculares constituyen actualmente un campo emergente de investigación del que aún son muchas las aplicaciones y beneficios que se pueden obtener. Vemos que hoy en día la seguridad en carretera es un tema que nos preocupa como sociedad y toda mejora posible resulta muy positiva.

Con este trabajo de fin de grado me introduzco en el mundo de las redes vehiculares para comprobar cuales son las múltiples y diversas aplicaciones que se le pueden dar a las mismas.

Todas estas aplicaciones observadas no serían posibles sin una buena forma de comunicar a los diferentes vehículos que necesitarán intercambiar información entre ellos sobre la marcha. El desarrollo de un buen protocolo de enrutamiento que funcione en entornos altamente móviles y por ello cambiantes constituye la base de toda aplicación a implementar en las redes vehiculares.

Estudiando los distintos protocolos de enrutamiento para VANET (*Vehicular Ad-hoc Network*) que se usan hoy en día se pretende realizar una implementación real sobre máquinas *Raspberry Pi* que pueda solucionar con solvencia la comunicación entre nodos en entornos VANET, obteniendo bajas latencias, buena tasa de error y tratando de minimizar el *overhead* en la red.

1.1 Objetivos

Al embarcarse en un proyecto hay que marcarse unas metas al inicio, que nos darán una idea de lo que se espera del mismo y que se intentarán cumplir en su mayoría al momento de la finalización del mismo.

Para la realización de este proyecto se plantea satisfacer los siguientes objetivos:

- Análisis sobre la tendencia actual a la hora de implementar redes vehiculares.
- Conocer los distintos tipos de protocolos de enrutamiento en redes vehiculares y sus características.
- Análisis del protocolo a implementar sobre los nodos reales.
- Implementación funcional del protocolo.
- Experimentación con el protocolo implementado y obtención de resultados.
- Análisis del resultado obtenido.
- Propuesta de mejoras sobre el proyecto.

1.2 Estructura de esta memoria

Con esta memoria se pone en contexto al lector en lo relativo a las redes vehiculares, de forma que pueda entender a grandes rasgos que se intenta obtener con este proyecto y que se espera del mismo.

En el capítulo uno de la memoria se presenta la introducción al proyecto donde se nos dice porque se ha realizado el mismo y cuales son los objetivos marcados para la fecha de su finalización.

Pasamos al capítulo dos, donde se comentan las tecnologías de las que nos hemos servido para desarrollar todo lo necesario para poner el protocolo a funcionar. Cada

tecnología es estudiada y comprendida dentro de este documento para poder enfocar su uso dentro del proyecto.

Habiendo ya comentado todas las tecnologías usadas, pasaremos a comentar aspectos sobre la implementación realizada. En el capítulo tres se comenta cada módulo del programa con la utilidad que tiene dentro del mismo sin entrar en detalles del código. Se presentan aquí también los distintos problemas con los que me voy encontrando durante el desarrollo y la solución a los mismos.

Teniendo una implementación válida corriendo en los nodos podremos realizar experimentos, con la finalidad de ver como se comporta el protocolo en determinadas situaciones y saber si nos da el comportamiento que se esperaba. Estos experimentos serán listados y analizados sus resultados dentro del capítulo cuatro de esta memoria.

En base a los resultados obtenidos y al funcionamiento del protocolo podremos sacar conclusiones y exponerlas en el apartado final, añadiendo en el mismo cualquier mejora que se considere oportuna.

2. Tecnologías relacionadas

El proyecto se realiza en base a diferentes ideas y tecnologías que ayudan a modelar un programa final que se acerque lo máximo posible a los objetivos marcados, en este apartado se comentan cuáles hemos añadido al proyecto para conseguirlo.

2.1 Protocolos VANET

Las VANET son un tipo especial de MANET (*Mobile Ad-hoc Network*) donde los nodos son vehículos comunicados inalámbricamente y actuando como servidores y clientes al mismo tiempo para poder intercambiarse información entre ellos.

Tanto las VANET como las MANET comparten la idea de no confiar en infraestructuras de red fijas. Al ser así, se auto regulan y organizan. Como inconvenientes en este tipo de redes tenemos que los anchos de banda para transferir información no son muy altos y también que el radio inalámbrico en el cual se pueden enviar datos es bastante limitado.

El enrutamiento de datos entre vehículos en movimiento es un reto (Comunicaciones Vehículo con vehículo "V2V" o Vehículo con infraestructura "V2I"), debido a los frecuentes y rápidos cambios en la topología de red y las continuas conexiones y desconexiones, que hacen difícil diseñar un protocolo de enrutamiento eficiente.

En este tipo de redes la información es compartida entre los nodos sin que haya ningún nodo central que coordine todo, esto es positivo ya que no hay solo un punto de fallo en la red, si un nodo no funciona correctamente no supone un grave problema para la transmisión de información. Por otro lado, el hecho de que los nodos que de proveen información realmente importante para las aplicaciones sean móviles provoca que la transmisión de datos esté muy lejos de ser óptima.

2.1.1 Arquitecturas de red en VANET

Las VANET, según el tipo de enlaces que usen para formar la red pueden encajarse en diferentes grupos [2]:

- ***Pure cellular* o WLAN**: Este tipo de VANET se sirve de *cellular gateways* fijas y/o puntos de acceso WLAN o WiMax distribuidos en las intersecciones de tráfico que nos puedan proveer de acceso a internet, información sobre el tráfico o que nos puedan servir como *routers* hacia otros nodos en la red.
  Las VANET pueden servirse a la vez de *cellular gateways* y WLAN para formar la red.
- **Ad-hoc:** Este tipo de redes VANET están formadas por los propios vehículos dotados de conectividad y también de dispositivos fijos que contactarán con los dispositivos de la calzada. Estos dispositivos y los vehículos formarán una red ad-hoc pura.
- **Híbridas:** Este tipo de redes están formadas por las redes ad hoc comentadas anteriormente y a su vez los nodos ad hoc podrán contactar e intercambiar información con redes infraestructura WLAN o *pure cellular*.

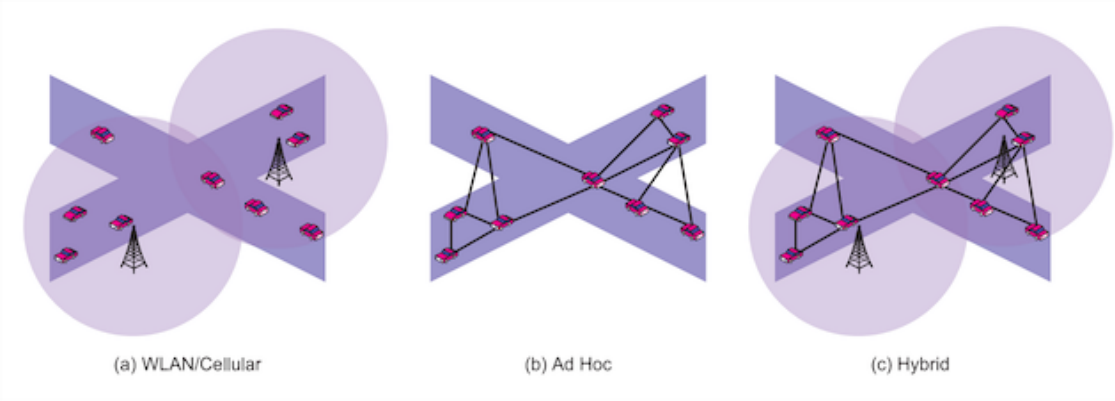


Ilustración 1. Tipos de infraestructura en una red VANET

2.1.2 Características de las VANET

Una red VANET no puede ser tratada de la misma forma que una MANET, ya que posee ciertas características que la hacen más complicada:

- Topología de red cambiante con frecuencia.
- Es una red distribuida, todos sus nodos participan en la creación de la red operativa.
- Red en frecuente desconexión, la densidad de nodos para un nodo ejecutando el protocolo variará muy sustancialmente en el tiempo.
- Sus nodos poseen gran capacidad de almacenamiento y mucha energía para funcionar.
- Los sensores que vienen incorporados en los nodos han de ser tratados para obtener de ellos información de vital importancia en este tipo de redes.
- El ambiente en el que se muevan los nodos afectan en gran medida a la calidad de la señal. Los nodos no se comunicarán de igual forma en ambientes densos (muchos obstáculos donde rebote y se disipen las comunicaciones) que en ambientes menos cargados donde pueda llegar sin problemas la señal.
- En ciertas aplicaciones (sobretodo algunas relativas a los ITS o *Intelligent Transport Systems*) se necesita que la información sea entregada en los destinos con el mínimo retardo posible. Por ejemplo, en un sistema de conducción automática por carretera cuando un coche pulsa el freno esa información ha de llegar a sus vecinos casi de inmediato para evitar colisiones entre vehículos.

2.1.3 Distribución de la información

La manera en la que se distribuye la información dentro de una red de este tipo estará marcada por el protocolo de enrutamiento subyacente. Dentro de las VANET se han utilizado diversos protocolos afrontando el reto de diseminar la información en una red tan cambiante, podemos observar grandes subgrupos[1][3][4]:

1. **Protocolos basados en la topología de red**

   Se nutren de información de los enlaces a lo largo de nuestra red. Cada nodo conoce la topología a su alrededor, en base a esto y a criterios que varían según el protocolo veremos distintas decisiones de enrutamiento.

Dentro de los protocolos de enrutamiento basados en a topología podemos ver dos tipos:

- **Protocolos proactivos:** Guardan la información relativa al enrutamiento (como el *next-hop* o los vecinos) en caché para ser utilizada por el protocolo. En este tipo de protocolos no es necesario realizar un descubrimiento de ruta de forma muy frecuente ya que una vez se descubre una buena ruta ésta es usada hasta que deja de ser válida. Sin embargo, esta clase de protocolos no dan resultados muy favorables en nuestro caso de estudio, ya que las VANET tienen una naturaleza dinámica que añadiría demasiado *overhead* en la red al tener que descubrir rutas nuevas con frecuencia.

- **Protocolos reactivos:** Son aquellos que sólo abren una ruta cuando sea necesaria para un nodo, de ahí que consideren su comportamiento como ad-hoc. Debido a su similitud con una MANET, se han probado diversos protocolos orientados a estas como pueden ser AODV o DSR. Estos protocolos son de propósito general y mantienen únicamente aquellas rutas que sean necesarias, pero en una VANET hay tantos cambios en cortos periodos de tiempo que se dificulta mucho el buen hacer de los mismos. Por ello se pensó que tal vez estos protocolos podrían adaptarse al nuevo entorno, teniendo en cuenta sus características para resolver los problemas que nos aparecen.
  Un ejemplo de estas mejoras destinadas a combatir la pérdida de enlaces provocada por la movilidad es la creación de protocolos de predicción. Un claro ejemplo de este tipo de protocolos son PRAODV y PRAODVM que lo que hacen es predecir cuanto va a durar un enlace en base a la velocidad relativa entre los nodos y la distancia de los mismos. Cuando se predice que el enlace está a punto de perderse se pone en marcha un nuevo cálculo de ruta hacia el destino, de esta forma el fallo que se daría en el protocolo AODV normal se vería cubierto con esta ruta más fresca.
  Ejemplos: PRAODV, PRAODVM, AODV, TODA, DSR...

2. Protocolos basados en coordenadas geográficas

El movimiento en los nodos que transitan una VANET suele venir prefijado por las calles en las que se mueven, y suele tener dos direcciones. Este hecho nos da a pensar que las estrategias de enrutamiento basadas en localización geográfica, apoyadas con mapas, modelos de tráfico o sistemas de navegación tengan sentido.

En diversos estudios se ha comparado el rendimiento de los protocolos basados en la topología frente a los basados en la localización de los nodos. Los resultados dejan ver que utilizar la localización de los nodos arroja mejores resultados que buscar rutas a usar para enviar los mensajes en la red.

A pesar de este hecho, estos protocolos tienen problemas que afrontar. La mayoría basas sus decisiones en información sobre la localización.

Ejemplos de esto son:

- ***Greedy routing*:** Protocolo consistente en enviar los paquetes al *next-hop* mas cercano al destino.
- ***Greedy perimeter stateless routing*:** Sufre de problemas al usarse en escenarios urbanos, debido a los obstáculos que pueden haber en las calles que impiden que la señal llegue al destino correctamente.
- ***Greedy perimeter coordinator routing*:** Aquí se combate el problema de los obstáculos añadiendo nodos coordinadores en las zonas susceptibles a problemas como pueden ser las intersecciones. Los coordinadores toman decisiones de enrutamiento y se eligen mediante heurísticas.

Ejemplos: *Connective aware routing* (CAR), *Diagonal-Intersection-Based Routing Protocol* (DIR) y protocolos tolerantes a retrasos (MOVE, VADD, SADV).

3. Enrutamiento usando *clusters*

En este tipo de protocolos los nodos móviles se agrupan en infraestructuras de red virtuales llamadas *clusters*. Los nodos que forman parte de un *cluster* pueden comunicarse de forma ad-hoc directa. La comunicación entre *clusters* se lleva a cabo mediante un nodo central (llamado *cluster head*) que coordina las comunicaciones dentro del *cluster* que dirige y las comunicaciones con el resto de *clusters*.

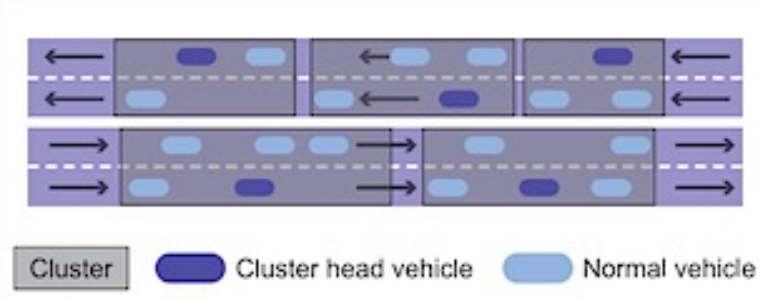


Ilustración 2. Descripción de estructura de red en *clusters*

Los protocolos que se basan en esta idea se comportan bastante bien en redes MANET, pero las diferencias respecto a una VANET hace complicado que éste tipo de protocolos funcionen bien en las redes vehiculares. La creación de *clusters* en un entorno tan cambiante no arroja buenos resultados, ya que se disuelven muy rápidamente y no proporcionan la escalabilidad deseada a la red.

Protocolos basados en *clustering* para VANET:

- ***Clustering for open IVC Networks* (COIN):** El nodo que actúa como *head* se elige en base al movimiento de los vehículos y a las intenciones del conductor.
  También hace frente a los cambios en las distancias de los vehículos de la red. Los resultados obtenidos muestran que los *clusters* formados son mas estables pagando un poco de *overhead* en la red como penalización.

- **LORA_CBF:** es un protocolo de localización reactivo, usa *flooding* dentro de los *clusters*. En él, un nodo puede ser *head*, *gateway* o miembro normal de un *cluster*. Un nodo actuando como *gateway* es encargado de la conexión entre *clusters* y la información sobre que miembros y que *gateways* tenemos en la red la llevan los nodos *head*.
  Los paquetes de este protocolo se envían al destino de forma parecida al modo *greedy*. Si no conocemos la localización del destino entra en juego el paquete *Location Request* (LREQ) que recibe un *Location Reply* (LREP) como respuesta.
  Estos LREQ y LREP solo pueden ser diseminados por nodos *head* y *gateway*.
  En escenarios típicos urbanos y de carretera muestra que el protocolo funciona mucho mejor que AODV o DSR, sobretodo cuando escalamos la red o se dan situaciones de mayor movilidad de los nodos.
  Sin embargo, no resultan muy positivos el *delay* y el *overhead* necesarios para formar y modificar un *cluster*, ya que este hecho sucede con frecuencia en una VANET.
  Ejemplos: COIN, LORA-CBF

4. Protocolos basados en *Broadcast*

El enrutamiento usando *broadcast* es muy usado en las VANET; mucha información (tráfico, meteorológica, emergencias...) es de utilidad para todos los vehículos de la red.

El *broadcasting* es usado también en protocolos de *routing unicast* para encontrar información que nos permita encontrar una ruta eficiente hacia el destino. Cuando el destino sale del rango de alcance se pasa a usar *multi-hopping*.

La manera más simple y usada para realizar *broadcasting* es haciendo *flooding*. En él, el nodo emisor envía el paquete a sus nodos vecinos, y estos a su vez lo vuelven a enviar a todos sus vecinos excepto al que les entregó el paquete.

El hecho de que todos re-emitan el mensaje puede ocasionar bucles, un paquete podría estar retransmitiéndose eternamente, por ello se introducen los números de secuencia en los paquetes.

Cada vez que un nodo emite un paquete incrementa su número de secuencia en uno, de esta forma un paquete se ve identificado por la dupla <Dirección IP origen, número de secuencia>. Si recibimos un paquete de un nodo del que sabemos que tenemos un número de secuencia mayor en caché, se descarta el paquete por no ser "fresco" y no se enviaría a todos los vecinos.

El principal problema del *flooding* es que al escalar la red, el ancho de banda necesario puede incrementarse de manera exponencial. Pueden darse colisiones y contenciones ya que los nodos emiten los mensajes casi al mismo tiempo.

Pasamos ahora a destacar algunos protocolos basados en *broadcast*:

- **BROADCOMM [6]**
  Este protocolo sigue una estructura jerárquica a usar en autopistas. Los nodos se organizan en células virtuales y en dos posibles niveles de jerarquía. El primer nivel incluye todos los nodos en una célula, el segundo nivel está compuesto por los reflectores de célula (*cell reflectors*) que son unos pocos nodos situados cerca del centro de la célula.

  Los reflectores de célula actúan durante cierto tiempo como lo haría un head en los *clusters* comentados anteriormente, tratando los mensajes de emergencia de los miembros de la célula o miembros cercanos de otra. Además sirve para enrutar dentro de la célula mensajes de emergencia que vienen de otras células.

  Este protocolo deja en evidencia a otros protocolos de *flooding* similares, pero al ser tan simple sólo resulta adecuado para un escenario de autopista.

- ***Urban Multi-Hop Broadcast Protocol* (UMB)**
  Destinado a combatir interferencias, colisiones de paquetes y el problema del nodo oculto cuando hacemos *broadcasts* de más de un hop.

  Los nodos emisores intentan seleccionar el nodo más lejano en la dirección del *broadcast* para asignarle la tarea de enviar y comprobar el envío del paquete sin tener ninguna información sobre la topología de red.

  En las intersecciones disponemos de repetidores para poder enviar paquetes a todas las direcciones en la intersección. Este protocolo tiene mayor éxito cuando hay mucha carga de paquetes y la densidad del tráfico que otros protocolos 802.11 basados en aleatoriedad y distancias.

- ***Vector-based Tracking detection* (V-TRADE) y *History-enhanced V-Trade* (HV-TRADE)** son protocolos de *broadcast* basados en GPS.[7]

  Su idea básica es similar a la del protocolo unicast llamado *Zone Routing Protocol* (ZRP). Basándose en la información de posición y movimiento, sus métodos clasifican a los vecinos en diferentes grupos. Para cada grupo sólo un subgrupo de *border vehicles* son seleccionados para reenviar mediante otro *broadcast* el mensaje. De este modo se mejora la utilización del ancho de banda con una ligera pérdida de alcance, al no usar todos los vehículos como repetidores. Aún así tienen bastante *overhead* al tener que seleccionar los nodos emisores en cada salto.

Ejemplos: BROADCOMM, UMB, V-TRADE, V-CAST

5. Routing basado en geocast

Son desarrollados con el objetivo de enviar el paquete de manera *multicast* pero seleccionando y creando el grupo *multicast* según la zona geográfica de interés (ZOR, *Zone of relevance*).

Una aplicación de esto sería el caso en el que un vehículo tiene un accidente y el sistema en sí detecta este hecho mediante la comprobación de ciertos sensores del coche. Al detectarse el accidente el protocolo acota una zona de interés a la

que serán enviados los paquetes, ahorrándose así paquetes innecesarios en zonas donde no sería necesaria la información del accidente.

La mayoría de estos protocolos se basan en hacer *flooding* acotado a una determinada zona (fuera de esta no habrá reenvío). También hay protocolos que no se basan en el *flooding*, aunque aún siendo así incluso podrían utilizarlo en ciertas regiones.

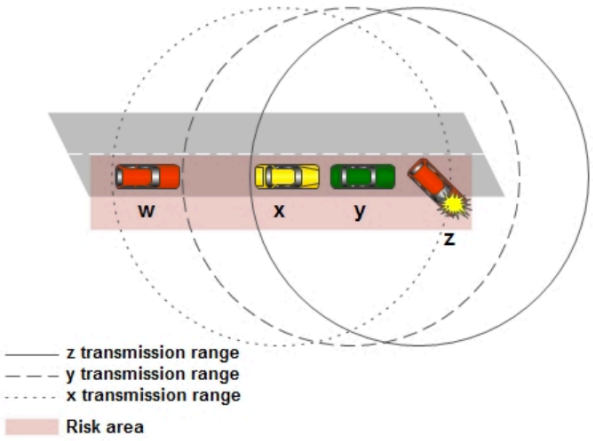


Ilustración 3. Zona de riesgo donde se realizan envíos GEOCAST marcada en rojo

Se busca evitar colisiones y reducir el número de "*rebroadcasts*" en el protocolo *geocast*. En [8] se presenta un protocolo de este tipo.

Cuando un nodo recibe un paquete no lo reenvía inmediatamente, primero espera un tiempo para tomar la decisión de enviar el *"rebroadcast"* o no. Cuanto más lejos está el nodo que ha recibido el paquete del emisor menos se tarda en tomar la decisión. Cuando el tiempo expira si no hemos recibido el mismo mensaje de otro nodo procedemos a su "*rebroadcast*". Así se evita el problema que suponen las *broadcast storms* y el envío es optimizado para el vehículo que inicia la comunicación.

***Intervehicles geocast protocol* (IVG)** es un ejemplo de este tipo de protocolos que sirve para enviar alertas dentro de un área de riesgo.

Hay otras aproximaciones añadiendo cachés a la capa de *routing* que mantengan paquetes que no hayan podido ser enviados dentro de una ZOR, ya que se tiene en cuenta los frecuentes cambios de vecinos. Cuando un nuevo vecino llega a una ZOR o hay cambios dentro de la misma habrá posibilidad de enviar ese paquete que antes no ha tenido salida.

Los estudios demuestran que el tener una caché para mensajes que no han podido ser enviados debido a problemas con particionamiento de la red o vecinos desfavorables mejora el ratio de envío de paquetes correcto.

Ejemplos: IVG, DG-CASTOR, DRG

2.1.4 Aplicaciones en VANET

Las redes VANET pueden ser utilizadas para comunicar vehículos en movimiento con múltiples propósitos, habrá que tener en cuenta ciertas características a la hora de decidir que aplicación correremos sobre ellas que se resumen en la siguiente tabla.

Vehicular Network	Tipo de aplicación	• Aplicación de seguridad • Transporte inteligente • Destinada al confort
	Calidad de servicio	• Tiempo real relajado • Tiempo real estricto • Tolerante a retardos
	Alcance	• Local • *Wide area*
	Arquitectura de red	• Ad hoc • Infraestructura • Híbrida
	Tipo de comunicación	• V2I (Vehículo-Infraestructura) • V2V(Vehículo-Vehículo)

Tabla 1. Características de red vehicular[10]

Tenemos aplicaciones destinadas a diversos campos [9], el más importante es el de la seguridad; las aplicaciones de este campo mejoran las condiciones para la conducción y reducen las probabilidades de accidente alertando al conductor de posibles peligros con tiempo o, por ejemplo, aplicando los frenos cuando sea necesario.

Dentro de las aplicaciones de seguridad podemos encontrar distintas funciones:

- Alerta de colisión cooperativa
- Manejo de incidentes
- *Streaming* de video de emergencia

Otro campo en el que las redes vehiculares pueden ser útiles es en la creación de sistemas de transporte más inteligentes (ITS) que consigan una entrega más rápida y eficaz de la información relacionada al tráfico y la conducción. Estas aplicaciones pueden hacer uso de procesamiento colaborativo de información para detectar aspectos del tráfico que no son visibles a simple vista.

Podemos destacar en estas:

- Monitorización del tráfico
- Gestión del tráfico
- *Platooning* o trenes de carretera

- *Tracking* de vehículos
- Servicios de notificación

El último campo reseñable en este documento para aplicaciones vehiculares sería el relacionado con las funciones que otorgarían al conductor una mayor comodidad al utilizar su vehículo.

- Gestión de plazas de aparcamiento
- Juegos distribuidos o comunicación por voz
- Aplicaciones *Peer-to-Peer*

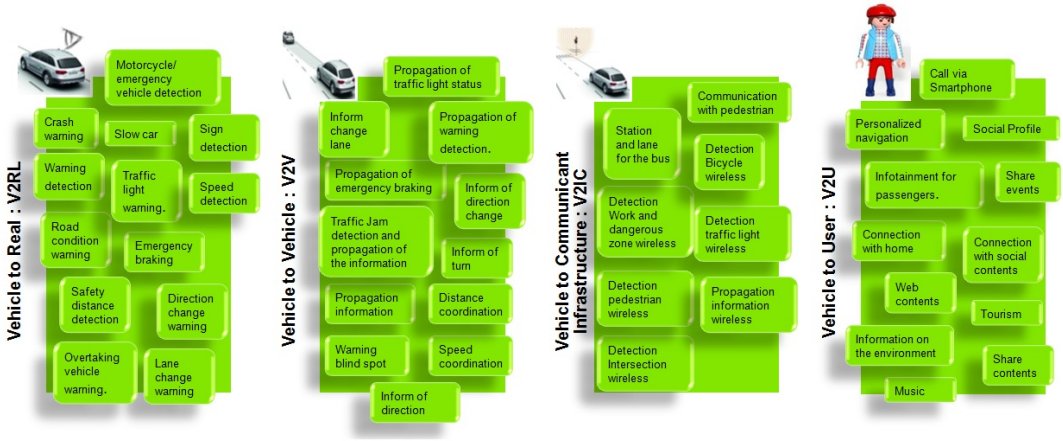


Ilustración 4. Posibles utilidades de aplicaciones VANET

Todas estas aplicaciones tendrán que tener en cuenta que requisitos de los nombrados en la tabla han de cumplir, y por tanto habrá que ver que red y que protocolo de enrutamiento resulta mas beneficioso para la ejecución de cada una.

2.1.5 Modelos de movilidad en VANET

A la hora de probar los protocolos antes de ser llevados al mundo real, es importante ver que se puede obtener de los mismos un comportamiento seguro.

Para ellos tenemos los modelos de movilidad que obtienen una gran importancia al ser el método que tenemos para evaluar como se comportaría el protocolo en escenarios que se darían en la realidad. En algunos protocolos estos modelos de movilidad sirven de apoyo para predecir los siguientes movimientos de los nodos de la red y por ello tomar mejores decisiones de enrutamiento.

Tenemos distintos modelos:

- ***Random WayPoint mobility model [11]:*** Es un modelo basado en la aleatoriedad para la generación de destinos. El modelo va generando destinos aleatorios a los que deberá acudir el nodo. Este modelo es ampliamente utilizado en simulaciones de redes ad hoc pero no representa ninguna situación real.
  Con el tiempo, se modificó este modelo para parametrizarlo con datos como la longitud de la carretera, el número de carriles o la distancia entre vehículos.
- Un **modelo** que se acerca a las situaciones cotidianas fue creado por **Saha y Johnson**, quienes, con la ayuda de la información de un mapa de Estados

Unidos *("TIGER US roadmap by US bureau"*) convertido a grafo pudieron aplicar sobre el algoritmos que calculaban la ruta más corta entre dos puntos, tenían en cuenta aspectos como las velocidades en esas calles.

- ***Street random waypoint* (STRAW)** también está basado en el mismo mapa. Tiene en cuenta el tráfico en el escenario urbano y la interacción entre vehículos y controles de tráfico.
- **Uso de trazas reales de vehículos** durante su tránsito en carretera, el movimiento de estos se define en duplas <posición, tiempo>. Con estas trazas se puede dotar de más información a los modelos de movilidad.

2.2 Módulos *wireless*

No todos los módulos *wireless* nos van a dar el mismo rendimiento. Por ello aquí especificamos cuales son los requisitos a tener en cuenta a la hora de realizar la implementación para elegir módulos *wireless* que puedan comunicarse con los el resto de nodos evitando posibles problemas[12].

Buscamos un buen rendimiento en un escenario en el que el alcance de nuestra señal es primordial si queremos un sistema que funcione correctamente y no sufra muchas interferencias.

Por ello buscaremos un módulo que cuente con un *chipset* compatible con nuestra *Raspberry Pi* (hoy en día casi todos los módulos que se comercializan cumplen con este requisito) y que también nos proporcione la funcionalidad necesaria para crear una red ad-hoc.
El tema de los estándares 802.11 nos afecta en el aspecto de que banda de frecuencia utilizaremos a la hora de comunicar nuestros dispositivos, y es que varios estándares solo funcionan en la banda de 2.4 GHz, que nos proporciona un mayor alcance de la señal, pero sus canales son más propensos a interferencias ya que los 2.4 GHz son utilizados por toda clase de dispositivos. El uso de la banda 2.4GHz nos interesa por el hecho del mayor alcance que le otorga a la señal. Las redes vehiculares tienen su propio estándar llamado 802.11p[21].

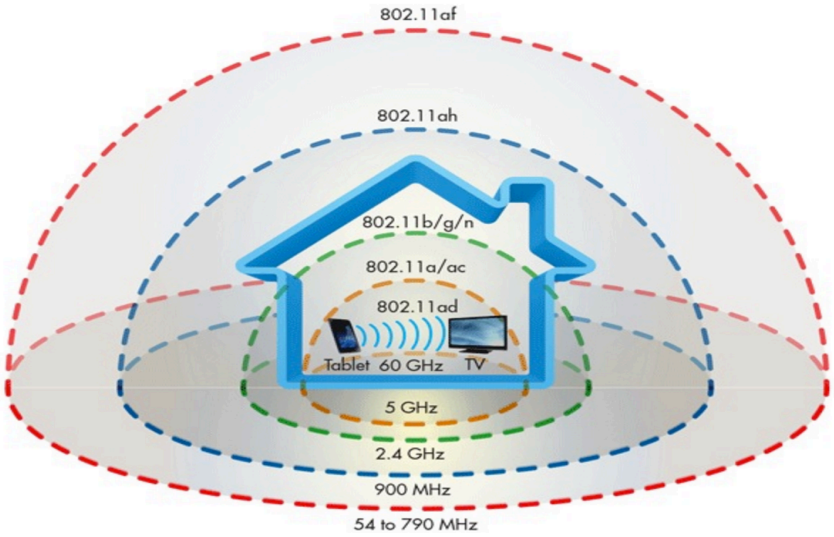


Ilustración 5. Alcances simbólicos de la señal para cada estándar *wireless* ordenados.

El ancho de banda que podamos transmitir con el módulo comprado en principio no será un problema, las aplicaciones en redes vehiculares usan paquetes que no suponen un gran volumen de datos ya que necesitan pequeñas cantidades de información de los nodos vecinos para poder funcionar.

2.3 *Global Positioning System* (GPS) e *Inertial Measurement Unit* (IMU)

El protocolo implementado precisa de coordenadas geográficas para poder funcionar correctamente por ello necesitamos realizar la elección de un módulo GPS que nos otorgue una buena precisión en la localización (que nos de localizaciones exactas con errores muy bajos, de lo contrario nuestro protocolo al no usar datos correctos podría realizar incorrectamente el enrutamiento de los paquetes a destinos que no sean óptimos según el punto de vista del protocolo).

También podremos utilizar el módulo para obtener la hora vía satélite y asegurarnos así la sincronización de todos los relojes de la red.

Una vez elegido el módulo a usar se utilizará una API (*Application Programming Interface*) para obtener de el los datos relevantes para el protocolo.

Sería interesante también contar con una IMU en nuestro nodo, ya que nos podría proporcionar parámetros interesantes como la orientación de nuestro nodo o el sentido de la marcha que está tomando. Con estos datos podríamos nutrir al sistema de decisiones de nuestro protocolo de forma que pueda proporcionar un enrutamiento más preciso y eficiente.

La unidad de medición inercial se integraría en nuestra *Raspberry Pi* mediante la compra de algún módulo compatible. La finalidad de un dispositivo como éste es la de obtener mediciones de aceleraciones, orientaciones y fuerzas gravitacionales, quaterniones y ángulos de Euler. En los últimos años se ha conseguido con ellas, incluso, la creación de dispositivos GPS que no se vean afectados por la interferencia electromagnética.

Los principales problemas a afrontar a la hora de integrar una IMU en nuestro proyecto son el encontrar un módulo que cumpla con los requisitos del sistema global y a su vez ver como funciona y saber como obtener e interpretar los datos que obtenemos en el programa provenientes del módulo que la integra. Para todo esto, en [13] se presenta un trabajo de donde se pueden obtener las pautas a seguir.

Mediante el uso de este dispositivo y ciertos cálculos se puede estimar el movimiento del vehículo en zonas donde no alcanza la señal GPS. (Sistema de posicionamiento para vehículos autónomos)

2.4 *Network Time Protocol (NTP) [14][15]*

Este protocolo es el método más utilizado para sincronizar el reloj *software* de un sistema operativo GNU/Linux con los servidores horarios de internet.

Con este protocolo se combaten problemas como la latencia que puede afectar al enviar la hora vía red. De esta forma se consiguen sincronizaciones de hora dentro de un margen de decenas de milisegundos respecto a Internet. En redes de área local la sincronización se puede obtener con una precisión mayor, de hasta un milisegundo.

Este protocolo es una buena forma de lograr una sincronización correcta de relojes entre nodos que no posean fuente fiable alguna de dónde obtener una hora para poner en común.

De esta forma, para sincronizar los relojes en los dispositivos de una red local, este protocolo puede ser utilizado.

Mediante la modificación dentro del sistema operativo del fichero de configuración de este protocolo se podrá hacer que un nodo pueda actuar como nodo servidor facilitando su hora al resto de nodos de la red, que actuarán como clientes del mismo y podrán obtener la hora del mismo, quedando todos sincronizados a partir de este.

El nodo que actúa como servidor añadirá su propio reloj local como fuente de reloj posible para los mensajes de respuesta a las peticiones del cliente, ya que en caso de desconexión a Internet podrá ser usado.

Los clientes tendrán que configurar sus fuentes de reloj de forma que apunten a la dirección del servidor creado, y finalmente podremos forzar una actualización de la hora haciendo uso de utilidades como *ntpdate* o el comando “ntpd –qg”

3. Implementación

3.1 Protocolo de enrutamiento implementado

El protocolo implementado sobre la *Raspberry Pi* es el llamado *Stability and reliability aware routing (SRR)* descrito en [16][17] y adaptado a las comunicaciones *Vehicle to vehicle* para que los vehículos se puedan comunicar sin una infraestructura subyacente.

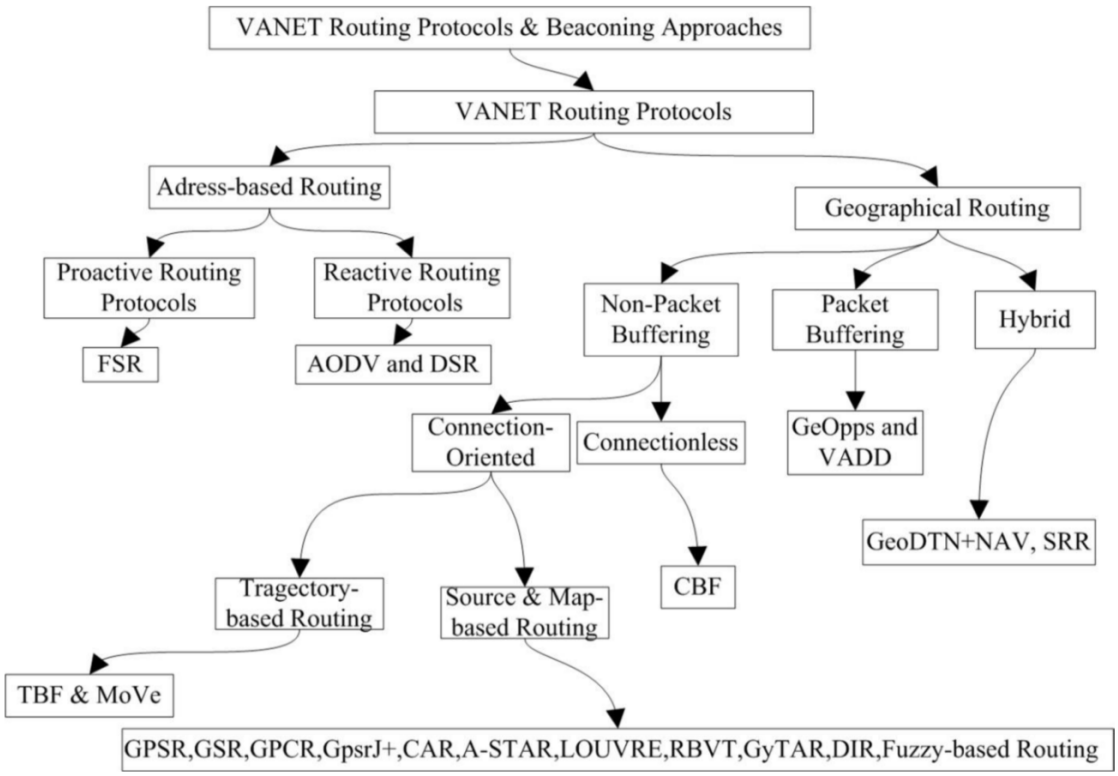


Ilustración 6. Clasificación de protocolos posibles para VANET

Este protocolo se puede encajar en el marco de los protocolos geográficos (basados en la posición de los nodos) e híbridos. Utiliza también características de los protocolos de *broadcast*, ya que sus mensajes HELLO iniciadores de la comunicación se envían a todos los nodos de la red utilizando *flooding*.

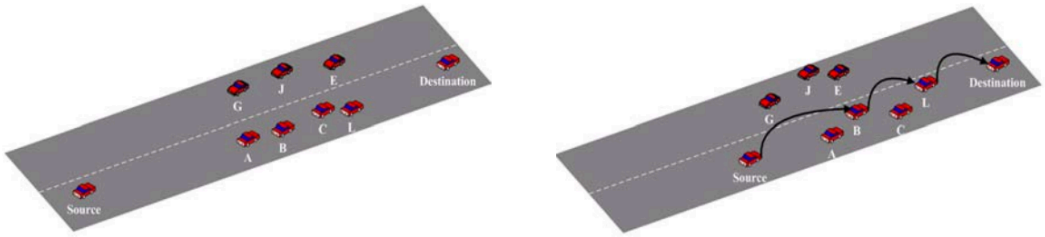
La ruta a seguir por un paquete de origen a destino se calcula sobre la marcha en base a decisiones tomadas mediante parámetros calculados a partir de la posición y orientación de los nodos en la red.

El protocolo cuenta con dos modos de funcionamiento:

- **El método SRR que se usa en zonas de tráfico denso:** El emisor comprueba que hay al menos un nodo en su tabla de vecinos. Si su tabla no está vacía, comprueba si el destino del paquete está en ella, si lo está se le entrega el paquete.
  Si el destino no figura entre los vecinos del nodo emisor, entra el juego el motor de inferencia basado en lógica difusa. El nodo que quiere enviar el paquete calcula el coste *fuzzy* de cada nodo vecino y se lo envía al que mejor resultado

tenga (mayor *fuzzy cost* implica mejor vecino). Este coste *fuzzy* es calculado en base a dos parámetros, que son distancia y orientación relativa frente al destino. Se buscan así enviar los paquetes a los nodos que nos ofrezcan distancias medias y mejor direccionalidad frente al destino. El proceso de inferencia *fuzzy* se realiza hasta que el paquete llega al destino o hasta que su *hop count* se agota y ha de ser descartado.

- **El método *carry and forward*:** se utiliza cuando el vehículo está desconectado de la red vehicular y no tiene vecinos. En este caso el protocolo no se dedica a descartar el paquete, lo guarda un determinado tiempo en caché para intentar un envío posterior.



(a) A source node uses carry-and-forward mechanism to keep packets in unconnected wireless links

(b) A source node switches to SRR protocol when a vehicle has been found within its radio range

Ilustración 7. Modos de funcionamiento del protocolo según escenario

El sistema de decisiones necesita métricas de las que poder obtener las reglas de entradas, utilizaremos como ha sido comentado anteriormente las métricas relativas a la distancia entre el nodo intermedio y el vecino; así como la dirección relativa entre ambos, desde el punto de vista del emisor del mensaje, que será quien haga los cálculos necesarios y "alimente" al sistema de decisiones *fuzzy logic*.

Métrica relativa a la distancia entre el nodo vecino y el destino:

Da mayor valor a los nodos que están a una distancia más intermedia del destino. Podríamos pensar que cuanto más cerca se encuentre el vecino del destino será mejor para el protocolo (de hecho así razonan los protocolos llamados *greedy*).
El hecho de que se valoren más las distancias medias es por las siguientes razones:

1. A mayor distancia entre el receptor y el destino del mensaje habrá un mayor número de saltos que tendrá que dar el paquete hasta llegar al destino. También los nodos muy cercanos al emisor pueden provocar mayores interferencias.
2. Cuando la distancia es menor entre el destino y el receptor puede que nos acerquemos al límite de cobertura, dando pie a una mayor probabilidad de fallo del enlace.

Podría darse el caso en el que un nodo sea vecino en un determinado momento, en el que decidimos enviarle el paquete, y cuando se le envíe este ya no se encuentre en la zona de cobertura. Esto se ve solucionando, una vez mas, dándole prioridad a los nodos que están a una distancia intermedia del nodo origen.

Para el cálculo de distancia entre los nodos he tenido en cuenta que hablamos de coordenadas geográficas, por ello se ha utilizado la fórmula de Haversine[18] que está orientada al cálculo de distancias de este tipo.

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1) * \cos(\varphi_2) * sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R * c$$

Ecuación 1. Fórmula de Haversine donde φ *es la latitud,* λ *es la longitud,* R *es el radio de la tierra (radio = 6,371km)*

Métrica relativa a la dirección relativa:

Calculada entre un nodo intermedio y el destino, respecto a nuestro nodo origen, valora más los nodos que ofrecen un menor ángulo entre el nodo intermedio y el destino. Esto es igual a decir que se valoran más los nodos que están mejor dirigidos hacia el destino, respecto del origen.

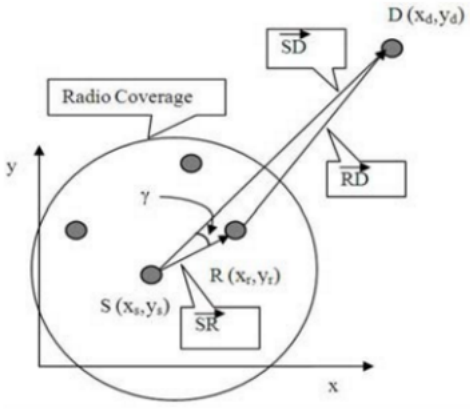


Ilustración 8. Esquema de los nodos involucrados en el calculo de dirección relativa

Como se aprecia en la imagen, para calcular el *fuzzy cost* del nodo intermedio R, necesitaremos obtener la dirección relativa entre el nodo D y el nodo R desde el punto de vista del nodo origen S.

Para ello debemos comenzar calculando los siguientes vectores

$$\overrightarrow{SR} = < (x_r - x_s), (y_r - y_s) >$$
$$\overrightarrow{SR} = < (x_d - x_r), (y_d - y_r) >$$
$$\overrightarrow{SR} = < (x_d - x_s), (y_d - y_s) >$$

Ecuación 2. Cálculo de los vectores necesarios

Una vez los tengamos, podremos proceder a calcular el ángulo que nos dará la dirección relativa buscada:

$$\cos\gamma = \frac{\overrightarrow{SR} * \overrightarrow{SD}}{\left|\overrightarrow{SR}\right| * \left|\overrightarrow{SD}\right|}$$

Ecuación 3. Cálculo de dirección relativa

Combinando ambas métricas podremos crear un sistema de decisión “inteligente”.

Diseño del sistema de decisiones basado en lógica difusa:

Los sistemas llamados de *fuzzy logic*[19] toman decisiones "razonadas" siguiendo un conjunto de *fuzzy rules* o reglas difusas. De esta forma tratan de emular el comportamiento humano a la hora de tomar decisiones, interpretando diferentes tipos de información y actuando en consecuencia.

Este tipo de sistemas se comportan bien en escenarios como sistemas de decisión, procesos de control, estimación y predicción.

En el protocolo el sistema de lógica difusa nos permitirá saber en todo momento qué nodo vecino es el mas indicado para propiciar el enrutamiento el paquete.

'Fuzzyfication' de entradas y salidas:

Una vez calculadas las métricas de enrutamiento mencionadas, pasamos a adaptarlas dentro de un rango más manejable para el sistema de decisión.

Para el grado de dirección este rango va desde el -1 (que indica la menor direccionalidad hacia el destino) hasta el 1 (que indica la máxima direccionalidad alcanzable). Su valor está representado por el coseno del ángulo obtenido.

Para la distancia el rango va desde 0.05 hasta 0.96 y este valor se calcula teniendo en cuenta que tenemos un radio de alcance y todos los nodos vecinos estarán dentro del mismo. Por ello dividiendo la distancia con el radio obtendremos un número que medirá la cercanía o lejanía del nodo al emisor.

$$RDistance = 1 - \frac{D}{R}$$

Ecuación 4. Cálculo del parámetro *RDistance*. R es el radio de alcance *wireless* definido

En base a estos dos parámetros obtenemos el valor *fuzzy* de salida, acotado desde -0.5 a 0.5. Un mayor valor implica mejor ranking del nodo.

Motor de inferencia *fuzzy:*

	IF		**THEN**
Regla	**Distancia**	**Direccionalidad**	**Coste *fuzzy***
1	Lejos	Baja	Bajo
2	Lejos	Media	Medio
3	Lejos	Alta	Alto
4	Intermedio	Baja	Medio
5	Intermedio	Media	Alto
6	Intermedio	Alta	Muy alto
7	Cerca	Baja	Muy bajo
8	Cerca	Media	Bajo
9	Cerca	Alta	Medio

Tabla 2. Reglas del motor de inferencia

Como se puede observar, para cada par de reglas de entrada el motor de inferencia calculará una regla de salida, cuyo valor numérico dentro de cada regla será calculado de la forma que comentaremos más adelante.

Para las direcciones tenemos tres reglas de entrada:

1. Baja: *DegreeDirectness* en el rango [-1,-0.05[
2. Media: *DegreeDirectness* en el rango [-0.05,0.85[
3. Alta: *DegreeDirectness* en el rango [0.62,1]

Del mismo modo, para la direccionalidad tenemos tres reglas de salida:

1. Lejos: *RDistance* en el rango [0,0.35[
2. Intermedio: *RDistance* en el rango [0.35,0.62[
3. Cerca: *RDistance* en el rango [0.62,1]

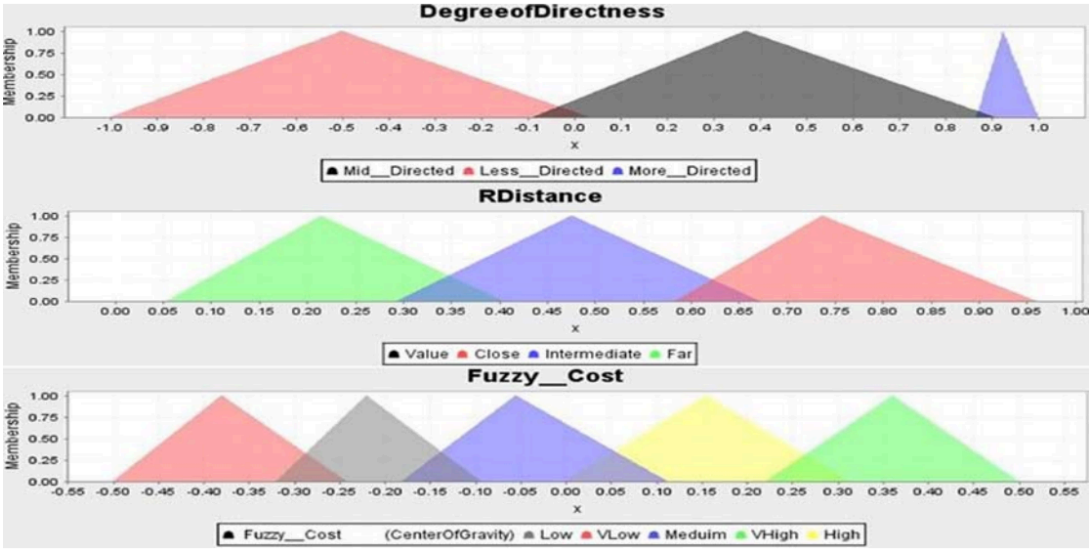


Ilustración 9. Valores de las reglas de entrada y salida en función del valor de sus parámetros

El motor, en base a los valores de distancia y direccionalidad dará un coste *fuzzy*. Por lo visto, los valores *fuzzy* siguen una distribución normal dentro de cada regla de salida. Por ello, se crea un generador aleatorio basado en una distribución normal, que nos de valores numéricos para cada regla de salida, acotado dentro de los rangos establecidos para cada regla de salida, observables en la ilustración 9.

3.2 Aspectos de la implementación

Una vez se han descrito las ideas base sobre las que se va a desarrollar el protocolo empiezo a comentar como he llevado a cabo el trabajo.

El protocolo ha sido implementado utilizando C++ como lenguaje de programación.

La implementación de este protocolo se ha abordado por partes, hemos visto que módulos o clases se necesitaría crear para llegar a tener el protocolo funcionando.

Se empieza viendo que componentes son importantes a la hora de hacer funcionar el protocolo:

- Se precisa de una tabla que almacene datos sobre nodos de la red en cada uno de los nodos, en este caso la llamo *routing table*.
- Es necesario generar un esquema para cada tipo de mensaje a enviar, por ello se crean las clases oportunas.

- Al tratarse de un protocolo de red, son necesarias las funciones para crear sockets y enviar datos por ellos, por ello se crea un fichero definiendo en el todas las funciones para interactuar en red.
- Se necesita, por supuesto, la clase central que implemente el protocolo, que integra todas las clases mencionadas anteriormente y que resuelva con su código cual es el trato que se le da a los paquetes entrantes y salientes.
- Haremos uso de dos hilos para el envío de mensajes HELLO y STATUS de forma periódica. El hilo para enviar STATUS lo utilizo para enviar paquetes DATA cada tantos STATUS enviados, y así poder ver como se comporta el protocolo a la hora de enviar mensajes DATA en nuestro experimento.
- Para intercambiar mensajes por los sockets hemos de tener en cuenta que no podemos enviar los objetos que representan los mensajes por los sockets sin un previo tratamiento de los mismos. Se necesita una serialización de los datos en el emisor y una deserialización de los mismos en el receptor.

3.2.1 Mensajes utilizados

El protocolo hace uso de cuatro tipos de mensajes, cada uno con una funcionalidad bien definida. En este apartado se incide en su estructura y en las funciones que tenemos para tratar con su creación y su paso a la red. Como serán tratados se analizará en otro apartado.

3.2.1.1 Mensaje Hello

Este mensaje es creado con el propósito de que todos los nodos de la red tengan datos del resto como pueden ser su localización, dirección IP o dirección MAC.

La manera que se tiene de que un mensaje HELLO llegue al mayor número de nodos posibles de la red (esto dependerá del *hop count* del paquete, de la calidad de la red y de la densidad de nodos) es realizar lo que se conoce como *flooding* y ya se ha comentado en apartados anteriores.

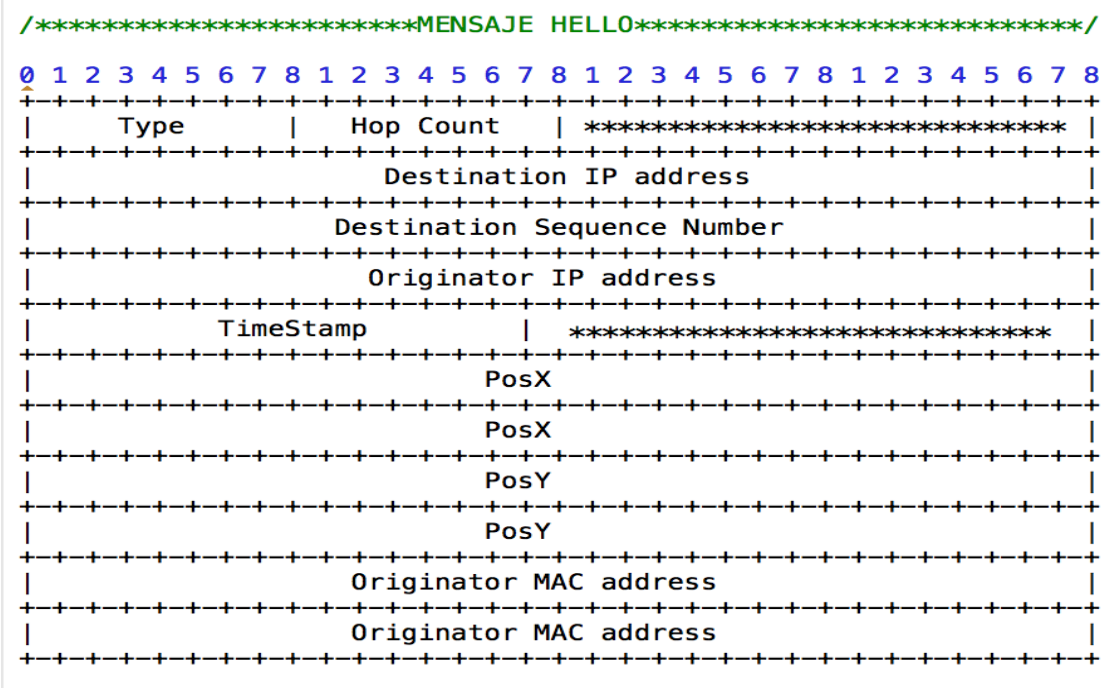


Ilustración 10. Esquema de datos del paquete HELLO

3.2.1.2 Mensaje Status

La finalidad de este mensaje es, dado un nodo emisor, hacer que todos sus vecinos en la red que corran el protocolo sepan características de utilidad para el protocolo. Las características listadas en nuestro paquete son la posición (una vez mas, para que tengamos información sobre la posición de los vecinos más actualizada), la velocidad (con esto podemos calcular la duración potencial del enlace, en algunos protocolos este parámetros es útil y aquí se podría usar también) y el estado de la carretera alrededor del nodo. Se envía en modo *unicast* a cada vecino.

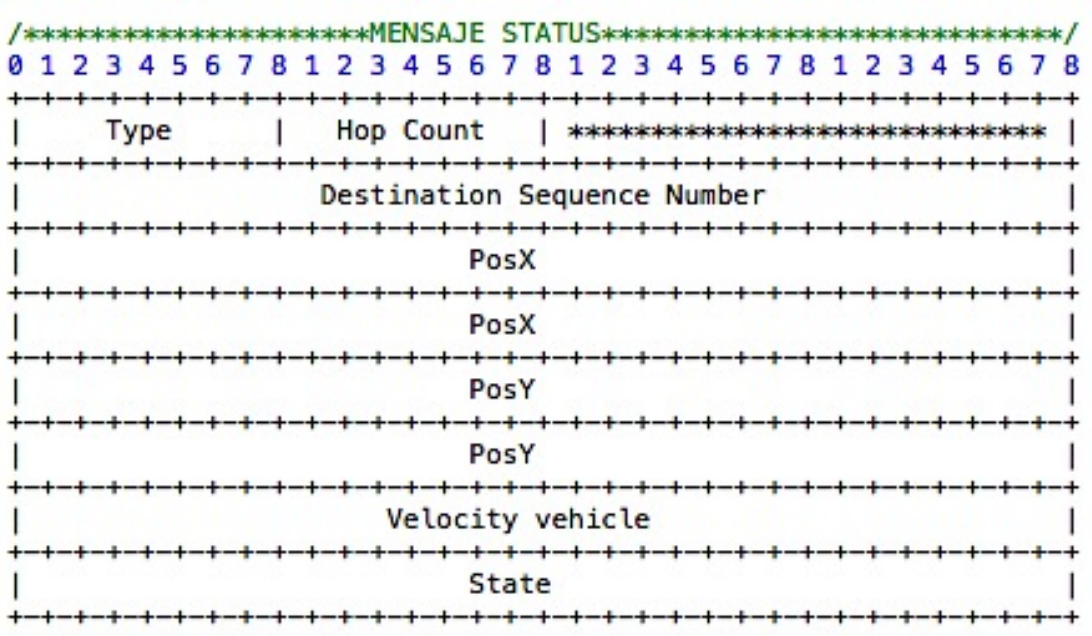


Ilustración 11. Esquema de datos del paquete STATUS

3.2.1.3 Mensaje ACK

El propósito de este mensaje es la verificación de la recepción de los mensajes HELLO, de esta forma el mensaje enviado por *broadcast* recibirá datos de todos los nodos de la red que han recibido el mensaje HELLO. Éste mensaje utiliza las técnicas de *fuzzy logic* para llegar al emisor del HELLO que lo ha originado.

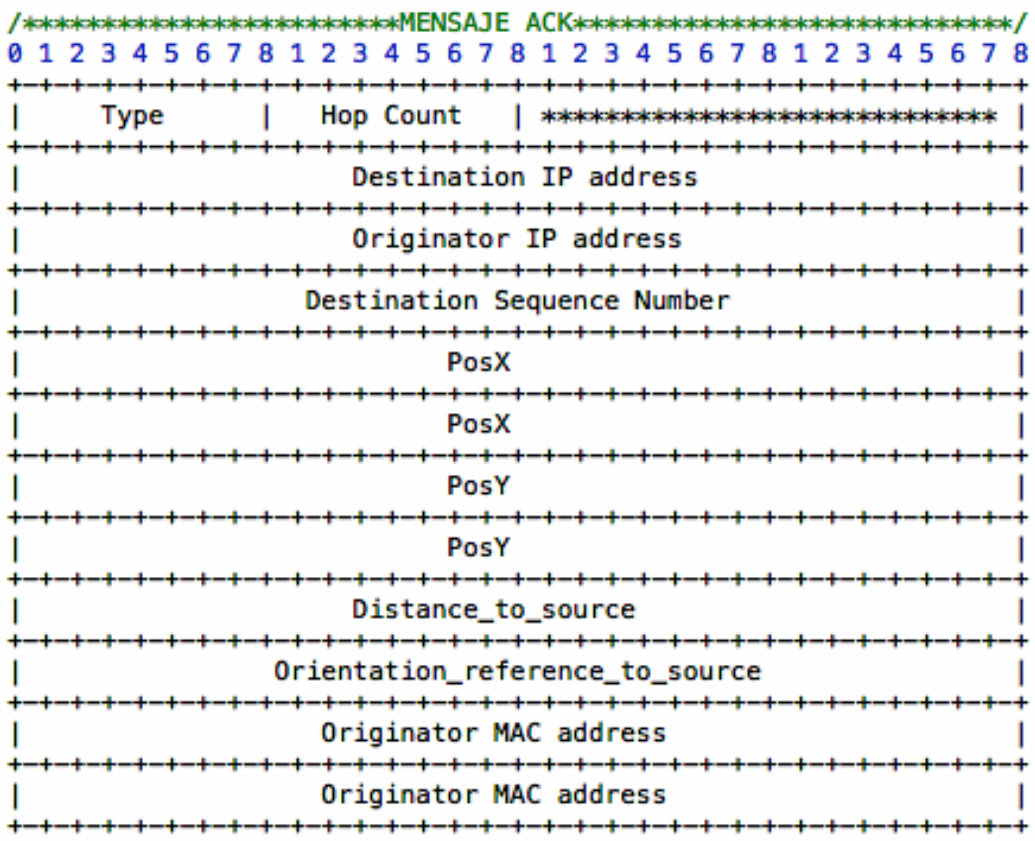


Ilustración 12. Esquema de datos del mensaje ACK

3.1.2.4 Mensaje Data

Éste mensaje tiene como finalidad transportar en su *payload* los datos que queramos que sean recibidos en otros nodos. Una vez que el nodo tenga datos de los vecinos podrá enviar estos mensajes mediante las técnicas de *fuzzy logic* que calcularán una buena ruta para los mismos.

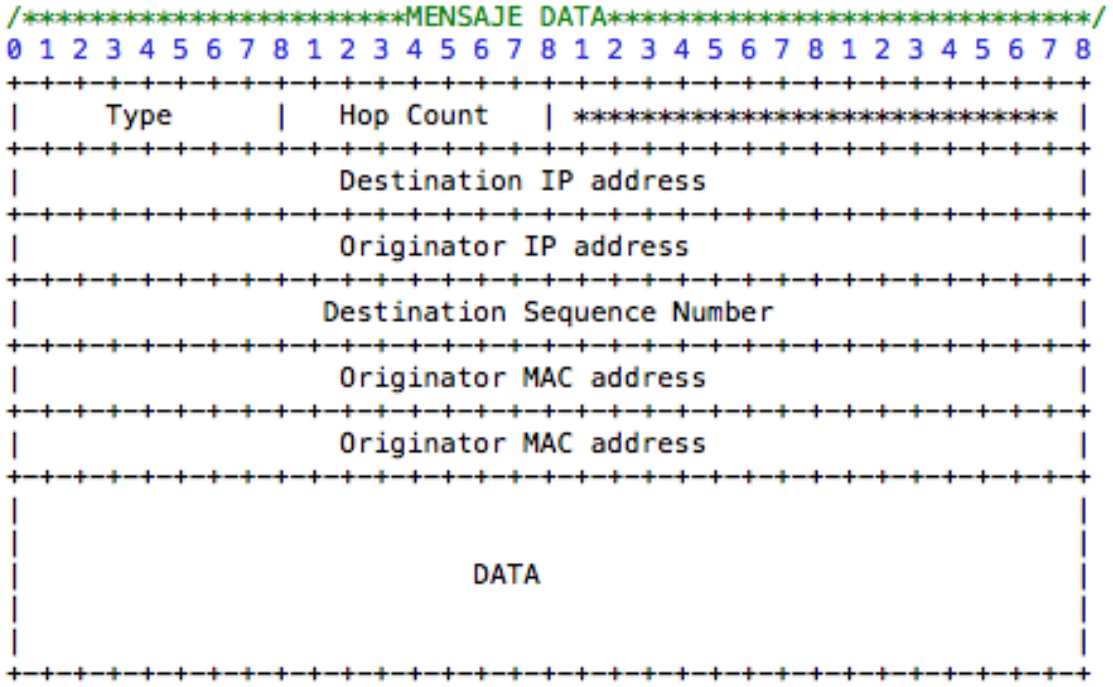


Ilustración 13. Esquema de datos del mensaje de datos

3.1.2.5 Funciones asociadas a los mensajes y parámetros contenidos

Todos estos mensajes cuentan con funciones asociadas que nos servirán para lo siguiente:

- **Funciones para serializar un objeto mensaje** del tipo a tratar y convertirlo en un *string* de las librerías de C++ que podrá ser convertido en *stream* de *bytes* para ser enviado sin problemas por un *socket*.
  Ejemplo: string serializa_hello(HelloHeader Hello)

- **Función para deserializar un stream de bytes que venga de la red** que sepamos que contiene un mensaje del tipo a tratar. Así transformamos el *buffer* de *bytes* proveniente del *socket* en un objeto del mensaje correspondiente en el receptor que podrá interpretarlo.
  Ejemplo: HelloHeader deserializa_hello(char * buffer)

- **Función para mostrar por pantalla el contenido del mensaje** a tratar.
  Ejemplo: void MuestraHello(HelloHeader hello)

- **Función para crear un mensaje con parámetros** asignando el valor que creamos necesario.
  Ejemplo: DataHeader creaData(uint8_t hop_count, uint32_t dest_IP, uint32_t orig_IP, uint32_t destination_sequence_number,uint64_t mac, char * message)

Podemos observar también los siguientes campos en los paquetes anteriores:

- ***Type:*** Campo para indicar al protocolo a la hora de recibir mensajes que tipo de mensaje ha de tratar, este campo es común a todos los paquetes. Es necesario porqué en función del mensaje que se reciba el programa deberá realizar distintas acciones.
  Los tipos de mensajes definidos son: HELLO (tipo 0), STATUS (tipo 1), ACK (tipo 2), DATA (tipo 3).

- ***Hop count:*** Indicará a la hora de recibir cuantos reenvíos por la red puede soportar este mensaje, el TTL de cada paquete estará marcado en tiempo de compilación y en nuestro caso es 10. Cada vez que el paquete salte de un nodo a otro se verá decrementado, hasta llegar a cero, tiempo en el que dejará de ser enviado a otros nodos. Éste atributo también es común al resto de paquetes.

- ***Destination/Originator IP address:*** Este campo permite al protocolo saber, para el paquete que los contiene, la dirección IP del destino al que tiene que llegar el paquete así como la dirección del nodo que lo ha creado y enviado en primera instancia.
  En el caso de los paquetes HELLO la dirección IP de destino será la dirección de *broadcast* de la red en la que se encuentre el emisor. Todo paquete que pueda dar mas de un salto en nuestra red deberá contar con esta dupla de parámetros, ya que los nodos intermedios deben saber a quién tiene que llegar el paquete, y el destino necesita saber quien fue el emisor del mismo.

- ***Destination sequence number:*** Éste número servirá para identificar a un paquete dentro de la red de forma única mediante la dupla que forma junto a la dirección IP del emisor. Teniendo identificado así cada paquete HELLO podremos hacer frente a problemas como *loops* (paquetes que se envían en bucle sin llegar al destino final) o la aceptación de datos de paquetes antiguos que provienen de usar una técnica como el *flooding* para el envío de datos. Sólo se aceptarán paquetes con números de secuencia más elevados que el número de secuencia que tenemos para el nodo emisor en nuestra caché. Esto es así porque a cada envío se incrementa en uno este valor, indicando así un valor más alto que el paquete contiene información actualizada.
  Todos los paquetes cuentan con este campo para poder tener un registro de los paquetes enviados y recibidos a la hora de analizar estadísticas del protocolo, pero donde realmente es útil para el buen hacer del protocolo es en los mensajes HELLO.

- ***Timestamp:*** Este campo es usado por el protocolo para detectar paquetes HELLO que lleguen con un retardo no deseado el destino, que tendrán que ser descartados.

El tema del descarte de paquetes antiguos es tratado también con los números de secuencia, pero no es suficiente cuando un nodo "desaparece" de la red y de la *routing table* de un destino que recibe este paquete. El nodo al que llega el mensaje no tiene en ese momento información del nodo, y por ello acepta el paquete y lo procesa como bueno; sin embargo, con este *timestamp* podremos saber si el paquete ha pasado su período de validez (definido en función del tiempo que dura una entrada en nuestra *routing table*) y por ello debe ser descartado.

- ***PosX/PosY:*** Estos campos nos dan la latitud y la longitud del nodo que crea el mensaje, parámetros necesarios en todos los nodos de la red que puedan participar en el enrutamiento. En esta implementación no se ha hecho uso de un módulo GPS para obtener coordenadas, para realizar las pruebas se han usado coordenadas estáticas previamente seleccionadas. En el apartado 7 se comenta este hecho. Todos los paquetes, exceptuando el de datos, contienen estos parámetros referidos a su emisor.

- ***Velocity vehicle:*** Con este parámetro de un nodo podremos saber (en base a la distancia máxima para el buen funcionamiento del enlace y a la dirección relativa del movimiento entre dos nodos cuanto tiempo) cuanto tiempo se estima que el nodo destino sea un buen receptor del mensaje.

- ***Distance to source:*** Parámetro contenido en los ACK que calcula la distancia del emisor respecto al nodo que lo envía.

- ***Orientation referenced to source:*** Parámetro también contenido en los ACK que nos marca cual es la orientación relativa entre nodos emisor y destino del ACK.

- ***Originator MAC address:*** Debido a los problemas obtenidos que se tuvo a la hora de filtrar que nodos deberían ser vecinos y que nodos no (algunos tenían baja calidad de enlace y no deberían ser elegidos nunca como vecinos posibles ya que los paquetes raramente les llegaban mediante envío directo), fue necesario obtener la dirección MAC de la "NIC" o *Network Interface Card* que estaba usando el protocolo para enviar y recibir datos. La utilidad de esto es mostrada en el apartado referente a la tabla de vecinos del fichero "*neighborTable.h* ".

- ***State:*** En implementaciones finales este parámetro puede representar el estado de la calzada alrededor del vehículo emisor, información que puede ser de mucha utilidad en aplicaciones para redes vehiculares.

3.2.2 Funciones del protocolo implementado

El sistema hace uso de múltiples funciones, todas imprescindibles para modelar su comportamiento final. Previa a la compilación se encuentran varios ficheros, con los contenidos que podemos observar en los apartados siguientes:

3.2.2.1 Fichero serializacion.h

Contiene las clases que representan los distintos tipos de mensaje intercambiados por el programa y las funciones relacionadas con los mismos.

Ya se ha mostrado en el apartado anterior cuales son las funciones asociadas a cada tipo de mensaje, y en que consiste la serialización y la deserialización de datos.

Esta serialización/deserialización es necesaria tenerla implementada previamente a transmitir los paquetes en red, ya que según el ordenador en el que estemos los bytes de datos pueden interpretarse de distintas formas si no pasan por este proceso (pueden darse problemas de distinto *endianness* en la forma de interpretar la información y problemas con el *padding* de los datos que harían que dos máquinas interpretasen la información de distinta forma).

3.2.2.2 Fichero localizacion.h

En este fichero encontramos las clases representando al GPS y la IMU. Están implementadas como una serie de *"getters"* y *"setters"* de los atributos que tienen que utilizo para obtener y establecer las posiciones ficticias de cada nodo para hacer pruebas.

Se puede ampliar el trabajo mediante la adquisición de un módulo GPS a usar en la *Raspberry*, en ese caso en esta clase irían todas las funciones destinadas a la obtención de datos del módulo GPS y a la configuración del mismo. También se podría utilizar la IMU para saber en que dirección y sentido se está moviendo un nodo. En ese caso las funciones relacionadas con la obtención de datos y configuración de la IMU también irían en este fichero.

Como no hay disponibilidad para utilizar un módulo GPS y tampoco para usar una IMU no se ha realizado la implementación de esta forma y se comprobará como funciona el sistema con coordenadas estáticas que previamente han sido obtenidas de *Google Maps*.

3.2.2.3 Fichero clientesServidores.h

Este fichero cuenta con todas las clases referentes al envío y la recepción de paquetes en red, de esta forma se abstrae al programador en la creación de sockets válidos para enviar datos.
Se cuenta con las siguientes funciones:

- **int creaServidor(uint32_t ip, uint16_t puerto)**
  Abre un socket y realiza un *bind* en la dirección IP y puerto que se nos dice en los parámetros. Devuelve el descriptor del socket abierto en nuestro sistema, que deberá ser utilizado como indicador del socket para futuros envíos y recepciones.

- **void envia(int socket, uint32_t ip_servidor, uint16_t puerto, string msg)**
  Envía el mensaje "msg" a la dirección y puerto marcados en los parámetros, mediante el uso del *socket* demandado. El *string* será convertido dentro de esta función a tipo "char *" usando la función "msg.c_str()" para poder ser enviado por el socket.
- **uint64_t mac_addr_interfaz(int fd, char * interfaz)**
  Función utilizada por el programa para conocer en tiempo de ejecución la dirección MAC de 48 bits de la interfaz que estamos utilizando, dato vital para el funcionamiento del protocolo.
- **uint32_t obten_ip_interfaz(char * interfaz)**
  De la misma forma, podemos conocer a tiempo de ejecución la dirección IP que tiene asignada la interfaz que vamos a usar para el intercambio de datos, otro dato vital para el protocolo.
- **uint32_t obten_broadcast_interfaz(char * interfaz)**
  Nos dará la dirección de *broadcas*t de la red a la que pertenece la interfaz que marquemos, esto será utilizado para el envío de mensajes HELLO.
- **string recibe(int socket, uint32_t * sourceAddress)**
  Con esta función haremos frente a la recepción de datos por el socket pasado como parámetro; además, en el puntero a la variable *sourceAddress* guardaremos el valor correspondiente a la dirección IP del emisor del mensaje, gracias al uso de la función *recvfrom*.
- **int cierraSocket(int socket)**
  Con esta función cerraremos el socket del que hemos pasado como parámetro su descriptor .

3.2.2.4 Fichero FuzzyCalculator.h:

Contiene las funciones usadas por el protocolo para saber que nodos vecinos son más adecuados para transmitir el paquete hasta su destino final.

Las decisiones se toman basando al sistema de decisión en parámetros como la distancia y la orientación relativa entre los nodos emisor, intermedio y receptor. Estos parámetros actúan como entrada del sistema de lógica difusa que nos dice que nodo es el vecino más favorable para recibir el paquete.

El sistema de decisiones de lógica difusa, en base a las reglas de entrada nos dará una regla de salida, cuantificable mediante [16, figura 6], aquí se ha realizado una aproximación de las reglas de salida del sistema difuso, el valor cuantificado se calcula mediante un generador de números aleatorios que seguirá una distribución normal dentro del rango [mínimo, máximo] asociado a la regla (ya que los valores dentro de cada regla siguen esa distribución en el ejemplo, como se ve en la ilustración 8).

En este fichero tendremos la definición del "struct nodo" que usaremos para crear las entradas de nuestra *routing table*, además de una función para asignar coordenadas a un struct nodo que me fue de utilidad para realizar pruebas.

También vemos aquí la clase *fuzzyCalculator* que cuenta con una función pública usada

para devolvernos el coste de la regla de salida asociada al nodo intermedio que será el que busquemos en el protocolo constantemente:

- **double costeFuzzy(struct nodo nodoInicio, struct nodo nodoIntermedio, struct nodo nodoDestino)**
  Ésta será la función que utilizará el protocolo para poder encontrar a su mejor vecino. Hará uso de las funciones que se presentan a continuación para obtener el parámetro que indica la calidad del nodo intermedio como nodo enrutador de información.

Como funciones privadas, usadas por la función "*costeFuzzy*" para obtener el resultado buscado, tenemos:

- **double genera_random_normal(double min, double max)**
  Generador de números aleatorios siguiendo distribución normal.
- **float calculaDistanciaEntreNodos(struct nodo NodoInicio, struct nodo NodoDestino)**
  Calcula la distancia entre los nodos pasados como parámetros usando la formula de Haversine.
- **double calcula_RDistance(float distancia_nodos, double radio_alcance)**
  Calcula el valor *Rdistance*, en función del radio máximo de alcance *wifi* y de la distancia entre los nodos. Su valor es necesario para el motor de inferencia de lógica difusa.
- **float calculaOrientacionRelativa(struct nodo NodoInicio, struct nodo NodoDestino)**
  Calcula la orientación del nodo intermedio con el destino, desde el punto de vista del emisor.
- **double calcula_DegreeDirectness(double angulo)**
  Calcula el valor *degree of directness* necesario como entrada del motor de inferencia de lógica difusa.
- **double funcion_fuzzyCost(double Rdistance, DegreeDirectness)**
  Calcula el valor *fuzzy* de salida usando la inferencia de lógica difusa mediante los dos parámetros de entrada.

3.2.2.5 Fichero neighborTable.h

Este fichero cuenta con una clase llamada *neighborTable* que contiene a todos los nodos que están dentro del alcance de la señal de nuestro nodo, con una calidad mínima fijada en tiempo de compilación (con una potencia de señal RSSI -72 dB los datos se transferían sin problemas).
Las funciones que nos facilita esta clase son:

- **uint64_t mac_number(unsigned char mac[9])**
  Esta función tiene la utilidad de *parsear* un *string* (analizar una cadena de texto en busca de patrones) correspondiente a la *MAC address* de una determinada interfaz y transformarla a su valor numérico de 48bits.

- **void actualizaVecinos()**
  En las *Raspberry* tenemos la utilidad "iw" que nos da información obtenida a partir de la interfaz *wireless* que le ordenemos.

  Entre esta información, tendremos la posibilidad de obtener las *MAC address* de dispositivos vecinos que tenemos dentro de la red *wireless* a la que pertenece cierta interfaz. Junto a la información referente a la MAC, también tenemos la calidad de la señal que nos llega en el instante que ejecuta el comando.

  El comando mencionado es el siguiente:
  iw dev wlan0 station dump |grep 'Station|signal:'

  Aquí se nos darán dos líneas para cada vecino, llamando a la función desde el programa podremos analizarlas y sacar de ellas los valores referentes a la MAC del vecino y a su calidad de señal. Cada entrada de la tabla tendrá un struct con estos dos valores. Sólo se añadirán a la tabla aquellos vecinos que cumplan con el mínimo de calidad de señal determinado.

- **vector<uint64_t>mac_vecinos()**
  En esta función se llama a actualizaVecinos() para obtener los vecinos en ese instante y después se recorre la tabla y se van añadiendo las MAC de la misma al vector a devolver. Usada en la clase *routingTable* para poder devolver un vector con las direcciones IP de los nodos que sean vecinos.

- **bool esVecino(uint64_t mac)**
  En esta función se buscará la dirección MAC pasada entre los vecinos, si se encuentra devuelve true, si no false.

- **void listaVecinos()**
  Todas las funciones son públicas, lo único privado que tiene es la tabla, implementada con un "std::map" de C++.

3.2.2.6 Fichero routingTable.h

En este fichero se encuentra la clase más importante, integra el sistema de decisión de lógica difusa, el tratamiento de paquetes, la obtención de los nodos vecinos y de aquellos sobre los que nos va llegando información mediante la inundación de mensajes HELLO.

La tabla en está implementada utilizando un "std::*map*", indexado por la dirección IP de los nodos y conteniendo para cada entrada un "struct" nodo con los datos que son importantes para el funcionamiento del protocolo.

Los datos que tenemos para cada nodo serán:

- Latitud, longitud
- Velocidad
- Orientación
- Número de secuencia

- Una variable de tiempo (*time_t*) que marca el momento en el que la entrada fue añadida a la tabla o actualizada
- La dirección MAC asociada al nodo

En esta clase tenemos las siguientes funciones:

- **struct nodo busca_por_ip(uint32_t ip)**
  Esta función busca en la tabla el nodo asociado a la dirección IP que se le pasa como parámetro, de no encontrarlo, devuelve un nodo con todas sus variables a cero.

- **void crea_actualiza_entrada(HelloHeader hello)**
  void crea_actualiza_entrada(uint32_t ip_origen ,StatusHeader status)
  void crea_actualiza_entrada(ACKHeader ack)
  Estas tres funciones, según el paquete obtenido por el socket, nos permiten extraer del mismo la información relevante sobre el nodo y añadirla a la *routing table*. Como indica el nombre pueden actualizar una entrada o crearla, todas buscan la entrada asociada, cambian los parámetros necesarios de la misma y la vuelven a añadir. Esto es posible gracias a la función busca_por_ip que en caso de no encontrar la entrada nos devuelve un nodo vacío, al que se le añaden los nuevos datos.

- **bool esVecino(struct nodo aux)**
  Esta función extrae la información sobre la dirección MAC asociada a la entrada a evaluar, posteriormente se hace un llamamiento a la función esVecino de la *neighbor table* usada para comprobar si efectivamente el nodo a evaluar está dentro de nuestro alcance *wireless* y cuenta con la potencia de señal suficiente como para ser un potencial receptor directo de paquetes.

- **void CleanOutdatedRTEntries()**
  Itera sobre la tabla para buscar entradas que hayan expirado su validez (recuerdo que para cada entrada referida a un nodo teníamos una variable *time_t* que se refrescaba al realizar cambios sobre la entrada). Si el *difftime* (función usada para calcular el tiempo transcurrido entre dos variables *time_t*) supera el máximo permitido (definido por el parámetro *TTLRoutingTable*), se procede a borrar la entrada.
  De esta forma se eliminan las entradas antiguas e inservibles. Esta función es llamada cada vez que realicemos una consulta a nuestra tabla, así tendremos la seguridad de que la entrada consultada es válida.

- **vector<uint32_t> ips_tabla()**
  Nos dará un vector con todos los nodos que puedan considerarse vecinos, esto será usado por el hilo que emite los mensajes STATUS a todos los vecinos.

- **vector<uint32_t> ips_sinfiltro_vecino()**
  Aquí se nos dan todas las direcciones IP para las que tenemos entrada en la tabla.
  La uso en el experimento cuando quiero enviar a varios destinos mensajes

DATA, sin importar si son o no vecinos.

- **uint32_t mejor_vecino(uint32_t ip_destino, double my_pos_X, double my_posY)**
  Es de lejos la función más importante para nuestro protocolo, nos devolverá la dirección IP del nodo intermedio al que deberemos enviar un paquete siguiendo las indicaciones del protocolo.

  El flujo de ejecución de esta función es el siguiente:

  Creamos un nodo conteniendo nuestras coordenadas, buscaremos el destino en la *routing table* usando la función busca_por_ip para obtener las coordenadas del mismo. Si no tenemos datos de ese nodo la función mejor_vecino indicará que ha habido un error. En caso de tener datos pasaremos a comprobar si el nodo es vecino haciendo uso de la función esVecino, de ser vecino ya tendríamos un valor de retorno, la dirección IP a la que enviar el mensaje sería la dirección IP del propio destino.

  En caso de que el destino no sea vecino, habiendo asignado su entrada a una variable "struct nodo nodoIntermedio" nos tocaría iterar por la *routing table* para obtener al mejor vecino.

  Para cada entrada iterada representando un posible nodo intermedio, realizamos una comprobación mediante la función esVecino para poder saber si el nodo asociado a la entrada puede recibir el paquete, de poder recibirlo, se utiliza la función de la clase *fuzzyLogic* que nos da el coste *fuzzy* del nodo intermedio.

  Tenemos una variable que nos guardará el mejor coste obtenido y la dirección IP que lo ha conseguido, cuando se acaba de iterar se devuelve la dirección IP del nodo que mejor resultado ha obtenido.

3.2.2.7 Fichero routingProtocol.h:

Este fichero integra todos los mencionados anteriormente y hace uso de todas las funciones y clases comentadas para conseguir el objetivo de proporcionar el enrutamiento de los paquetes mediante el uso del sistema de decisión.

En este fichero tenemos, además de las cabeceras necesarias, un *"#define*" para el puerto que usará el protocolo para comunicarse con el resto de nodos, se le asignó el puerto 1234.

También contamos con otro *"#define*" que nos marcará el TTL de los paquetes, el número de saltos que podrá dar cada paquete por la red hasta ser descartado, en este caso se le fijó un valor de 10.

El resto de fichero está compuesto por la clase *routingProtocol* del que se comentan a continuación sus ficheros y funciones a grandes rasgos:

En cuanto a sus atributos:

- **chrono::time_point<chrono::system_clock>startProgram**
  chrono::time_point<chrono::system_clock>helloInterval
  Atributos que marcarán los puntos en los que se pone el protocolo a funcionar y el tiempo en el que se envió el último HELLO (necesario para su envío periódico).

- **GPS datosGPS**
  Instancia de la clase GPS, a la que podremos realizar el set de las coordenadas a probar y de la cual podemos obtener las coordenadas de cierto instante.

- **double tasa_hello, tasa_status, tasa_data**
  Estos mensajes marcarán el intervalo en el que se tendrán que emitir paquetes de cada uno de los tipos a los que referencian.

- **double numEnvios**
  Para saber a cuantos nodos de la red debe enviar paquetes de datos nuestro nodo, lo utilicé para ir enviando mensajes DATA de forma automática y realizar pruebas.

- **uint32_t sequence_number**
  Valor que marca el número de secuencia del siguiente mensaje HELLO a enviar.

- **uint32_t data_sent**
  Valor que marca el número de paquetes de datos que llevamos enviados, también actúa como número de secuencia de los paquetes de datos.

- **thread *helloThread, *statusThread**
  Referencias a los hilos de ejecución que lanzaremos para enviar mensajes STATUS y mensajes HELLO.

- **Variables para objetos de cada tipo de mensaje**
  Dos de cada tipo, una para los mensajes a enviar y otra para los mensajes recibidos.
  Ejemplo: **StatusHeader statusMessage, statusMessage_recibido…**

- **RoutingTable routeTable**
  Instancia de la clase en la que guardaremos la información sobre los nodos de la red de los que nos vaya llegando información.

- **int TTLRT**
  Con esta variable puesta en función de la tasa de envío de mensajes HELLO, podremos hacer un set en la *routing table* que modifique el tiempo durante el que sus entradas serán válidas.

- **uint32_t sourceAddress**
  Este valor nos indicará, en la función destinada a recibir información, quien nos ha enviado el paquete que nos ha llegado en el momento de recibir.

En este fichero, hacemos uso de las siguientes funciones, que modelan el comportamiento final que exhibe el protocolo.

- **routingProtocol(double tasa_envio_hello, double tasa_envio_status, int tasa_envio_data, int numeroEnvios)**
  Creará un objeto *routingProtocol* con los parámetros pasados como argumento. Esta función es la que se llama desde el "main()" de nuestra aplicación para crear una instancia del protocolo mediante parámetros pasados por el usuario, al que se le han solicitado previamente por consola.

- **void coordenadasMAC(uint32_t ip)**
  Ésta función la creé para asignar las coordenadas que tenia miradas para cada nodo de forma estratégica en función de donde estuviese colocado el nodo, como la dirección IP asignada a la NIC (*Network Interface Card*) de cada nodo era estática, la usé para en función de esta saber que nodo estaba ejecutando el protocolo y asignarle las coordenadas que le tocaban.

- **double difftime_preciso(**
  chrono::time_point<chrono::system_clock> fin,
  chrono::time_point<chrono::system_clock> inicio)
  Función creada con la finalidad de devolver la diferencia de tiempo entre los dos puntos marcados en segundos con decimales. La función *difftime* que usaba variables *time_t* siempre redondeaba a segundos, y en el protocolo necesitaba mayor precisión a la hora de medir diferencias de tiempo.

- **void routingProtocol::runThreaded()**
  Esta función se encarga de poner todos los hilos de la aplicación en marcha, creamos dos hilos extra a parte del principal.
  Los hilos extra ejecutarán las funciones *SendHello* y *SendStatus* comentadas mas adelante. El hilo principal pasa a ejecutar *RouteInput*, recibiendo y tratando los mensajes que nos van llegando.

- **void RecvSRR()**
  He usado esta función como la que inicia el funcionamiento del protocolo después de haber creado la instancia del mismo. La función realiza las siguientes tareas:

  1. Guarda en el atributo *startProgram* el instante de tiempo en el que hemos iniciado el funcionamiento.
  2. Obtiene las direcciones IP, MAC y de *broadcast* relativas a la interfaz que va a usar el protocolo (en mi caso wlan0), haciendo uso de las funciones del fichero "clientesServidores.h"
  3. Crea el socket por el que enviaremos y recibiremos la información.
  4. Llama a la función coordenadasMAC y posteriormente hace un *"set"* al objeto "datosGPS" para tener asignadas las coordenadas obtenidas.

5. Posteriormente se utiliza la función que redirige la salida de error a un fichero para poder tener un registro de lo que pasa en el protocolo mediante el volcado de información a "clog".
6. Se escribe por primera vez en el fichero, con datos relativos al inicio del experimento.
7. Se lanzan todos los hilos del programa a ejecución mediante el uso de la función *runThreaded()*.

- **void RouteInput()**
  Esta función se encargará de ir mirando el socket en busca de mensajes recibidos y de tratar el tipo de paquete que nos vaya llegando. Después de comprobar que el socket está activo, pasa a funcionar en bucle exhibiendo el siguiente comportamiento.

  1. Recibe el mensaje usando la función recibe de "clientesServidores.h"
  2. Mira que la dirección de origen del paquete no sea la de nuestro propio nodo, de ser así se descartaría el mismo y se pasaría a recibir otro.
  3. Con la *string* de datos que nos ha devuelto el recibe que representa el mensaje obtenemos el primer carácter, representando al tipo de mensaje.
  4. Pasamos el tipo de mensaje obtenido a un *switch*, que dependiendo el tipo de mensaje que hayamos recibido tendrá un flujo de ejecución distinto.
     4.1. En el caso de recibir un mensaje HELLO se deserializa en su variable *"helloMessage_recibido"* y se deja el resto a la función *"RecvHello"* de esta clase, de la que hablo mas adelante.
     4.2. En el caso de recibir un STATUS se deserializaría el mismo en la variable *"statusMessage_recibido"*, se actualizaría la *routing table* utilizando su función de clase "crea_actualiza_entrada" correspondiente y se dejaría constancia de la recepción en el fichero destinado a *"logging"* de la aplicación.
     4.3. En el caso de recibir un ACK se deserializaría el mismo en la variable *"ackMessage_recibido"* y se dejaría el resto a la función *"RecvACK"*.
     4.4. Si lo que recibimos es un mensaje de datos, se deserializa el mismo en su variable *"dataMessage_recibido"* y se deja el resto a la función *"RecvData"*

- **void RecvHello()**
  Cuando recibimos un mensaje HELLO por el socket, precisamos de un tratamiento especial para este tipo de mensajes. Queremos conseguir que los mensajes HELLO lleguen a la mayor cantidad posible de nodos dentro de la red mediante el uso de la técnica conocida como *flooding*, y es en esta función donde reside la clave de la misma.

  Lo primero que se hace al entrar en esta función es decrementar el *hop_count* del paquete que hemos recibido, si llega el momento de realizar un reenvío del mismo miraremos que el valor decrementado no sea cero.

Se extraen del paquete los datos asociados a la dirección IP del emisor y al número de secuencia del paquete, estos dos datos nos identifican de forma única al paquete.

Una vez tenemos estos datos pasamos a ver si tenemos datos sobre el nodo emisor del paquete usando su IP de origen como parámetro de la función busca_por_ip de nuestra *routing table*.

Si hemos encontrado datos del nodo en nuestra tabla pasaremos a comprobar que el número de secuencia que tenemos en la tabla sea menor que el que contiene el paquete. Si el número del paquete es mayor que el de la tabla implica la aceptación del paquete, ya que el mismo contiene información más novedosa. Si el número es menor descartaríamos el paquete.

El primer número de secuencia para un mensaje HELLO que genera el programa es '1', la función busca_por_ip devuelve un nodo con valor '0' en ese campo que indicaría que el nodo no se ha encontrado en la tabla. Por ello el paquete sería aceptado por el protocolo en caso de no encontrar el nodo, ya que su número de secuencia contenido siempre será mayor que '0'.

También comprobamos antes de seguir que el paquete no haya sido originado por nuestro propio nodo, de ser así lo descartaríamos.

El último filtro que tiene que pasar el paquete antes de ser aceptado para ser obtenida su información lo puse al darme cuenta de que si el emisor del mismo se desconectaba de la red, su entrada de todas las tablas en los nodos acabaría borrándose, y que posiblemente algunos antiguos paquetes HELLO suyos (que llegan con un retardo mayor del esperado) siguiesen por la red. El nodo que no tiene entrada para el emisor en ese momento aceptaría el paquete, y además propiciaría el reenvío del mismo, haciendo creer a todos los nodos que el nodo emisor sigue activo.

Se ha hecho frente a esta situación con el *timestamp* añadido en el paquete, mediante dicho *timestamp* veo si el paquete supera el tiempo "TTLRT" que dura una entrada en la tabla, y si lo supera significa que el paquete es antiguo y debe ser descartado.

Pasados todos los filtros podrá ser aceptado el paquete y añadida su entrada a la tabla mediante el uso de la función "crea_actualiza_entrada" correspondiente, además se dejará constancia de la recepción en el fichero de registro.

Una vez añadido, pasamos a proceder con el reenvío del mismo, comprobamos que todavía su *hop_count* sea mayor que cero y si esto se cumple procedemos a serializar el mismo para ser enviado a la dirección de *broadcast* de la red. De esta forma todos los vecinos del nodo intermedio lo podrán recibir.

También enviaremos un ACK correspondiente al HELLO recibido, haciendo uso

de la función *sendACK*.

- **void SendACK()**
  Teniendo en cuenta que esta función se llama en la misma iteración que recibimos un mensaje HELLO, podemos extraer la dirección IP de destino del ACK buscándola en el paquete *helloMessage_recibido*.

  Teniendo esta información, haremos una llamada a la función "mejor_vecino" de nuestra *routing table*.

  Si la función, haciendo uso de las técnicas de inferencia mediante lógica difusa ha podido encontrar al mejor vecino, ya tendremos ruta de salida para nuestro paquete. En caso de no haber vecinos, se procedería a su encolado.

  Añadiremos en el los campos necesarios, como son nuestra posición, distancia, orientación, número de secuencia (que será el mismo que en el HELLO recibido), direcciones origen y destino y *hop_count*.

  Teniendo el paquete creado se procede a su serialización y envío por el socket, dejando constancia de ello en el registro.

- **void RecvACK()**
  Cuando el mensaje que nos llega es un ACK lo primero que realizaremos será una disminución de su *hop_count*. Después veremos si el paquete está originado por nuestro nodo para descartarlo en caso de ser así.

  Si pasa ese filtro comprobaremos si nuestro nodo es el destino del mensaje, si lo es pasaremos a usar la función "crea_actualiza_entrada" correspondiente y dejaremos constancia en el registro de la recepción.

  Si nuestro nodo no es destino deberemos comprobar que el *hop_count* del paquete sea mayor que cero para poder continuar con su ruta hacia el destino mediante la búsqueda del mejor vecino haciendo uso de la función mejorVecino de la *routingTable*.

  Una vez mas, si no encontramos vecinos el mensaje se encolaría, en el caso contrario se enviaría el mensaje serializado por el socket y se dejaría constancia de ello en el registro.

- **void SendData(uint32_t ip_destino, char * mensaje)**
  Primero buscará el mejor vecino para llegar al nodo de dirección "ip_destino". De no encontrarlo se encolaría el mensaje.

  Si lo encuentra se confecciona el mensaje DATA con la función "creaData" del fichero serializacion.h asignándole el mensaje que tenemos pasado como argumento.

  Este mensaje se serializará y se enviará por el socket, dejando constancia de ello

en el registro. También se incrementará la variable "*data_sent*" usada como identificador de los paquetes junto con la IP de origen.

- **void RecvData()**
  Esta función tiene un comportamiento análogo a la función *RecvACK()*, pero en este caso con los mensajes de tipo DATA.

- **void SendHello()**
  Esta función se lanza en un hilo a parte. Lo primero que realiza al lanzarse es una comprobación de que el socket del programa se encuentra activo y listo para enviar y recibir datos.

  Una vez se ha comprobado esto se inicializa la variable *chrono::time_point<chrono::system_clock> helloInterval*, que representará el último instante en el que se envió un mensaje HELLO. Esta variable se inicializará y actualizará mediante el uso de *chrono_system_clock::now()* que nos devuelve el valor del *"time_since_epoch"* actual.

  Acto seguido crearemos el mensaje HELLO inicial para el programa con el comando creaHello. Los parámetros que contendrá a destacar son:
    - El número de secuencia incrementado a cada HELLO enviado.
    - El *timestamp*, que consistirá en una llamada a time(NULL) devolviendo un *time_t* representando la hora actual.
    - La dirección MAC y la dirección IP del nodo que lo origina.

  Una vez creado se serializa el mensaje y se envía a todos los nodos vecinos, ya que la dirección de destino de este paquete es la dirección de *broadcast*.

  Habiendo enviado este mensaje se entra en un bucle infinito que esperará hasta que hayan pasado "tasa_hello" segundos para enviar el siguiente mensaje HELLO.

  Para este nuevo mensaje HELLO se actualizan campos como el número de secuencia, el *timestamp* del paquete y las coordenadas del nodo. Teniendo actualizado esto pasamos a serializarlo y a enviarlo vía IP *broadcast* por el socket.

  Cuando se haya enviado actualizamos la variable *"helloInterval"* con el tiempo actual, a dejar constancia del envío en el registro y a incrementar el número de secuencia para obtener el del paquete siguiente. Hecho esto se pasa a la siguiente iteración.

- **void SendStatus()**
  Esta función también se lanza en otro hilo. Lo primero que se hace es llamar a la función ips_tabla() de nuestra *routing table* que como se ha comentado lo que consigue es un vector con las direcciones IP de aquellos nodos de la tabla que sean vecinos. El resultado de la función lo guardo en un vector llamado nodos.

Tenemos otro *chrono::timepoint* llamado *"t_aux"* que marcará la última vez que enviamos un mensaje STATUS.

Crearemos el mensaje STATUS teniendo en cuenta que el número de secuencia de este mensaje viene marcado por la variable *"status_sent"*.

Una vez lo tenemos creado y serializado correctamente procedemos a enviarlo, el envío esta vez se realiza a todas las IP del vector nodos. Para cada IP del vector se realiza el envío y se registra el mismo en el fichero de *"logging"*.

Una vez enviado este primer STATUS e incrementado la variable *"data_sent"* se entrará en un bucle infinito que realiza las siguientes acciones:

1. Esperar hasta que haya pasado un tiempo *"tasa_status"* desde que enviamos el último mensaje STATUS.
2. Cuando ha pasado el tiempo mencionado se obtienen los vecinos con una nueva llamada a "ips_tabla".
3. Se actualizan los datos del mensaje correspondientes a la posición del nodo, su velocidad y su número de secuencia.
4. Teniéndolo ya actualizado se procede a su envío a todos los vecinos del vector nodos, registrando cada envío.
5. Incrementaremos la variable *send_message*, que será un indicador sobre cuando debemos mandar mensajes DATA desde este hilo.
6. En el caso de que *send_message* haya alcanzado un valor de "*tasa_data*" procederemos al envío de mensajes DATA, a tantos nodos en la red como le hayamos indicado al programa a la hora de arrancarlo.
    6.1. El procedimiento que se sigue es la obtención de todas las direcciones IP de la tabla en la variable "nodosRT", una vez obtenidos, se recorrerá el vector tantas iteraciones como envíos queramos realizar siempre que el número de envíos no supere el tamaño de "nodosRT". Los envíos se realizan con la llamada a la función *SendData*, que como he comentado anteriormente envía y registra los envíos.
    6.2. Una vez hemos enviado a todos los nodos que queremos mensajes DATA ponemos la variable *"send_message"* a '0'

Al acabar la iteración se refresca la variable "t_aux" con el tiempo actual.

3.2.3 Funcionamiento de la aplicación

Una vez tenemos el código listo para funcionar, simplemente nos queda comprobar que la ejecución del programa es correcta, y que nos da el comportamiento esperado, necesario para permitirnos experimentar con el mismo.

El nombre que se le ha dado al ejecutable a la hora de compilarlo ha sido "protocolo", por ello desde el terminal de nuestra Raspberry podremos lanzarlo, provocando el siguiente comportamiento:

```
Introduce la tasa de envío de mensajes HELLO (en segundos) -> Valores probados: 0.1 , 1 , 2, 2.5
0.5
Introduce la tasa de envio de mensajes STATUS (en segundos) -> Valores probados 0.2, 2, 4, 5
1
Introduce la tasa de envío de mensajes de datos. Segun el entero que pongamos enviará datos cada tantos STATUS
2
Introduce a cuantos nodos enviar mensajes de dato cada vez (Maximo 4 en nuestro experimento)
2
Eth0 3232261123
Eth0 Broadcast 3232261375
Abierto socket en 1234 e ip 0
Esperando mensaje...
Enviando mensaje...
Recibidos 48 bytes de 3232261123 por el puerto 1234 de 3232261123
Descartado nuestro propio paquete broadcast
Esperando mensaje...
Mensaje enviado... Enviados 48 caracteres a 3232261375
Enviando mensaje...
Mensaje enviado... Enviados 48 caracteres a 3232261375
Recibidos 48 bytes de 3232261123 por el puerto 1234 de 3232261123
Descartado nuestro propio paquete broadcast
Esperando mensaje...
Enviando mensaje...
Mensaje enviado... Enviados 48 caracteres a 3232261375
Recibidos 48 bytes de 3232261123 por el puerto 1234 de 3232261123
Descartado nuestro propio paquete broadcast
Esperando mensaje...
```

Ilustración 14. Arranque del protocolo

Se puede observar como se nos piden los parámetros correspondientes a la tasa de envío de HELLO, STATUS y DATA. También se nos pide el número de nodos al que enviaremos DATA cada vez que toque realizar envíos.
Una vez arrancado, se nos muestra información referente a la dirección IP que estamos utilizando, la dirección *broadcast* a la que se envían los mensajes HELLO y el puerto que utiliza la aplicación.

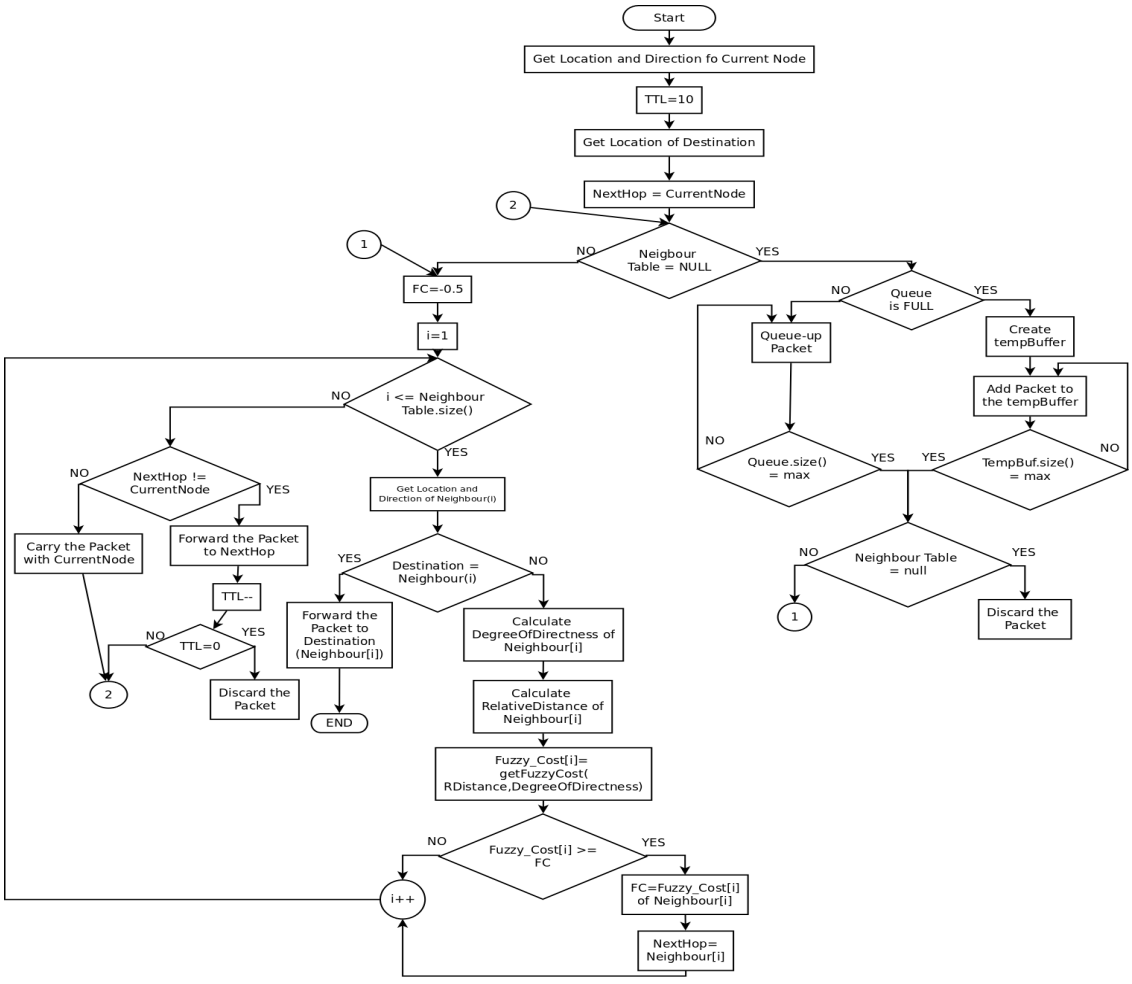


Ilustración 15. Diagrama de flujo para comportamiento análogo al sistema implementado

Habiéndose mostrado esos datos el protocolo empieza a funcionar enviando mensajes HELLO, y tratando de descubrir nodos vecinos en nuestra red, que a su vez puedan aportar información de otros nodos no vecinos.

Su hilo principal se encargará de visualizar el socket en busca de paquetes entrantes, cuando nos entre uno, se procederá a observar de quien proviene, y se actualizará o añadirá la información que tenemos en la *routing table* para dicho nodo dependiendo del tipo de paquete que nos haya llegado.

Este hilo principal también se encarga de realizar los envíos de mensajes ACK, ya que estos son generados en respuesta a los mensajes HELLO que nos llegan vía algún vecino que contribuye en el *flooding* de este tipo de mensajes. Para el envío de este tipo de mensajes se obtiene la dirección de origen del mensaje HELLO y se le pasa al sistema de decisiones, ya que será la dirección destino de nuestro ACK, si se encuentra un buen vecino, se le envía el ACK.

También se trata en este hilo principal el tema de los reenvíos de paquetes, cuando el nodo que ejecuta el programa actúa como nodo intermedio. Para reenviar los mensajes DATA y los ACK se hace uso del sistema de decisiones, para el reenvío de los HELLO se vuelve a hacer un broadcast en nuestro nodo. En todos los reenvíos se modifica y comprueba el valor del *hop_count*.

En el caso de los mensajes STATUS no tendremos reenvío alguno, estos mensajes se envían únicamente a nodos vecinos.

Los otros dos hilos lanzados a ejecución por el protocolo se encargarán de lo siguiente:

1. Un hilo se encargará de iniciar la creación (con los parámetros adecuados) y posterior comunicación de mensajes HELLO al resto de nodos, mediante el uso de la dirección de *broadcast* y el reenvío de estos mediante nodos intermedios. El parámetro “tasa_hello” le indicará al hilo el período que debe tener para enviar un nuevo mensaje.
2. El otro hilo se encargará de iniciar la creación y el envío a los vecinos de mensajes STATUS, ya se ha comentado previamente el proceso que se tiene para obtenerlos y como, después de enviar y registrar los envíos periódicos (cada “tasa_status” segundos se envían nuevos STATUS) el hilo comprobará si toca enviar mensajes de tipo DATA, y hará uso del sistema de lógica difusa para enviarlos a cada destino.

Todos estos mensajes (exceptuando los DATA) contienen información para que nuestro sistema de decisiones pueda nutrirse de ella y proporcionar mejores decisiones. De momento únicamente son usadas las coordenadas de cada nodo, pero pueden utilizarse, por ejemplo, la dirección y el sentido con el que se mueve cada nodo para ver si los nodos se van a acercar o a alejar, ya que los nodos que se alejan del destino podrían no ser buenos receptores.

Cuando detenemos el protocolo en un nodo, tendremos la información relativa a su ejecución en el fichero “logfile.txt”, del que se pueden extraer conclusiones sobre su funcionamiento.

4. Experimentos realizados

Una vez implementado y comprobado que el protocolo exhibe el comportamiento esperado, se puede proseguir con una campaña de experimentación con el objetivo de evaluar su comportamiento en distintos escenarios y condiciones. Estos experimentos nos darán orientaciones para futuros ajustes y mejoras del mismo.

Para la realización de nuestros experimentos podremos medir tres variables significativas a la hora de evaluar el rendimiento de los nodos de una red:

- Latencia de los paquetes, es decir, el tiempo que tardan en llegar desde el emisor hasta el receptor del mismo.
- Para los mensajes HELLO podremos medir su RTT, ya que el protocolo se sirve de mensajes ACK en respuesta.
- Veremos cual es la tasa de entrega correcta de paquetes y cuantos paquetes se pierden de media.

Todos estos experimentos están orientados a evaluar dichas variables, pero en la realidad los mismos dependen de distintos parámetros que facilitarán o dificultarán el buen comportamiento de los nodos en la red. Los parámetros a modificar en los experimentos corresponden a:

- Número de nodos fuente, es decir, aquellos nodos que envían mensajes de datos a otros en la red.
- Tamaño máximo del paquete de datos.
- Tasa de envío, es decir, tiempo entre envíos de mensajes HELLO. También hay tasa de envío para mensajes STATUS.
- Número de nodos destino.

Los primeros experimentos corresponderán a un escenario sin cortes entre los vecinos, donde no todos están al alcance del resto pero siempre habrá un camino para enviar un paquete de un nodo a otro. La responsabilidad de elegir el mejor camino para nuestro paquete recae sobre el protocolo.

Habiendo realizado todos los experimentos con los distintos parámetros de entrada en este escenario (al que llamaremos escenario estático, debido a la ausencia de cambios en la red) pasaremos a probar otro escenario.

El siguiente escenario corresponde a uno en el cual el protocolo está funcionando correctamente pero de repente un nodo pierde la conexión. Con esto queremos comprobar como reaccionaría el protocolo al perder un nodo importante en el enrutamiento y ver como encontraría la nueva ruta, además de analizar las distintas variables que medirán como ha afectado a la red el cambio en la topología.

4.1 Requisitos del experimento

Necesitamos varias cosas para realizar estos experimentos y observar los resultados correctamente:

- **Registro de mensajes enviados y recibidos, así como información sobre a quien enviaremos cada paquete y cual será el siguiente nodo en el camino hacia el destino.**
  La forma que se tiene para crear el registro es redirigir la salida de error de C++ a un fichero al que le puse de nombre logfile.txt.
  De esta forma imprimiendo los datos registrados mediante el "stream std::clog" orientado a cuestiones de logging podremos obtener en logfile.txt toda la información necesaria para ser posteriormente analizada.
  Dependiendo del evento, cada entrada tendrá un formato diferente. A parte de los eventos de envío y recepción de los distintos paquetes también añadí entradas para ver paquetes descartados por cierta causa, reenvíos o líneas que nos informan sobre la selección del mejor vecíno. Las que nos importan en nuestro experimento son las líneas que representan envíos y recepciones, el resto nos darán información para poder encontrar errores y agilizar su solución a la hora de desarrollar el protocolo.

- **Registro de tiempos, se usa el "time since epoch" de Linux (valor que medirá el tiempo pasado desde el 1 de Enero del 1970 con hora 00:00:00). Al inicio de cada entrada del fichero de registro tendremos el valor del time since epoch en ese instante**.
  Esto se consigue mediante el uso de la librería para C++ llamada chrono, al usar sus relojes de tipo "system_clock" se puede obtener un valor para el "time_since_epoch" y usar sus métodos para obtener una representación tan precisa como sea posible del tiempo pasado desde aquella fecha (en este caso se ha usado una representación del tiempo en microsegundos).

- **Sincronización de los relojes de todos los nodos, sin tener los relojes sincronizados no habría forma de conocer los retardos reales al transferir los paquetes de un nodo a otro.**
  Al no contar con conexión a internet en todas las Raspberry a la hora de realizar el experimento, y al no guardarse en estas la hora una vez se quedan sin alimentación eléctrica, se necesita usar el protocolo NTP ("Network Time Protocol") para usar un nodo que difunda su reloj al resto.
  El nodo que quiera enviar su reloj deberá tener instalado NTP y modificar el fichero de configuración de forma que permita el envío de paquetes NTP a los nodos que hay dentro de la red "ad-hoc".
  De la misma forma cualquier nodo que quiera recibir información NTP de un nodo emisor dentro de la red deberá notificarlo así en su fichero de configuración para NTP, poniendo como "fuente" al nodo emisor.
  Una vez teniendo esto listo, los nodos clientes deberán también tener instalado el programa "ntpdate" para forzar una actualización de hora.
  El programa ntpdate no funcionará en el caso en el que tenemos el servicio NTP en marcha, ya que usan el mismo puerto, por ello creé un script que pausaba el servicio, actualizaba la hora mediante el servidor de nuestra red "ad-hoc" y

volvía a activarlo posteriormente. Este script es necesario en un nodo intermedio al menos, ya que necesitaremos tener el servicio NTP activado, ya que a parte de ser cliente del nodo servidor, también hice que actuase como servidor del nodo que estaba fuera del rango wireless del servidor NTP.

- **Alguna forma de identificar inequívocamente cada paquete intercambiado.**
  Se usa también la idea de usar números de secuencia para identificar los paquetes, esta idea ha sido usada para implementar correctamente el "flooding" de mensajes HELLO en la red y ahora vemos que los números de secuencia, en combinación con la dirección IP del emisor pueden identificar los paquetes correctamente.

- **Es interesante la creación de un script que a partir de un "parsing" de los ficheros de registro "logfile.txt" de cada nodo saque todas las estadísticas asociadas al experimento a analizar.**

Una vez miradas esas cosas se puede pasar a definir lo que será el formato de cada entrada en el fichero de registro. Conocerlo facilita su análisis posterior.
Defino así las entradas más importantes, que son las correspondientes al envío y recepción de paquetes:

[Tiempo desde epoch] SENT HELLO IP_BROADCAST seqnumber X
[Tiempo desde epoch] RECEIVED HELLO IP_IP_EMISOR seqnumber X
[Tiempo desde epoch] SENT ACK IP_DESTINO via IP_NEXTHOP seqnumber X
[Tiempo desde epoch] RECEIVED ACK IP_EMISOR via IP_LASTHOP seqnumber X
[Tiempo desde epoch] SENT STATUS IP_VECINO seqnumber X
[Tiempo desde epoch] RECEIVED STATUS IP_ORIGEN seqnumber X
[Tiempo desde epoch] SENT HELLO IP_BROADCAST seqnumber X
[Tiempo desde epoch] SENT DATA IP_DESTINO via IP_NEXTHOP seqnumber X

Tabla 3. Formato para tipos de mensajes de registro de envíos y recepciones

```
[1472322795652247] RECEIVED STATUS 3232261121 seqnumber 4
[1472322795653643] RECEIVED ACK 3232261125 via 3232261122 seqnumber 27
[1472322795685869] SENT HELLO 3232261375 seqnumber 30
Se descarta el RECEIVED HELLO 3232261123 30 seqnumTabla 0
[1472322795737621] RECEIVED ACK 3232261121 via 3232261121 seqnumber 30
[1472322795755135] RECEIVED DATA 3232261125 via 3232261122 seqnumber 6 Message: Hay mas destinos
Se descarta el RECEIVED HELLO 3232261123 30 seqnumTabla 0
Se descarta el RECEIVED HELLO 3232261123 30 seqnumTabla 0
[1472322795864742] RECEIVED ACK 3232261124 via 3232261124 seqnumber 30
[1472322795909908] RECEIVED ACK 3232261122 via 3232261122 seqnumber 30
[1472322795927640] RECEIVED HELLO 3232261122 seqnumber 12
FORWARDED RECEIVED HELLO  3232261122 TO 3232261375seqnumber12
[FUZZY] El destino es vecino, enviamos a su ip 3232261122
[1472322795965659] SENT ACK 3232261122 via 3232261122 seqnumber 12
Se descarta el RECEIVED HELLO 3232261122 12 seqnumTabla 12
Se descarta el RECEIVED HELLO 3232261122 12 seqnumTabla 12
[1472322796053005] RECEIVED HELLO 3232261124 seqnumber 13
FORWARDED RECEIVED HELLO  3232261124 TO 3232261375seqnumber13
```

Ilustración 16. Extracto de *logfile* del nodo 3 con las entradas mencionadas.

4.2 Experimento estático

Para la realización de este experimento únicamente se usan tres nodos. La finalidad del mismo es ver como se comportan los enlaces a la hora de enviar datos y como responden a distintos parámetros de entrada. La presencia de paredes dificulta que la señal abarque grandes distancias y añade mucho ruido a la misma.

```
pi@raspberrypi:~ $ sudo iw dev wlan0 station dump | grep -e "bitrate" -e "Statio
n" -e signal
Station 00:80:5a:51:0c:9a (on wlan0)
        signal:         -56 dBm
        signal avg:     -57 dBm
        tx bitrate:     54.0 MBit/s
        rx bitrate:     48.0 MBit/s
Station 00:80:5a:51:0c:94 (on wlan0)
        signal:         -82 dBm
        signal avg:     -80 dBm
        tx bitrate:     1.0 MBit/s
        rx bitrate:     1.0 MBit/s
Station 00:80:5a:51:0c:9c (on wlan0)
        signal:         -66 dBm
        signal avg:     -66 dBm
        tx bitrate:     1.0 MBit/s
        rx bitrate:     1.0 MBit/s
Station 00:80:5a:51:0c:90 (on wlan0)
        signal:         -58 dBm
        signal avg:     -57 dBm
        tx bitrate:     54.0 MBit/s
        rx bitrate:     54.0 MBit/s
```

Ilustración 17. Velocidad de transmisión/recepción de los módulos wireless

Aquí comprobamos que el valor máximo de transmisión y recepción alcanzado de nuestros módulos (en enlaces con tráfico, los de 1 Mbit/s no están usando toda su capacidad) es 54MBit/s.

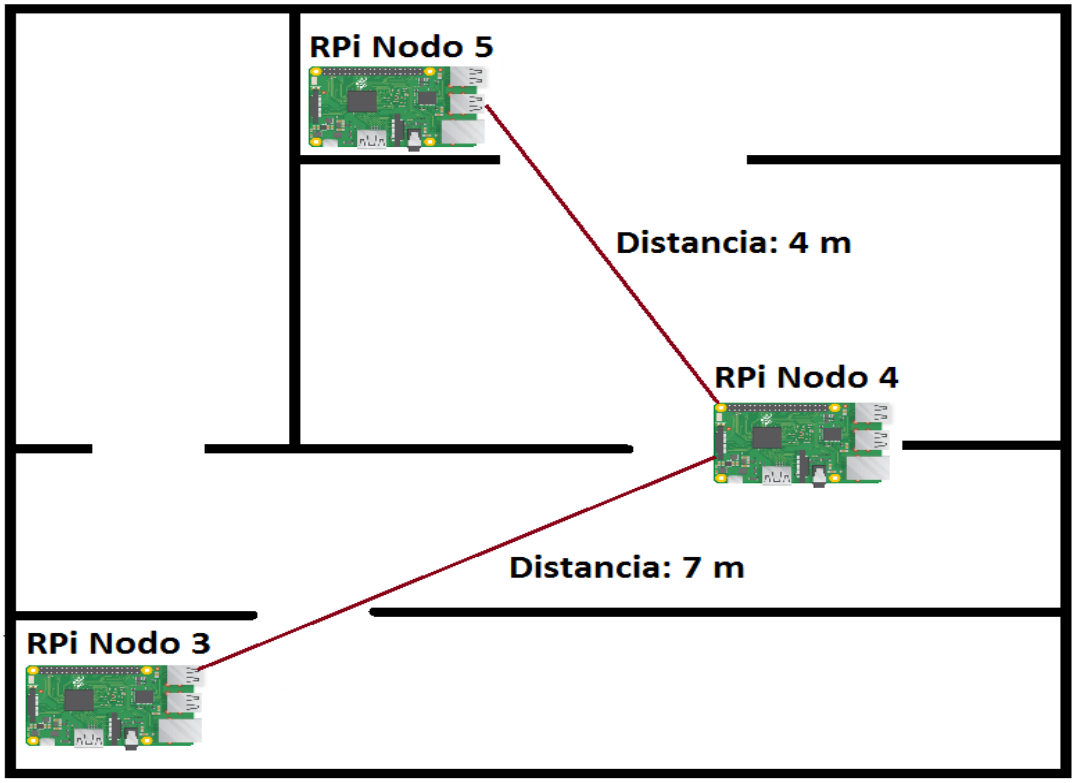


Ilustración 18. Distribución nodos de la red para este experimento. Las líneas rojas representan los enlaces entre vecinos.

Se comprueba como se realiza el intercambio de datos tanto entre nodos vecinos (mediante transmisión directa del paquete al vecino) como entre nodos que no se ven entre ellos (no están dentro del rango de alcance inalámbrico del otro, por tanto hará uso del protocolo). Así podemos ver que parámetros nos proporcionarán una mayor fiabilidad de los enlaces a la hora de probar el otro tipo de experimentos que contarán con más nodos y por tanto más enlaces, además de tener más de una ruta posible para cada paquete enviado.

Como se observa, el experimento ha sido realizado en interior, el nodo emisor se sitúa en un punto desde donde le llega buena señal del nodo vecino (a pesar de los obstáculos que se tienen para obtenerla) pero sin embargo no conoce a priori nada sobre el nodo receptor.

El nodo intermedio se sitúa estratégicamente en un punto desde el cual se obtiene buena señal de los nodos emisor y receptor, para que pueda realizar correctamente la tarea de realizar el enrutamiento de paquetes entre uno y otro.

El nodo destino está situado, por tanto, de forma en la que solo ve al nodo intermedio y no conoce nada del nodo destino.

Habrán 9 experimentos para cada tamaño de paquete, los tamaños escogidos han sido 128 Bytes y 1024 Bytes en los paquetes de datos.
Realizados quedan los siguientes experimentos para cada tamaño de paquete, añadidos en el anexo:

1. El nodo emisor con IP acabada en 3 envía a un solo nodo.
2. El nodo emisor con IP acabada en 3 envía a los otros dos nodos.
3. Los 3 nodos envían a todos.

Los tres experimentos se realizarán cada uno con tres variantes:

1. Tasa HELLO = 0,5 segundos; Tasa STATUS = 1 segundo; Tasa DATA = 3 segundos.
2. Tasa HELLO = 1 segundo; Tasa STATUS = 2 segundos; Tasa DATA = 4 segundos.
3. Tasa HELLO = 2 segundos; Tasa STATUS = 2 segundo; Tasa DATA = 4 segundos.

Quedando así una cantidad de 9 experimentos para cada tamaño de paquete.

Los resultados de los experimentos son mostrados en tablas, donde los nodos de las filas envían paquetes a los nodos de las columnas. La nomenclatura utilizada es la siguiente:

- %H, %D, %A, %S: Valores correspondientes al porcentaje de entrega correcta de paquetes HELLO, DATA, ACK y STATUS, respectivamente.
- TH,TS,TD,TA: Retardo en la entrega de paquetes de tipo HELLO, STATUS, DATA y ACK, respectivamente.
- “**********” Implica que no haya enlace.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=81,2 %A=98,76 %S=100 %D=100 TH=27ms TS=28ms TD=19ms TA=38ms	%H=71 %A=95,23 %D=95 TH=135ms TD=211ms TA=257ms
Nodo 4	**********	**********	%H=60,9 %A=94,39 %S=95,3 %D=90,47 TH=80,3ms TS=57ms TD=71,5ms TA=88,2ms	**********	%H=81,2 %A=95,87 %S=95,38 %D=95,23 TH=123ms TS=223ms TD=183ms TA=223ms
Nodo 5	**********	**********	%H=49,6 %A=93,181 %D=82,35 TH=49,1ms TD=149ms TA=140ms	%H=75,19 %A=100 %S=100 %D=100 TH=16,3ms TS=24,5ms TD=26,6ms TA=16,5ms	**********

Tabla 4. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 0.5 sec, Status_jitter = 1 sec, Período envío DATA = 3 sec. Todos los nodos enlazados envían al resto.

En estos experimentos lo que llama la atención es que los paquetes HELLO, enviados utilizando BROADCAST, tienen una tasa de entrega bastante mala incluso entre nodos con buena calidad de señal. Llama la atención porque el resto de paquetes tienen tasas de entrega más elevadas y son entregados de manera “unicast”.

```
[1472326742659154] SENT DATA 3232261124 via 3232261124 seqnumber 0 Message Hay mas destinos
[1472326742743598] SENT DATA 3232261125 via 3232261124 seqnumber 1 Message Hay mas destinos
[1472326745840327] SENT DATA 3232261124 via 3232261124 seqnumber 2 Message Hay mas destinos
[1472326745922805] SENT DATA 3232261125 via 3232261124 seqnumber 3 Message Hay mas destinos
[1472326749048383] SENT DATA 3232261124 via 3232261124 seqnumber 4 Message Hay mas destinos
[1472326749116891] SENT DATA 3232261125 via 3232261124 seqnumber 5 Message Hay mas destinos
[1472326752198761] SENT DATA 3232261124 via 3232261124 seqnumber 6 Message Hay mas destinos
[1472326752322042] SENT DATA 3232261125 via 3232261124 seqnumber 7 Message Hay mas destinos
[1472326755400660] SENT DATA 3232261124 via 3232261124 seqnumber 8 Message Hay mas destinos
```

Ilustración 19. Extracto del “log” del nodo 3

Con la imagen anterior podemos observar como efectivamente los paquetes que el nodo 3 envía al nodo 4 los realiza de manera directa, mientras que los dirigidos al nodo 5 pasan vía nodo 4.

```
[1472326740546190] SENT STATUS 3232261124 seqnumber 0
[1472326741591742] SENT STATUS 3232261124 seqnumber 1
[1472326742617676] SENT STATUS 3232261124 seqnumber 2
[1472326743770494] SENT STATUS 3232261124 seqnumber 3
[1472326744793522] SENT STATUS 3232261124 seqnumber 4
[1472326745810369] SENT STATUS 3232261124 seqnumber 5
[1472326746942544] SENT STATUS 3232261124 seqnumber 6
[1472326747988594] SENT STATUS 3232261124 seqnumber 7
[1472326749013612] SENT STATUS 3232261124 seqnumber 8
```

Ilustración 20. Extracto del "log" del nodo 3

Con este extracto observamos que el nodo 3 sólo envía mensajes STATUS a su vecino de IP acabada en 4, ni rastro del quinto nodo que no está dentro de su alcance *wireless*.

```
[1472326742918754] RECEIVED DATA 3232261123 via 3232261124 seqnumber 1 Message: Hay mas destinos
[1472326745991622] RECEIVED DATA 3232261123 via 3232261124 seqnumber 3 Message: Hay mas destinos
[1472326749323718] RECEIVED DATA 3232261123 via 3232261124 seqnumber 5 Message: Hay mas destinos
[1472326752517827] RECEIVED DATA 3232261123 via 3232261124 seqnumber 7 Message: Hay mas destinos
[1472326758871176] RECEIVED DATA 3232261123 via 3232261124 seqnumber 10 Message: Hay mas destinos
[1472326761927474] RECEIVED DATA 3232261123 via 3232261124 seqnumber 12 Message: Hay mas destinos
```

Ilustración 21. Extracto del "log" del nodo 5 con las recepciones referentes al nodo 3

Mediante el uso del comando “grep” se filtran las recepciones de paquetes de datos provenientes del nodo 3, usando la cadena “RECEIVED DATA 3232261123” como patrón a buscar. Podemos observar comparando a partir de la ilustración de los envíos del nodo 3 que nos han ido llegando todos los mensajes de datos vía nodo 4.

4.3 Experimento dinámico

Con estos experimentos se mostrará el comportamiento de lo implementado en un escenario más realista, donde un nodo puede desconectarse de la red y el protocolo debe hacer frente a este hecho generando un cambio de ruta para los paquetes que deban realizar más de un salto en nuestra red.

Añadiremos 2 nodos mas en este experimento, con las siguientes distancias entre ellos:

- El nodo 2, situado a 2 metros del nodo 1, A 3 metros del nodo 5, a 2 metros del 4 y a 6 metros del 3.
- El nodo 1, situado a 7 metros del nodo 5 (habiendo paredes de por medio), a 3 metros del nodo 3, a 4 metros del nodo 4 y a 2 metros del nodo 2.

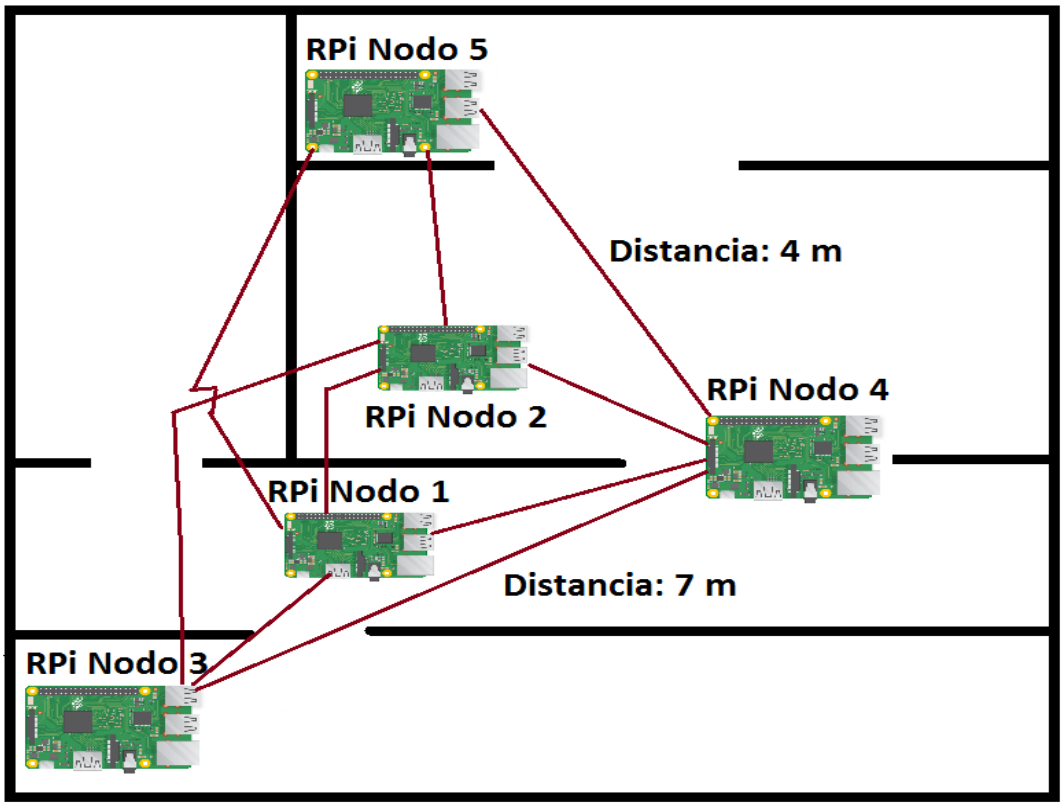


Ilustración 22. Distribución de nodos en la red para este experimento. Líneas rojas representan enlaces entre vecinos

Para ello he realizado 11 experimentos diferentes, todos ellos de la siguiente forma:

1. Los nodos 3 y 5 lanzarán el programa los primeros, no tendrán ruta posible entre ellos, por tanto no habrá intercambio de información.
2. Pasados 10 segundos, lanzaremos en el resto de nodos el programa, de esta forma los paquetes empezarán a fluir entre los nodos 3 y 5, así como entre el resto de nodos conectados.
3. Pasados 25 segundos desconectaremos el nodo 4, que es el elegido por los nodos 3 y 5 para enviarse paquetes entre ellos. El protocolo deberá hacer frente a este

hecho mediante la elección de un nuevo vecino con el mejor coste “fuzzy” de la “routing table”.

4. Pasados 20 segundos desconectaremos el nodo 2, que es el que tenia mejor coste, quedándonos únicamente el nodo 1 como interfaz entre los nodos 3 y 5. Se puede apreciar una pérdida de paquetes mayor cuando utilizan este nodo para comunicarse, ya que no es un buen vecino del nodo 5, que sólo es capaz de verlo e intercambiar información con el en ocasiones.
5. Pasados 15 segundos, pararemos los nodos restantes y podremos obtener los resultados del fichero “logfile.txt” de cada nodo.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=97,72 %A=100 %S=100 %D=100 TH=103ms TS=226ms TD=206ms TA=148ms	%H=93,65 %A=100 %S=100 %D=100 TH=29,9ms TS=65,5ms TD=69,6ms TA=38,4ms	%H=100 %A=100 %S=100 %D=100 TH=24ms TS=40,7ms TD=64,8ms TA=44,1ms	%H=85,71 %A=73,68 %S=67,85 %D=73,33 TH=1200ms TS=953ms TD=1087ms TA=1353ms
Nodo 2	%H=100 %A=100 %S=100 %D=100 TH=27,5ms TS=16,6ms TD=8,3ms TA=5ms	**********	%H=100 %A=100 %S=100 %D=100 TH=34,2ms TS=132ms TD=19ms TA=22ms	%H=95,65 %A=100 %S=100 %D=100 TH=17,5ms TS=25,2ms TD=7,3ms TA=5,9ms	%H=90,9 %A=92,85 %S=100 %D=100 TH=1670ms TS=1750ms TD=1764ms TA=1520ms
Nodo 3	%H=95,16 %A=100 %S=100 %D=100 TH=8ms TS=4,4ms TD=3,3ms TA=12,7ms	%H=93,18 %A=100 %S=100 %D=100 TH=70,8ms TS=110ms TD=121ms TA=125ms	**********	%H=87,5 %A=100 %S=100 %D=100 TH=22,2ms TS=12,2ms TD=14,5ms TA=9,8ms	%H=73,17 %A=77,19 %D=80 TH=1024ms TD=1349ms TA=1373ms
Nodo 4	%H=100 %A=100 %S=90,9 %D=80 TH=85,3ms TS=79,6ms TD=149ms TA=120ms	%H=100 %A=100 %S=90,9 %D=80 TH=102ms TS=151ms TD=303ms TA=175ms	%H=96 %A=95,23 %S=81,81 %D=80 TH=85,3ms TS=124ms TD=251ms TA=121ms	**********	%H=92 %A=69,56 %S=72,72 %D=80 TH=1976ms TS=2038ms TD=2300ms TA=1873ms
Nodo 5	%H=91,8 %A=93,181 %S=100 %D=100 TH=46,9ms TS=31,8ms TD=109ms TA=123ms	%H=97,67 %A=100 %S=100 %D=100 TH=148ms TS=186ms TD=196ms TA=156ms	%H=70,37 %A=90,47 %D=87,5 TH=75,8ms TD=294ms TA=250ms	%H=95,65 %A=100 %S=100 %D=100 TH=35,1ms TS=31,2ms TD=105ms TA=51ms	**********

Tabla 5. Experimento dinámico 3 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Todos los nodos envían al resto

Podemos observar que pese al aumento de nodos en la red respecto al experimento estático la tasa de mensajes HELLO recibido ha aumentado, esto se debe a que ahora un mensaje de este tipo tiene más rutas posibles a parte de la de sus vecinos directos, paliando así en parte el problema que generaba el *broadcasting* en un medio inalámbrico.

Observamos aquí también que los únicos nodos no vecinos en los experimentos son los nodos 3 y 5, ya que son los únicos que no se envían mensajes de tipo STATUS entre ellos.

La tasa de recepción de mensajes STATUS nos puede indicar la calidad del enlace formado entre los nodos. Viendo este parámetro podemos observar que los envíos del nodo 1 a 5 se pueden ver afectados por la baja calidad del enlace (menor del 80%), un hecho que tiene sentido, ya que la distancia entre ambos es la mayor de todos los vecinos del nodo 5. Además la señal debe atravesar dos paredes como se puede ver en la ilustración asociada al experimento.

Podemos observar como el protocolo ha funcionado con bastante solvencia en cuanto al hecho de enviar los paquetes de datos al destino que toca. Si observamos los envíos lanzados desde el nodo 3 hacia el nodo 5 vemos que el 80% de los paquetes han sido enviados sin problemas. Probablemente la pérdida de paquetes que se aprecia es provocada al desactivar los dos vecinos que hasta el momento de su desconexión estaban actuando como nodos intermedios en la ruta del paquete. Finalmente, el hecho de dejar únicamente al nodo 1 como posible nodo intermedio provoca que se pierdan más paquetes de datos cuando lo usemos (recuerdo que su enlace con el nodo 5 era bastante deficiente).

En el caso contrario, para los envíos del nodo 5 dirigidos al nodo 3 podemos observar un mayor porcentaje de paquetes entregados, puede ser debido a que en este sentido del envío el nodo 5 si que puede ver al nodo 1 como un buen vecino (100% de paquetes STATUS le llegan) y por ello no se producen las pérdidas de datos que veíamos en el otro escenario al usar al primer nodo como intermedio en la ruta del paquete.

Pasemos a comprobar que el sistema de decisiones funciona correctamente, según he comprobado, los cálculos del mismo según las coordenadas asignadas a los nodos dan un orden de *fuzzy cost* de esta mañera cuando enviamos paquetes del nodo 3 al nodo 5:

FuzzyCostNodo4 > FuzzyCostNodo2 > FuzzyCostNodo1

Por ello procedo a comprobar en los registros “logfile.txt” correspondientes a los nodos 3 y 5 cuales son los envíos y recepciones de paquetes DATA que se tienen registrados.

```
                    EXPERIMENTOSDINAMICOSFINALES128B javiermoraga$ cat captura3nodo3.txt |
 grep "SENT DATA 3232261125"
[1472322787079019] SENT DATA 3232261125 via 3232261124 seqnumber 3 Message Hay mas destinos
[1472322791339341] SENT DATA 3232261125 via 3232261124 seqnumber 7 Message Hay mas destinos
[1472322795627951] SENT DATA 3232261125 via 3232261124 seqnumber 11 Message Hay mas destinos
[1472322799915782] SENT DATA 3232261125 via 3232261124 seqnumber 15 Message Hay mas destinos
[1472322804248472] SENT DATA 3232261125 via 3232261124 seqnumber 19 Message Hay mas destinos
[1472322808570235] SENT DATA 3232261125 via 3232261124 seqnumber 23 Message Hay mas destinos
[1472322812798304] SENT DATA 3232261125 via 3232261122 seqnumber 26 Message Hay mas destinos
[1472322817060119] SENT DATA 3232261125 via 3232261122 seqnumber 29 Message Hay mas destinos
[1472322821279805] SENT DATA 3232261125 via 3232261122 seqnumber 32 Message Hay mas destinos
[1472322825530646] SENT DATA 3232261125 via 3232261122 seqnumber 35 Message Hay mas destinos
[1472322829683759] SENT DATA 3232261125 via 3232261122 seqnumber 38 Message Hay mas destinos
[1472322833845855] SENT DATA 3232261125 via 3232261121 seqnumber 40 Message Hay mas destinos
[1472322837988267] SENT DATA 3232261125 via 3232261121 seqnumber 42 Message Hay mas destinos
[1472322842099321] SENT DATA 3232261125 via 3232261121 seqnumber 44 Message Hay mas destinos
[1472322846234747] SENT DATA 3232261125 via 3232261121 seqnumber 46 Message Hay mas destinos
```

Ilustración 23. Envíos DATA del nodo 3 al nodo 5

Como se puede observar, el nodo 3 realiza los envíos siguiendo el esquema previsto, aproximadamente los primeros 25 segundos realiza los envíos sirviéndose del nodo 4. Una vez este se desconecta, pasará a utilizar el nodo 2 durante los siguientes 20 segundos ya que el nodo 4 ha sufrido la desconexión programada. Cuando este ha sido desconectado pasará a usar el nodo 1 como nodo intermedio en la ruta hacia el destino. No se ha podido probar un escenario con más saltos debido al reducido espacio del que se disponía para colocar los nodos.

```
                    EXPERIMENTOSDINAMICOSFINALES128B javiermoraga$ cat captura3nodo5.txt | grep
"RECEIVED DATA 3232261123"
[1472322787906396] RECEIVED DATA 3232261123 via 3232261124 seqnumber 3 Message: Hay mas destinos
[1472322793301453] RECEIVED DATA 3232261123 via 3232261124 seqnumber 7 Message: Hay mas destinos
[1472322799003181] RECEIVED DATA 3232261123 via 3232261124 seqnumber 11 Message: Hay mas destinos
[1472322802505458] RECEIVED DATA 3232261123 via 3232261124 seqnumber 15 Message: Hay mas destinos
[1472322806349162] RECEIVED DATA 3232261123 via 3232261124 seqnumber 19 Message: Hay mas destinos
[1472322815245518] RECEIVED DATA 3232261123 via 3232261122 seqnumber 26 Message: Hay mas destinos
[1472322818432548] RECEIVED DATA 3232261123 via 3232261122 seqnumber 29 Message: Hay mas destinos
[1472322822320559] RECEIVED DATA 3232261123 via 3232261122 seqnumber 32 Message: Hay mas destinos
[1472322825631506] RECEIVED DATA 3232261123 via 3232261122 seqnumber 35 Message: Hay mas destinos
[1472322834073058] RECEIVED DATA 3232261123 via 3232261121 seqnumber 40 Message: Hay mas destinos
[1472322838107106] RECEIVED DATA 3232261123 via 3232261121 seqnumber 42 Message: Hay mas destinos
[1472322846267969] RECEIVED DATA 3232261123 via 3232261121 seqnumber 46 Message: Hay mas destinos
```

Ilustración 24. Recepciones en el nodo 5 de DATA provenientes del nodo 3

Aquí vemos que se han perdido los siguientes mensajes:

- El mensaje 23: Este mensaje ha sido perdido al no haber hecho frente al momento del envío a la desconexión del nodo 4. Los siguientes paquetes utilizan una ruta diferente.
- El mensaje 38: Del mismo modo que en el caso anterior, el protocolo no ha tenido tiempo a hacer frente a la desconexión del nodo 2. Los siguientes paquetes utilizarán la última ruta que les queda.
- El mensaje 44: Perdido debido al mal enlace establecido, como se ha comentado antes, entre el nodo 1 y el nodo 5 que provocan que algunos envíos no lleguen finalmente al destino.

5. Conclusiones

Acabada ya la experimentación podemos enumerar el trabajo realizado:

- Después del análisis de los diferentes protocolos que se pueden usar en redes vehiculares y las aplicaciones que se le pueden dar a estas, hemos entrado en contexto con nuestro protocolo diseñado, al ver que las decisiones tomadas se guiaban justamente por parámetros relativos al dinamismo (como pueden ser la posición, velocidad y sentido del movimiento), una característica muy propia de las redes VANET.
- Se ha relatado de manera breve cual es la idea general de funcionamiento para nuestro protocolo, que ha servido como referencia durante la implementación, y se han expuesto las ideas necesarias para facilitar la comprensión del mismo al lector.
- Se ha implementado un protocolo funcional, al que se le podrán aplicar posteriores mejoras. Durante el proceso de desarrollo hemos visto la importancia de la eficiencia a la hora de desarrollar un proyecto así y he conseguido mejorar un poco mis habilidades como programador.
- Se han realizado experimentos en dos escenarios, dentro de las posibilidades que se tenía. En base a esos experimentos podemos ver donde se ha flaqueado y donde se deben aplicar mejoras.
- Se intentó filtrar las conexiones según calidad de señal utilizando comandos asociados al módulo wireless, al no ser posible se trató de crear un tipo de red diferente (“red mesh”) con parámetros configurables mediante un script que añadiré al anexo. Este tipo de red arrojaba malos resultados así que se volvió a la idea inicial.
- La extracción de resultados de los registros se ha realizado mediante técnicas de *scripting* y *parsing* de la información.
- Es aquí en este apartado donde veremos que mejoras podemos aplicar a nuestro proyecto.

Se ha visto, comparando los experimentos estáticos frente a los dinámicos, que la tasa de recepción de mensajes HELLO es más elevada en el caso dinámico, cuando hay mas nodos. Usar *broadcasting* en un entorno inalámbrico da problemas, que se han visto reflejados en los resultados. En el caso dinámico, como ya se ha comentado, los paquetes *broadcast* tienen más rutas para llegar a su destino, es por ello que la tasa de recepción aumenta contra todo pronóstico.

Es muy importante la eficiencia del protocolo, el uso de sockets de una capa de nivel inferior (RAW sockets [20], a nivel de enlace) hubiese ahorrado al programa mucho procesamiento en los nodos, al no ser necesario ejecutar el comando destinado a ver los vecinos y su posterior análisis. En una implementación final, la creación de los paquetes podría realizarse de esta manera, posiblemente en Internet habrán APIs que permitirán crear tramas a nivel de enlace relacionadas con el estándar 802.11.

Los nodos, a menor potencia (nodos de IP 2 y 5 son los menos potentes) generan mayor retardo a la hora de añadir los datos de los mensajes al protocolo. Por ello prima la buena eficiencia del programa, así como contar con nodos que tengan la potencia suficiente para correrlo.

A medida que llenamos la red de mensajes mayor es el retardo generado en algunos nodos a la hora de recibir paquetes, probablemente este hecho se debe a que los mensajes se encolan en el socket antes de ser recibidos hasta ser atendidos.

Hay que vigilar correctamente que la calidad de señal establecida para los nodos sea lo suficientemente buena como para que los cambios bruscos en la misma no afecte al rendimiento del protocolo.

Cuando hay más de un salto para enviar un mensaje, el retardo del mismo para llegar al destino se ve aumentado, esto puede ser un problema si nuestro protocolo no tiene la capacidad de operar con la suficiente eficiencia y rapidez, ya que todos los retardos generados en los nodos intermedios afectarán al retardo de la entrega.

En cuanto a la entrega de paquetes obtenida en los experimentos, se han obtenido resultados muy satisfactorios, por lo que con todo lo anterior podemos afirmar que el trabajo realizado constituye una válida primera aproximación a la implementación sobre nodos reales del protocolo en cuestión, y que permitirá en un futuro próximo realizar las primeras pruebas sobre vehículos así como validar los modelos de simulación desarrollados.

5.1 Mejoras propuestas

Como se ha comentado, este proyecto puede ser mejorado mediante las siguientes acciones:

- Adquisición de datos del GPS, que mediante el uso de las *Raspberry* con baterías externas nos den la movilidad necesaria para comprobar el protocolo en un escenario más realista con cambios bruscos en las señales que recibe el nodo.
- Adquisición de datos mediante el uso de una IMU conectada a los buses SPI o I2C, de la cual se podrán extraer parámetros interesantes que podrán ser utilizados dentro de nuestro sistema de decisión.
- Reducir la carga en cuanto a procesamiento del protocolo mediante el uso de RAW sockets[20] que nos permitan obtener la dirección MAC de los nodos que envían los paquetes de manera directa, y así poder conocer los vecinos con mayor eficiencia.
- Mejora de la interfaz del sistema.
- Tratar de estudiar los problemas asociados al *broadcast* en un medio *wireless* e intentar reducir sus efectos.
- Implementar la cola de mensajes que no han podido ser enviados debido a la desconexión temporal de los nodos.

6. Bibliografía

[1] Bijan Paul,Md. Ibrahim and Md. Abu Naser Bikas, "*VANET Routing Protocols: Pros and cons"* ,International journal of Computer Applications (0975-8887) Volume 20-No.3,April 2011

[2] Mir, Z. and Filali, F. (2014) "*LTE and IEEE 802.11p for Vehicular Networking: A Performance Evaluation*" EURASIP Journal on Wireless Communications and Networking, 2014, 89. http://dx.doi.org/10.1186/1687-1499-2014-89

[3] Sandhaya Kohli, Bandanjot Kaur, Sabina Bindra. "*A comparative study of Routing Protocols in VANET"*

[4] Fan Li, Yu Wang. "*Routing in Vehicular Ad Hoc networks: A survey"*, IEEE Vehicular Technology Magazine, June 2007

[5]. G. Liu, B.-S. Lee, B.-C. Seet, C.H. Foh, K.J. Wong, and K.-K. Lee, *"A routing strategy for metropolis vehicular communications,"* in International Conference on Information Networking (ICOIN), pp. 134–143, 2004.

[5.1]. H. Füßler, M. Mauve, H. Hartenstein, M. Kasemann, and D. Vollmer, *"Locationbased routing for vehicular ad-hoc networks,"* ACM SIGMOBILE Mobile Computing and Communications Review (MC2R), vol. 7, no. 1, pp. 47–49, Jan. 2003.

[6] M. Durresi, A. Durresi, and L. Barolli, *"Emergency broadcast protocol for intervehicle communications,"* in ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems—Workshops (ICPADS'05), 2005.

[7] M. Sun, W. Feng, T.-H. Lai, K. Yamada, H. Okada, and K. Fujimura, *"GPS-based message broadcasting for inter-vehicle communication,"* in ICPP '00: Proceedings of the 2000 International Conference on Parallel Processing, 2000

[8] L. Briesemeister, L. Schäfers, and G. Hommel, *"Disseminating messages among highly mobile hosts based on inter-vehicle communication,"* in Proceedings of the IEEE Intelligent Vehicles Symposium, pp. 522–527, 2000.

[9] Marie-Ange Lèbre, Frédéric Le Mouël, Eric Menard, Julien Dillschneider, Richard Denis. "*VANET Applications: Hot Use Cases"*, Center of Innovation in Telecommunications and Integration of Services, University of Lyon,1 Jul 2014.

[10] Rakesh Kumar, Mayank Dave, *"A Comparative Study of Various Routing Protocols in VANET"* IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1, July 2011

[11] T. Nadeem, S. Dashtinezhad, C. Liao, and L. Iftode, *"Trafficview: traffic data dissemination using car-to-car communication,"* Mobile Computing and Communications Review, vol. 8, no. 3, pp. 6–19, 2004

[12] Androidauthority.com. (2016). *WiFi standards explained: what you should know about the new 802.11 ad, ah & af standards*. [Activa] Disponible en: http://www.androidauthority.com/wifi-standards-explained-802-11b-g-n-ac-ad-ah-af-666245/ [Accedida 3 Sep. 2016].

[13] Starlino.com. (2016). *A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications*. *« Starlino Electronics*. [Activa] Disponible en: http://www.starlino.com/imu_guide.html [Accedida 3 Sep. 2016].

[14] [RFC5905] D. Mills, U.Delaware, J.Martin, *"Network Time Protocol Version 4: Protocol and Algorithms Specification"*, ST, RFC 5905, June 2010.

[15] Wiki.archlinux.org. (2016). "*Network Time Protocol daemon (Español) - ArchWiki"* [Activa] Disponible en: https://wiki.archlinux.org/index.php/Network_Time_Protocol_daemon_(Espa%C3%B1ol) [Accedida 3 Sep. 2016].

[16] Kayhan Zrar Ghafoor, Kamalrulnizam Abu Bakar, Shaharuddin Salleh, Kevin C. Lee, Mohd Murtadha Mohamad, Maznah Kamat, and Marina Md Arshad. "Fuzzy logic-assisted geographical routing over vehicular ad hoc networks". International Journal of Innovative Computing, Information and Control, 8(7(B)):5095–5120, July 2012. ISSN 1349-4198.

[17] Tesina de Master UPV de José Navarro Alabarta, dirigida por Juan Vicente Capella Hernández y Ángel Valera, *"Evaluación mediante simulación de redes de sensores aplicatas a robots móviles y vehículos ligeros"*, 14 Diciembre de 2015

[18] Genbetadev.com. (2011). *¿Cómo calcular la distancia entre dos puntos geográficos en C#? (Fórmula de Haversine)*. [online] Disponible en: http://www.genbetadev.com/cnet/como-calcular-la-distancia-entre-dos-puntos-geograficos-en-c-formula-de-haversine [Accedida 3 Sep. 2016].

[19] Seattlerobotics.org. (2016). *Fuzzy Logic Tutorial - An Introduction*. [activa] Disponible en: http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html [Accedida 3 Sep. 2016].

[20] Gsyc.urjc.es. (2016). *Socket Raw*.. [activa] Disponible en: https://gsyc.urjc.es/~juaner/investigacion/pfc2/node22.html [Accedida 3 Sep. 2016].

[21] Bo Li, Mahdieh Sadat Mirhashemi, Xavier Laurent, Jinzi Gao, *"Wireless Access for Vehicular Environments"*

7. “Manual de usuario”

7.1 Puesta a punto de los nodos

Para poder tener los nodos operativos y listos para funcionar con el protocolo es necesario conocer las utilidades que nos sirven para hacer el protocolo funcionar y que configuraciones hay que realizar en algunas de ellas.

Los dispositivos que se me han prestado para realizar este trabajo son los siguientes, todos forman parte de la red 192.168.100.0/24:

- ***Raspberry Pi 2 Model B V1.1:*** A la que asigné la dirección IP 192.168.100.3. Este era el dispositivo desde el que controlaba el resto mediante conexiones SSH.
- ***Raspberry Pi 2 Model B V1.1:*** A la que asigné la dirección IP 192.168.100.1.
- ***Raspberry Pi 2 Model B V1.1:*** A la que asigné la dirección IP 192.168.100.4. Este era el mejor nodo para enrutar paquetes entre el nodo de IP acabada en 3 y el nodo de IP acabada 5 que no se veían entre ellos.
- ***Raspberry Pi 1:*** A la que asigné la dirección IP 192.168.100.2.
- ***Raspberry Pi 1:*** A la que asigné la dirección IP 192.168.100.5. Para acceder a este nodo desde el que controlaba al resto necesitaba conectarme al nodo de IP acabada en 4 vía SSH, y ya desde este conectarme de la misma forma. No podía conectarme de forma directa ya que los dos dispositivos no estaban dentro del rango *wireless* del otro.

Ninguno de estos dispositivos contaba con adaptador *wireless*, es por ello que también se me han prestado cinco dispositivos USB para dotar a las *Raspberry* de conectividad inalámbrica. Mediante el uso del comando “lsusb” en la línea de comandos de debían pude obtener la descripción de los dispositivos:

Ralink technology, Conceptronic C54RU v3 (Ralink RT2571W)

Ralink es el encargado de fabricar el chipset de este dispositivo, que utiliza el driver “RT2571W”. El modelo del dispositivo USB comercializado por la marca Conceptronic corresponde al nombre C54RUv3.

No necesité instalar ningún *driver* adicional para obtener plena funcionalidad de ellos.

7.1.1 Añadiendo nodos a la red

Una vez tengamos nuestro nodo con dispositivo *wireless* listo para añadir a la red necesitaremos modificar el siguiente fichero en el sistema operativo de las *Raspberry,* independientemente de la distribución instalado este fichero será el localizado en la ruta “/etc/network/interfaces”.

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

auto wlan0
allow-hotplug wlan0
iface wlan0 inet static
address 192.168.100.1
netmask 255.255.255.0
network 192.168.100.0
broadcast 192.168.100.255
wireless-mode ad-hoc
wireless-channel 1
wireless-essid pi

iface default inet dhcp
```

Ilustración 25. Contenido del fichero /etc/network/interfaces del nodo 1

Como se puede observar, la interfaz "wlan0" corresponde al dispositivo *wireless* que se utiliza en el proyecto, y se configura de forma que al conectarlo se le asigne una dirección IP estática dentro de la red 192.168.100.0/24. El modo de comunicación que usará el dispositivo será ad-hoc, con este modo no se necesita que los nodos contacten con un punto de acceso para enviarse paquetes entre ellos.

Todos los nodos conectados a la red deberán tener el mismo *wireless-channel* y *wireless-essid* para poder comunicarse.

7.1.2 Utilidades a instalar

Tendremos que instalar las siguientes utilidades para poder poner el protocolo en marcha, así como para poder realizar los experimentos correctamente:

COMPILADOR G++ v 4.7

Necesitamos un compilador que soporte:

- La opción "-pthread", que permite generar código soportando hilos POSIX.
- La opción "-std=C++11" que nos permite compilar código escrito en el estándar C++11 de C++.

La versión que yo he utilizado ha sido g++-4.7 en todas las *Raspberry Pi* con las que he trabajado.

En caso de no disponer de esta versión deberíamos ejecutar los siguientes comandos en la *Raspberry*, que deberá estar conectada a internet:

```
sudo apt-get update
sudo apt-get install g++-4.7
```

Aceptamos la instalación y esperamos a que se descargue e instale.

Una vez tengamos instalado el compilador, podremos compilar el código mediante el comando:

```
g++-4.7 -std=c++11 -pthread routingProtocol.cp -o protocolo
```

Y podremos lanzarlo con el comando

```
./protocolo
```

En caso de no tener permiso usaremos lo siguiente y volveremos a probar el lanzamiento

```
chmod +x protocolo
```

UTILIDAD "iw" de línea de comandos

Como se comenta en la descripción de la tabla de vecinos ubicada en la explicación del fichero "neighborTable.h", la forma de localizar los vecinos que tiene el programa es mediante el uso de este comando, por ello necesitaremos instalarlo, previamente al uso del programa. Por ello lanzaremos los siguientes comandos en caso de no disponer de ella:

```
sudo apt-get update
sudo apt-get install iw
```

Aceptaremos la instalación y esperaremos a que se instale. Una vez instalado podremos comprobar en la línea de comandos el comando del que se sirve el sistema lanzando en la misma:

```
iw dev wlan0 station dump |grep 'Station|signal:'
```

La salida del mismo contendrá dos líneas representando dirección MAC y potencia de señal para cada nodo vecino de nuestro nodo que se encuentre en la misma red. Si no hay vecinos no obtendremos salida.

Utilidad SSH

En el caso extraño de que no tuviésemos esta utilidad instalada pasaríamos a instalarla ejecutando los comandos

```
sudo apt-get update
sudo apt-get install ssh
```

Aceptaríamos y esperaríamos a que instalase correctamente, momento en el cual podríamos intentar una conexión a algún nodo vecino que también disponga de la misma utilidad usando el comando

```
ssh <dirección ip del vecino>
```

Se nos pedirá la contraseña del vecino al que queremos conectar, normalmente es la misma que usamos para acceder al mismo de forma física.

Utilidades NTP y NTPDate

Estas utilidades las necesito si quiero realizar un registro de los paquetes enviados y recibidos por los distintos tiempos y ver mediante estos registros el tiempo que han tardado los mismos en llegar al destino desde el momento que salieron del emisor.

Son necesarias ya que para saber ese dato es necesario que todos los nodos tengan sus relojes sincronizados. En el caso final, donde se implemente la aplicación mediante el uso de módulos GPS podremos ahorrarnos esta sincronización de relojes y obtener la hora actual mediante ellos, que nos dan una precisión bastante buena.

Las entradas del registro están indexadas por el "*time_since_epoch*" de UNIX, que marca el tiempo transcurrido desde el "*epoch time*" correspondiente a la fecha 1/1/1970 con hora 00:00:00. Este "*time_since_epoch*" lo escribo en el registro con valores en microsegundos para tener una medición precisa de los tiempos.

Imaginemos dos archivos de registro de dos nodos que tienen sus relojes sincronizados. En este escenario podemos estar seguros (siempre que la sincronización sea lo suficientemente precisa) de que si en un momento determinado un registro escribe un *time_since_epoch* de 'X' en el registro del otro nodo el *time_since_epoch* sería prácticamente idéntico (hablando a nivel de microsegundos, ya que siempre hay algo de error).

No sería igual para el caso en el que los relojes no estuviesen sincronizados, ya que en el mismo instante de tiempo cada sistema operativo interpretaría su propio "*time_since_epoch*".

El caso es que estas funciones son necesarias para realizar nuestra experimentación por ello procedemos a instalar ambas mediante el uso de los comandos:

```
sudo apt-get update
sudo apt-get install ntp
sudo apt-get install ntpdate
```

Una vez las tengamos instaladas podemos proseguir con el proceso necesario para sincronizar los relojes. Vamos a suponer un escenario donde el nodo de la red con dirección IP 192.168.100.3 actúa como servidor para los demás. Para activar esta funcionalidad necesitaremos modificar el fichero /etc/ntp.conf de la siguiente forma.

```
# You do need to talk to an NTP server or two (or three).
#server ntp.your-provider.example

# pool.ntp.org maps to about 1000 low-stratum NTP servers.  Your server will
# pick a different set every time it starts up.  Please consider joining the
# pool: <http://www.pool.ntp.org/join.html>
server hora.roa.es iburst
server 0.debian.pool.ntp.org iburst
server 1.debian.pool.ntp.org iburst
server 2.debian.pool.ntp.org iburst
server 3.debian.pool.ntp.org iburst
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

Ilustración 26.Fragmento del archivo /etc/ntp.conf del nodo servidor NTP

Como se observa, las líneas empezando por server son las que indican de que servidores está nuestro nodo obteniendo la hora mediante el uso del protocolo NTP.

Cabe mencionar que las dos últimas líneas que se observan sirven para considerar una fuente de hora fiable al propio reloj del sistema, que podrá ser transferida al resto de nodos que quieran contactar con nuestro nodo actuando como servidor NTP.

```
 # Clients from this (example!) subnet have unlimited access, but only if
 # cryptographically authenticated.
 restrict 192.168.100.0 mask 255.255.255.0 nomodify nopeer notrap


 # If you want to provide time to your local subnet, change the next line.
 # (Again, the address is an example only.)
 broadcast 192.168.100.255
```

Ilustración 27.Fragmento del archivo /etc/ntp.conf del nodo servidor NTP

Con las dos líneas descomentadas permitimos al nodo que acepte peticiones de nodos que se encuentren dentro de la red 192.168.100.0/24 y que envíe su hora mediante *broadcast* a todos los nodos de la misma.

En el lado del cliente la modificación a realizar sería la siguiente:

```
# pool.ntp.org maps to about 1000 low-stratum NTP servers.  Your server will
# pick a different set every time it starts up.  Please consider joining the
# pool: <http://www.pool.ntp.org/join.html>
server 192.168.100.3 iburst
server 0.debian.pool.ntp.org iburst
server 1.debian.pool.ntp.org iburst
server 2.debian.pool.ntp.org iburst
server 3.debian.pool.ntp.org iburst
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

Ilustración 28.Fragmento del archivo /etc/ntp.conf del nodo cliente

Como se observa, ponemos como fuente de hora a nuestro servidor en marcha en el nodo 192.168.100.3.

Teniendo esto, podremos usar el siguiente comando para reiniciar NTP en los nodos y que se aplique la nueva configuración:

```
sudo /etc/init.d/ntp restart
```

Es posible que no se refresque la hora en el cliente, por ello forzaremos este hecho con la ayuda del comando ntpdate instalado previamente. He creado un script a ejecutar en los nodos clientes cada vez que queramos obtener la hora del servidor, ya que no era necesario únicamente el uso del comando ntpdate, previamente teníamos que detener el servicio NTP, ya que ambos usan el mismo puerto para funcionar.

```
#! /bin/bash
sudo /etc/init.d/ntp stop
sudo ntpdate 192.168.100.3
sudo /etc/init.d/ntp start
```

Ilustración 29. Fichero syncNodo.sh

Con la ejecución correcta de este script en un nodo, ya tendríamos la hora sincronizada con el servidor que hemos creado.

7.1.3 Obtención de coordenadas

Las coordenadas usadas para cada nodo son las que muestro a continuación, todas han sido obtenidas usando *Google Maps* y han sido testeadas mediante el programa generado compilando el fichero "FuzzyCalculator.cpp", que nos da el coste *fuzzy* de los nodos intermedios de nuestra red, y se comprueba que el orden en cuanto a mejor coste es el deseado con estas coordenadas.

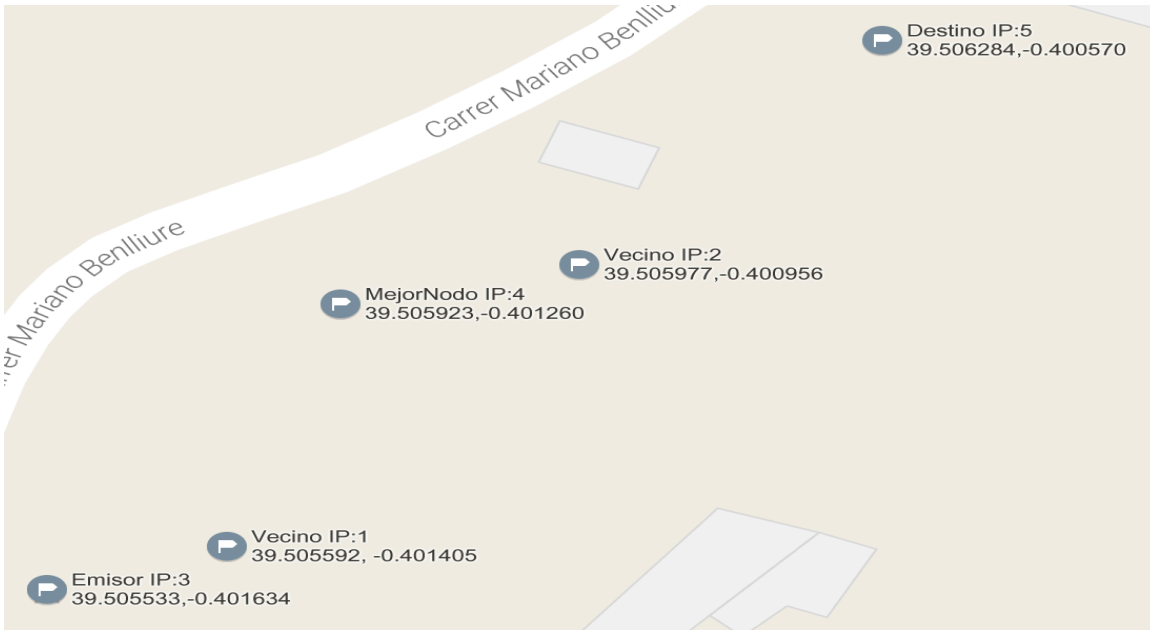


Ilustración 30. Coordenadas asignadas a los nodos suponiendo un rango de alcance de 100 metros

Estas coordenadas fueron puestas dentro de la función “coordenadasMAC” localizada en el fichero “routingProtocol.h”

7.2 Lanzamiento del programa

Teniendo todas estas utilidades instaladas podremos proceder al lanzamiento de los nodos mediante el programa compilado con el comando

```
g++-4.7 -std=c++11 -pthread routingProtocol.cp -o protocolo
```

Al lanzar “./protocolo” se nos pedirán 4 parámetros por pantalla, previo a la ejecución del protocolo en si, estos parámetros configurarán el período de envío de mensajes HELLO, STATUS y DATA. El último parámetro nos pregunta a cuantos nodos de la tabla queremos enviar mensajes DATA cada vez que enviamos.

Una vez cerremos el protocolo, podremos mirar los resultados en el fichero de registro llamado “logfile.txt”

8. Anexo

Los resultados de los experimentos los muestro en tablas, donde los nodos de las filas envían paquetes a los nodos de las columnas. La nomenclatura utilizada es la siguiente:

- %H, %D, %A, %S: Valores correspondientes al porcentaje de entrega correcta de paquetes HELLO, DATA, ACK y STATUS, respectivamente.
- TH,TS,TD,TA: Retardo en la entrega de paquetes de tipo HELLO, STATUS, DATA y ACK, respectivamente.
- "**********": No hay enlace entre los dos nodos

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=83.59 %A=98.63 %S=100 %D=100 TH=31,7ms TS=30,2ms TD=55ms TA=38,5ms	%H=63,77 %A=94,44 TH=105ms TA=260ms
Nodo 4	**********	**********	%H=57,03 %A=92,52 %S=95,08 TH=90ms TS=49,8ms TA=78.,1ms	**********	%H=75,96 %A=95,23 %S=96,77 TH=258ms TS=206ms TA=218ms
Nodo 5	**********	**********	%H=43,54 %A=98,76 TH=57,6ms TA=120ms	%H=66,66 %A=100 %S=100 TH=25ms TS=24ms TA=10ms	**********

Tabla 6. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 0.5 sec, Status_jitter = 1 sec, Período envío DATA = 3 sec. 1 Nodo envía a otro de la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=91,6 %A=98,7 %S=100 %D=100 TH=33,5ms TS=34,8ms TD=33,5ms TA=43ms	%H=83,8 %A=95,3 %D=95,23 TH=211ms TD=405ms TA=350ms
Nodo 4	**********	**********	%H=52,77 %A=87,78 %S=92.85 TH=82,3ms TS=49,5ms TA=73,1ms	**********	%H=79 %A=97,11 %S=95,77 TH=223ms TS=335ms TA=303ms
Nodo 5	**********	**********	%H=45,98 %A=90,51 TH=59,8ms TA=114ms	%H=74,63 %A=100 %S=100 TH=25,3ms TS=29,4ms TA=12ms	**********

Tabla 7. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 0.5 sec, Status_jitter = 1 sec, Período envío DATA = 3 sec. 1 Nodo envía a Todos los de la red

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=81,2 %A=98,76 %S=100 %D=100 TH=27ms TS=28ms TD=19ms TA=38ms	%H=71 %A=95,23 %D=95 TH=135ms TD=211ms TA=257ms
Nodo 4	**********	**********	%H=60,9 %A=94,39 %S=95,3 %D=90,47 TH=80,3ms TS=57ms TD=71,5ms TA=88,2ms	**********	%H=81,2 %A=95,87 %S=95,38 %D=95,23 TH=123ms TS=223ms TD=183ms TA=223ms
Nodo 5	**********	**********	%H=49,6 %A=93,181 %D=82,35 TH=49,1ms TD=149ms TA=140ms	%H=75,19 %A=100 %S=100 %D=100 TH=16,3ms TS=24,5ms TD=26,6ms TA=16,5ms	**********

Tabla 8. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 0.5 sec, Status_jitter = 1 sec, Período envío DATA = 3 sec. Todos los nodos envían al resto.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=96,96 %A=100 %S=100 %D=100 TH=19,6ms TS=4ms TD=4ms TA=4ms	%H=71,21 %A=100 TH=49,1ms TA=129ms
Nodo 4	**********	**********	%H=74,24 %A=100 %S=96,875 TH=31,2ms TS=16,2ms TA=48,4ms	**********	%H=72,72 %A=100 %S=100 TH=52,3ms TS=70,8ms TA=144ms
Nodo 5	**********	**********	%H=72,3 %A=100 TH=75,4ms TA=126ms	%H=76,56 %A=100 %S=100 TH=26ms TS=22ms TA=26ms	**********

Tabla 9. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 1 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. 1 Nodo envía a otro de la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=87,5 %A=100 %S=100 %D=100 TH=3,4ms TS=6,8ms TD=4,6ms TA=6,7ms	%H=70,31 %A=97,82 %D=100 TH=13,3ms TD=84ms TA=130ms
Nodo 4	**********	**********	%H=76,56 %A=98,21 %S=100 TH=45,9ms TS=14,1ms TA=34ms	**********	%H=71,87 %A=98,11 %S=96,77 TH=40,5ms TS=38,9ms TA=97ms
Nodo 5	**********	**********	%H=73,015 %A=100 TH=54ms TA=134ms	%H=84,74 %A=100 %S=100 TH=24,4ms TS=21ms TA=20,2ms	**********

Tabla 10. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 1 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. 1 Nodo envía al resto de nodos en la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=95,83 %A=100 %S=100 %D=100 TH=25,2ms TS=24,2ms TD=21,7ms TA=26,1ms	%H=71,62ms %A=97,29ms %D=94,44 TH=33,6ms TD=130ms TA=214ms
Nodo 4	**********	**********	%H=48,61 %A=95,58 %S=91,42 %D=94,117 TH=67,3ms TS=33,6ms TD=60,5ms TA=67,5ms	**********	%H=79,72 %A=94,23 %S=100 %D=94,44 TH=770ms TS=88ms TD=108ms TA=133ms
Nodo 5	**********	**********	%H=52,05 %A=86,27 %D=93,33 TH=63,3ms TD=120ms TA=133ms	%H=72,6 %A=100 %S=100 %D=100 TH=39,1ms TS=36,1ms TD=17,6ms TA=16,8ms	**********

Tabla 11. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 1 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. Todos los nodos envían al resto de nodos.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=80 %A=100 %S=100 %D=100 TH=7ms TS=36,7ms TD=61,1ms TA=62,3ms	%H=46,6 %A=90,9 TH=26,3ms TA=198ms
Nodo 4	**********	**********	%H=40 %A=95,83 %S=100 TH=36,4ms TS=35,6ms TA=31,6ms	**********	%H=80 %A=89,47 %S=100 TH=33,1ms TS=57ms TA=131ms
Nodo 5	**********	**********	%H=36,66 %A=85,71 TH=35,5ms TA=144ms	%H=63,33 %A=100 %S=100 TH=18ms TS=30,4ms TA=18ms	**********

Tabla 12. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 2 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. 1 Nodo envía a otro en la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=91,66 %A=100 %S=100 %D=100 TH=9,3ms TS=17,8ms TD=32,6ms TA=25,2ms	%H=66,66 %A=94,73 %D=100 TH=30,3ms TD=102ms TA=113ms
Nodo 4	**********	**********	%H=61,11 %A=81,25 %S=91,42 TH=35ms TS=24,3ms TA=49,5ms	**********	%H=86,11 %A=100 %S=100 TH=37,8ms TS=50,4ms TA=78,7ms
Nodo 5	**********	**********	%H=55,55 %A=86,95 TH=45,5ms TA=137ms	%H=77,77 %A=100 %S=100 TH=18ms TS=81,6ms TA=17,8ms	**********

Tabla 13. Experimento estático. Tam. paquete 1024 B, Hello_jitter =2 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. 1 Nodo envía al resto en la red

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=77,14 %A=100 %S=100 %D=100 TH=14ms TS=43,8ms TD=35,9ms TA=52,5ms	%H=68,57 %A=94,44 %D=93,75 TH=22,6ms TD=125ms TA=150ms
Nodo 4	**********	**********	%H=54,28 %A=100 %S=100 %D=100 TH=61,9ms TS=48,5ms TD=79,5ms TA=82,8ms	**********	%H=74,28 %A=95,83 %S=97,05 %D=94,17 TH=64ms TS=66,5ms TD=84,6ms TA=74,3ms
Nodo 5	**********	**********	%H=51,42 %A=95,65 %D=93,33 TH=54,7ms TD=131ms TA=143ms	%H=68,57 %A=100 %S=100 %D=100 TH=15,1ms TS=92ms TD=32ms TA=32,7ms	**********

Tabla 14. Experimento estático. Tam. paquete 1024 B, Hello_jitter = 2 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. Todos envían al resto de nodos en la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=92,968 %A=100 %S=100 %D=100 TH=18ms TS=22,4ms TD=26,7ms TA=23,8ms	%H=77,95 %A=94,6 TH=193ms TA=292ms
Nodo 4	**********	**********	%H=46,87 %A=88,13 %S=81,96 TH=86,2ms TS=81,1ms TA=88,5ms	**********	%H=77,34 %A=92,3 %S=95 TH=248ms TS=270ms TA=303ms
Nodo 5	**********	**********	%H=30,327 %A=94,791 TH=91,4ms TA=160ms	%H=73,98 %A=100 %S=100 TH=32ms TS=58,4ms TA=34,7ms	**********

Tabla 15. Experimento estático. Tam. paquete 128 B, Hello_jitter = 0.5 sec, Status_jitter = 1 sec, Período envío DATA = 3 sec. 1 Nodo envía a otro de la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=81,95 %A=100 %S=98,41 %D=100 TH=15ms TS=19,1ms TD=29,1ms TA=29,1ms	%H=65,15 %A=93,75 %D=94,73 TH=102ms TD=166ms TA=230ms
Nodo 4	**********	**********	%H=47,36 %A=94,54 %S=95,31 TH=54,7ms TS=89,8ms TA=87ms	**********	%H=75,37 %A=97,05 %S=96,923 TH=106ms TS=195ms TA=188ms
Nodo 5	**********	**********	%H=36,71 %A=94,25 TH=73ms TA=138ms	%H=78,46 %A=100 %S=100 TH=14,2ms TS=16,2ms TA=17,3ms	**********

Tabla 16. Experimento estático. Tam. paquete 128 B, Hello_jitter = 0.5 sec, Status_jitter = 1 sec, Período envío DATA = 3 sec. 1 Nodo envía a Todos los de la red

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=86,82 %A=96,55 %S=100 %D=100 TH=26,1ms TS=23,4ms TD=23,4ms TA=31,7ms	%H=66,66 %A=93,75 %D=100 TH=117ms TD=274ms TA=266ms
Nodo 4	**********	**********	%H=44,186 %A=90,09 %S=90,16 %D=95 TH=96,5ms TS=74,5ms TD=108ms TA=88,4ms	**********	%H=76,153 %A=95,505 %S=96,774 %D=100 TH=226ms TS=224ms TD=197ms TA=202ms
Nodo 5	**********	**********	%H=39,51 %A=92,941 %D=75 TH=73ms TD=148ms TA=160ms	%H=71,2 %A=100 %S=100 %D=100 TH=17,7ms TS=18,1ms TD=17,9ms TA=15,8ms	**********

Tabla 17. Experimento estático. Tam. paquete 128 B, Hello_jitter = 0.5 sec, Status_jitter = 1 sec, Período envío DATA = 3 sec. Todos los nodos envían al resto.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=87,3 %A=100 %S=100 %D=100 TH=12,7ms TS=17,3ms TD=21,2ms TA=20,2ms	%H=60,317 %A=95,83 TH=54ms TA=195ms
Nodo 4	**********	**********	%H=50,79 %A=85,18 %S=93,54 TH=57,4ms TS=40,7ms TA=55,5ms	**********	%H=68,25 %A=100 %S=100 TH=62,4ms TS=66,2ms TA=145ms
Nodo 5	**********	**********	%H=37,09 %A=94,59 TH=40,1ms TA=141ms	%H=74,19 %A=100 %S=100 TH=13ms TS=17ms TA=12,1ms	**********

Tabla 18. Experimento estático. Tam. paquete 128 B, Hello_jitter = 1 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. 1 Nodo envía a otro de la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=93,15 %A=100 %S=100 %D=100 TH=7ms TS=12,8ms TD=13ms TA=18,2ms	%H=69,86 %A=96,66 %D=100 TH=41,8ms TD=111ms TA=140ms
Nodo 4	**********	**********	%H=46,57 %A=100 %S=97,22 TH=13,9ms TS=44,4ms TA=47,8ms	**********	%H=69,86 %A=98,07 %S=97,22 TH=45,31ms TS=74ms TA=118ms
Nodo 5	**********	**********	%H=43.05 %A=96 TH=56,4ms TA=139ms	%H=72,22 %A=100 %S=100 TH=10,8ms TS=21,7ms TA=14,22ms	**********

Tabla 19. Experimento estático. Tam. paquete 128 B, Hello_jitter = 1 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. 1 Nodo envía al resto de nodos en la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=92,95 %A=100 %S=100 %D=100 TH=3,67ms TS=5ms TD=8,2ms TA=5,9ms	%H=63,38 %A=95,23 %D=100 TH=30,2ms TD=89ms TA=110ms
Nodo 4	**********	**********	%H=76.05 %A=100 %S=97.05 %D=100 TH=31ms TS=13,3ms TD=42,3ms TA=34,4ms	**********	%H=63,38 %A=96 %S=94,285 %D=94,117 TH=38,4ms TS=59,8ms TD=84,5ms TA=92,6ms
Nodo 5	**********	**********	%H=60 %A=97,77 %D=100 TH=57,3ms TD=121ms TA=120ms	%H=71,42 %A=100 %S=100 %D=100 TH=13,3ms TS=84,5ms TD=23,1ms TA=13,3ms	**********

Tabla 20. Experimento estático. Tam. paquete 128 B, Hello_jitter = 1 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. Todos los nodos envían al resto de nodos.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=94,28 %A=100 %S=100 %D=100 TH=1,6ms TS=2,3ms TD=1ms TA=3,1ms	%H=74,28 %A=96,42 TH=17,5ms TA=77,6ms
Nodo 4	**********	**********	%H=82,85 %A=100 %S=100 TH=2ms TS=13,3ms TA=23ms	**********	%H=62,85 %A=93,54 %S=97,14 TH=51,5ms TS=37,9ms TA=56ms
Nodo 5	**********	**********	%H=80 %A=100 TH=44ms TA=120ms	%H=88,57ms %A=100 %S=100 TH=9ms TS=87ms TA=20,1ms	**********

Tabla 21. Experimento estático. Tam. paquete 128 B, Hello_jitter = 2 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. 1 Nodo envía a otro en la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=87,5 %A=100 %S=100 %D=100 TH=1ms TS=6,3ms TD=7,5ms TA=7,5ms	%H=65 %A=92,85 %D=100 TH=38,5ms TD=72,2ms TA=77,2ms
Nodo 4	**********	**********	%H=85 %A=100 %S=100 TH=14,9ms TS=9,6ms TA=20,2ms	**********	%H=77,5 %A=94,11 %S=97,43 TH=49,8ms TS=44ms TA=94,1 ms
Nodo 5	**********	**********	%H=70 %A=100 TH=54ms TA=110ms	%H=85 %A=100 %S=100 TH=12,6ms TS=17,4ms TA=16,3ms	**********

Tabla 22. Experimento estático. Tam. paquete 128 B, Hello_jitter =2 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. 1 Nodo envía al resto en la red

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	**********	**********	**********	**********
Nodo 2	**********	**********	**********	**********	**********
Nodo 3	**********	**********	**********	%H=81,39 %A=100 %S=100 %D=100 TH=7,1ms TS=15,7ms TD=23,8ms TA=10,4ms	%H=67,44 %A=96,15 %D=100 TH=25,6ms TD=96,2ms TA=97,3ms
Nodo 4	**********	**********	%H=69,76 %A=100 %S=95,23 %D=90,47 TH=38ms TS=39ms TD=66,5ms TA=41,7ms	**********	%H=81,39 %A=100 %S=97,61 %D=95,23 TH=40,8ms TS=63,8ms TD=78,4ms TA=74,6ms
Nodo 5	**********	**********	%H=60,46 %A=96,551 %D=88,88 TH=35,6ms TD=130ms TA=121ms	%H=86,04 %A=100 %S=100 %D=100 TH=9,8ms TS=64ms TD=13,3ms TA=10ms	**********

Tabla 23. Experimento estático. Tam. paquete 128 B, Hello_jitter = 2 sec, Status_jitter = 2 sec, Período envío DATA = 4 sec. Todos envían al resto de nodos en la red.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=100 %A=100 %S=100 TH=139ms TS=131ms TA=40ms	%H=96,55 %A=100 %S=100 TH=32,5ms TS=18ms TA=37,8ms	%H=100 %A=100 %S=100 TH=28,8ms TS=22,8ms TA=29,2ms	%H=83 %A=81,81 %S=92,3 TH=586ms TS=438ms TA=438ms
Nodo 2	%H=100 %A=100 %S=100 TH=21ms TS=51ms TA=49ms	**********	%H=97,727 %A=100 %S=100 TH=4,9ms TS=113ms TA=17,8s	%H=100 %A=100 %S=100 TH=6,2ms TS=17,8ms TA=16ms	%H=95,45 %A=100 %S=100 TH=360ms TS=409ms TA=427ms
Nodo 3	%H=94,82 %A=100 %S=100 %D=100 TH=15,7ms TS=3,1ms TD=10,4ms TA=15,3ms	%H=93,023 %A=100 %S=100 TH=61,9ms TS=49,4ms TA=23,8ms	**********	%H=87,5 %A=95,83 %S=100 TH=28ms TS=12,2ms TA=15,5ms	%H=75 %A=91 TH=205ms TA=539ms
Nodo 4	%H=100 %A=100 %S=100 TH=66ms TS=134ms TA=147ms	%H=100 %A=100 %S=100 TH=113ms TS=303ms TA=185ms	%H=100 %A=100 %S=100 TH=71,4ms TS=153ms TA=156ms	**********	%H=100 %A=100 %S=100 TH=476ms TS=107ms TA=604ms
Nodo 5	%H=96,49 %A=95,83 %S=100 TH=36,8ms TS=22,8ms TA=68,8ms	%H=100 %A=100 %S=100 TH=57,4ms TS=108ms TA=94,3ms	%H=78,87 %A=93 TH=50ms TA=176ms	%H=100 %A=100 %S=100 TH=20,3ms TS=33,3ms TA=16,8ms	**********

Tabla 24. Experimento dinámico 1. Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Nodo 3 envía a un solo nodo de la red

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=97,727 %A=100 %S=100 TH=78,7ms TS=103ms TA=140ms	%H=93,22 %A=96,42 %S=100 TH=22,2ms TS=29,2ms TA=39,7ms	%H=95,65 %A=100 %S=100 TH=24,8ms TS=38ms TA=30ms	%H=84,74 %A=67,3 %S=80,76 TH=943ms TS=919ms TA=997ms
Nodo 2	%H=100 %A=100 %S=100 TH=17,3ms TS=33ms TA=14,8ms	**********	%H=100 %A=100 %S=100 TH=21,3ms TS=124ms TA=11,2ms	%H=95,83 %A=100 %S=100 TH=9,3ms TS=17,7ms TA=5,3ms	%H=100 %A=100 %S=100 TH=1000ms TS=1100ms TA=1111ms
Nodo 3	%H=96,55 %A=100 %S=96,55 %D=100 TH=17,8ms TS=4,4ms TD=1,7ms TA=6,5ms	%H=95,45 %A=100 %S=95 %D=100 TH=62,9ms TS=81,6ms TD=62,4ms TA=110ms	**********	%H=79,16 %A=100 %S=100 %D=100 TH=12,9ms TS=13ms TD=11,7ms TA=7,9ms	%H=72,2 %A=88,23 %D=78,57 TH=752ms TD=964ms TA=1100ms
Nodo 4	%H=91,304 %A=100 %S=100 TH=72,1ms TS=101,7ms TA=117,7ms	%H=91,6 %A=100 %S=100 TH=78,8ms TS=206ms TA=168ms	%H=92 %A=100 %S=100 TH=70,4ms TS=129ms TA=127ms	**********	%H=92 %A=86,95 %S=100 TH=932ms TS=1290ms TA=1060ms
Nodo 5	%H=91,22 %A=97,77 %S=100 TH=68,9ms TS=38,6ms TA=98,5ms	%H=95,34 %A=100 %S=100 TH=93,1ms TS=142ms TA=163ms	%H=71,83 %A=95,23 TH=83,9ms TA=214ms	%H=95,83 %A=100 %S=100 TH=32,1ms TS=31ms TA=30ms	**********

Tabla 25. Experimento dinámico 2 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Nodo 3 envía a todos los nodos de la red

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=97,72 %A=100 %S=100 %D=100 TH=103ms TS=226ms TD=206ms TA=148ms	%H=93,65 %A=100 %S=100 %D=100 TH=29,9ms TS=65,5ms TD=69,6ms TA=38,4ms	%H=100 %A=100 %S=100 %D=100 TH=24ms TS=40,7ms TD=64,8ms TA=44,1ms	%H=85,71 %A=73,68 %S=67,85 %D=73,33 TH=1200ms TS=953ms TD=1087ms TA=1353ms
Nodo 2	%H=100 %A=100 %S=100 %D=100 TH=27,5ms TS=16,6ms TD=8,3ms TA=5ms	**********	%H=100 %A=100 %S=100 %D=100 TH=34,2ms TS=132ms TD=19ms TA=22ms	%H=95,65 %A=100 %S=100 %D=100 TH=17,5ms TS=25,2ms TD=7,3ms TA=5,9ms	%H=90,9 %A=92,85 %S=100 %D=100 TH=1670ms TS=1750ms TD=1764ms TA=1520ms
Nodo 3	%H=95,16 %A=100 %S=100 %D=100 TH=8ms TS=4,4ms TD=3,3ms TA=12,7ms	%H=93,18 %A=100 %S=100 %D=100 TH=70,8ms TS=110ms TD=121ms TA=125ms	**********	%H=87,5 %A=100 %S=100 %D=100 TH=22,2ms TS=12,2ms TD=14,5ms TA=9,8ms	%H=73,17 %A=77,19 %D=80 TH=1024ms TD=1349ms TA=1373ms
Nodo 4	%H=100 %A=100 %S=90,9 %D=80 TH=85,3ms TS=79,6ms TD=149ms TA=120ms	%H=100 %A=100 %S=90,9 %D=80 TH=102ms TS=151ms TD=303ms TA=175ms	%H=96 %A=95,23 %S=81,81 %D=80 TH=85,3ms TS=124ms TD=251ms TA=121ms	**********	%H=92 %A=69,56 %S=72,72 %D=80 TH=1976ms TS=2038ms TD=2300ms TA=1873ms
Nodo 5	%H=91,8 %A=93,181 %S=100 %D=100 TH=46,9ms TS=31,8ms TD=109ms TA=123ms	%H=97,67 %A=100 %S=100 %D=100 TH=148ms TS=186ms TD=196ms TA=156ms	%H=70,37 %A=90,47 %D=87,5 TH=75,8ms TD=294ms TA=250ms	%H=95,65 %A=100 %S=100 %D=100 TH=35,1ms TS=31,2ms TD=105ms TA=51ms	**********

Tabla 26. Experimento dinámico 3 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Todos los nodos envían al resto

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=97,72 %A=100 %S=100 TH=68,4ms TS=208ms TA=99,1ms	%H=96,61 %A=100 %S=100 TH=32,6ms TS=70,6ms TA=56,7ms	%H=95,83 %A=100 %S=100 TH=50,6ms TS=218ms TA=57,7ms	%H=88,13 %A=62,74 %S=66,66 TH=738ms TS=964ms TA=975ms
Nodo 2	%H=100 %A=100 %S=100 TH=16,4ms TS=4ms TA=9,7ms	**********	%H=100 %A=100 %S=100 TH=21,7ms TS=27,9ms TA=9,1ms	%H=92 %A=100 %S=100 TH=9ms TS=199ms TA=13ms	%H=93,18 %A=97,61 %S=100 TH=1027ms TS=1114ms TA=987ms
Nodo 3	%H=91,38 %A=100 %S=100 %D=100 TH=12,9ms TS=5,8ms TD=7,4ms TA=5,8ms	%H=93 %A=100 %S=100 %D=100 TH=71,3ms TS=71ms TD=48,3ms TA=53,6ms	**********	%H=84 %A=100 %S=100 %D=100 TH=52,3ms TS=7,3ms TD=5ms TA=10,4ms	%H=70,83 %A=80 %D=61,53 TH=670ms TD=1124ms TA=1180ms
Nodo 4	%H=100 %A=100 %S=100 TH=84,5ms TS=109ms TA=123ms	%H=100 %A=100 %S=100 TH=137ms TS=258ms TA=134ms	%H=100 %A=80,95 %S=100 TH=95,5ms TS=172ms TA=146,5ms	**********	%H=96,15 %A=91,66 %S=100 TH=1567ms TS=1710ms TA=1515ms
Nodo 5	%H=87,71 %A=95,34 %S=100 %D=88,88 TH=45,5ms TS=20,4ms TD=88,1ms TA=101ms	%H=95,34 %A=100 %S=100 %D=100 TH=79ms TS=79,2ms TD=78,1ms TA=141,1ms	%H=70,42 %A=92,50 %D=88,88 TH=71,7ms TD=205ms TA=194ms	%H=100 %A=100 %S=100 %D=100 TH=34,8ms TS=36,9ms TD=9,7ms TA=20,6ms	**********

Tabla 27. Experimento dinámico 4 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Nodo 3 y nodo 5 envían a todos.

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=100 %A=100 %S=100 %D=100 TH=62ms TS=48ms TD=159ms TA=70ms	%H=98,3 %A=100 %S=100 %D=100 TH=34,8ms TS=22,4ms TD=69,9ms TA=42ms	%H=95,65 %A=100 %S=100 %D=100 TH=28ms TS=30,7ms TD=63ms TA=41,3ms	%H=88,13 %A=38,8 %S=28,57 %D=42,85 TH=1037ms TS=841ms TD=1487ms TA=1626ms
Nodo 2	%H=97,67 %A=100 %S=100 %D=100 TH=5ms TS=7,4ms TD=6,7ms TA=4,5ms	**********	%H=95,45 %A=100 %S=100 %D=100 TH=6,2ms TS=132ms TD=6,1ms TA=8ms	%H=100 %A=100 %S=100 %D=100 TH=10,5ms TS=26ms TD=3ms TA=11ms	%H=88,63 %A=95,23 %S=100 %D=88,8 TH=1377ms TS=1683ms TD=1461ms TA=1578ms
Nodo 3	%H=98,275 %A=100 %S=100 TH=6,8ms TS=3,5ms TA=3,7ms	%H=97,67 %A=100 %S=100 TH=88,5ms TS=43,1ms TA=28,2ms	**********	%H=91,6 %A=100 %S=100 TH=30ms TS=5,3ms TA=6,5ms	%H=77,46 %A=67,3 TH=971ms TA=1596ms
Nodo 4	%H=100 %A=100 %S=91,6 %D=83,3 TH=86,8ms TS=150ms TD=171ms TA=143ms	%H=100 %A=92 %S=91,6 %D=83,3 TH=103ms TS=194ms TD=230ms TA=125ms	%H=100 %A=100 %S=91,6 %D=100 TH=101ms TS=172,5ms TD=179ms TA=150ms	**********	%H=76 %A=86,95 %S=75 %D=50 TH=1808ms TS=2213ms TD=2325ms TA=2165ms
Nodo 5	%H=92,982 %A=95,238 %S=100 TH=43ms TS=56,2ms TA=141ms	%H=95,34 %A=100 %S=100 TH=68,5ms TS=82,7ms TA=113ms	%H=72,85 %A=90,24 TH=68,7ms TA=179ms	%H=94,73 %A=100 %S=100 TH=65,8ms TS=33,9ms TA=7,6ms	**********

Tabla 28. Experimento dinámico 5 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Nodos 1,2,4 envían a todos

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=97,56 %A=100 %S=94,73 TH=80ms TS=172ms TA=66,6ms	%H=92,98 %A=100 %S=92,85 TH=50,8ms TS=55,2ms TA=45ms	%H=100 %A=100 %S=100 TH=74,7 TS=33,4 TA=40,4ms	%H=87,71 %A=64,15 %S=50 TH=1000ms TS=1067ms TA=1136ms
Nodo 2	%H=100 %A=100 %S=100 TH=30,5ms TS=62,6ms TA=15,8ms	**********	%H=100 %A=100 %S=100 TH=18,2ms TS=121ms TA=6,7ms	%H=92 %A=100 %S=100 TH=9,8ms TS=22ms TA=13,3ms	%H=88,63 %A=90,697 %S=94,736 TH=1526ms TS=1504ms TA=1388ms
Nodo 3	%H=92,85 %A=100 %S=100 %D=100 TH=38,4ms TS=4,8ms TD=85ms TA=13,9ms	%H=88,3 %A=100 %S=100 %D=100 TH=98ms TS=84ms TD=49ms TA=38ms	**********	%H=92 %A=100 %S=100 %D=100 TH=32ms TS=8,4ms TD=5,6ms TA=7,2ms	%H=66,66 %A=72,72 %D=71,42 TH=1120ms TD=1538ms TA=1179ms
Nodo 4	%H=100 %A=100 %S=100 %D=83,3 TH=74,5ms TS=138ms TD=178ms TA=150ms	%H=100 %A=95,65 %S=100 %D=83,3 TH=104ms TS=211ms TD=235ms TA=132ms	%H=96,153 %A=100 %S=100 %D=83,33 TH=100ms TS=152ms TD=164ms TA=149ms	**********	%H=92,3 %A=87,5 %S=66,66 %D=66,66 TH=1499,s TS=1628ms TD=1874ms TA=1584ms
Nodo 5	%H=94,54 %A=93,47 %S=100 TH=74,3ms TS=23ms TA=145ms	%H=97,67 %A=100 %S=100 TH=77,8ms TS=78ms TA=121ms	%H=78,87 %A=91,66 TH=74,6ms TA=176ms	%H=100 %A=100 %S=100 TH=44ms TS=19,7ms TA=7,2ms	**********

Tabla 29. Experimento dinámico 6 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Nodos 3 y 4 envían a todos

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=97.67 %A=100 %S=100 %D=100 TH=155ms TS=106ms TD=149ms TA=164ms	%H=98,3 %A=100 %S=100 %D=100 TH=25ms TS=18,5ms TD=47ms TA=42,4ms	%H=95,83 %A=100 %S=100 %D=100 TH=28,5ms TS=32,2ms TD=43ms TA=38,2ms	%H=84,74 %A=94,44 %S=96,42 %D=100 TH=664ms TS=565ms TD=150ms TA=545ms
Nodo 2	%H=97,61 %A=100 %S=100 %D=100 TH=30,3ms TS=7,2ms TD=12,3ms TA=16,4ms	**********	%H=97,67 %A=100 %S=100 %D=100 TH=78,5ms TS=127ms TD=44ms TA=16,6ms	%H=100 %A=100 %S=100 %D=100 TH=24,8ms TS=31,1ms TD=16,8ms TA=13,9ms	%H=93 %A=100 %S=100 %D=100 TH=609ms TS=668ms TD=278ms TA=637ms
Nodo 3	%H=94,82 %A=100 %S=100 %D=100 TH=5ms TS=7,6ms TD=7,1ms TA=13,5ms	%H=95,34 %A=100 %S=100 %D=100 TH=90,5ms TS=100ms TD=122ms TA=160ms	**********	%H=95,83 %A=100 %S=100 %D=100 TH=24,7ms TS=14,5ms TD=25,3ms TA=12,9ms	%H=72,22 %A=90,566 %D=75 TH=357ms TD=458ms TA=740ms
Nodo 4	%H=100 %A=100 %S=90,9 %D=80 TH=151ms TS=130ms TD=178ms TA=159ms	%H=100 %A=100 %S=90,9 %D=80 TH=194ms TS=152ms TD=185ms TA=143ms	%H=100 %A=100 %S=81,81 %D=80 TH=130ms TS=144ms TD=198ms TA=128ms	**********	%H=96 %A=95,65 %S=81,81 TH=1188ms TS=1088ms TA=926ms
Nodo 5	%H=92,98 %A=100 %S=100 %D=100 TH=42m TS=11,5ms TD=35,2ms TA=32,5ms	%H=97,61 %A=100 %S=100 %D=100 TH=63,9ms TS=131,5ms TD=105ms TA=98ms	%H=74,64 %A=93,18 %D=100 TH=58,3 TD=287ms TA=221ms	%H=95,83 %A=100 %S=100 %D=100 TH=16,8ms TS=32,8ms TD=1ms TA=8,1ms	**********

Tabla 30. Experimento dinámico 7 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Todos envían a dos nodos

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=95,45 %A=100 %S=100 TH=26,5ms TS=147ms TA=49ms	%H=94,91 %A=100 %S=100 TH=21,7ms TS=33,9ms TA=43,4ms	%H=95,83 %A=100 %S=100 TH=13,8ms TS=35,1ms TA=45,4ms	%H=86,44 %A=85,71 %S=89,28 TH=210ms TS=488ms TA=338ms
Nodo 2	%H=100 %A=100 %S=100 TH=2,3ms TS=6,2ms TA=24,9ms	**********	%H=95,45 %A=100 %S=100 TH=5ms TS=27,4ms TA=23,4ms	%H=95,83 %A=100 %S=100 TH=7,9ms TS=22,4ms TA=9,3ms	%H=90,9 %A=100 %S=100 TH=315ms TS=355ms TA=408ms
Nodo 3	%H=93,1 %A=100 %S=100 %D=100 TH=14,7ms TS=5ms TD=7ms TA=11,5ms	%H=90,69 %A=100 %S=100 %D=100 TH=88ms TS=54ms TD=46ms TA=25ms	**********	%H=95,83 %A=100 %S=100 %D=100 TH=31,6ms TS=12,3ms TD=4,6ms TA=12ms	%H=70 %A=92,59 %D=87,5 TH=248ms TD=259ms TA=439ms
Nodo 4	%H=79,16 %A=86,95 %S=81,81 TH=97ms TS=85,8ms TA=102ms	%H=79,16 %A=100 %S=83,3 TH=209ms TS=289ms TA=112ms	%H=76 %A=87,5 %S=81,81 TH=127ms TS=141ms TA=126ms	**********	%H=72 %A=100 %S=83,3 TH=660ms TS=951ms TA=691ms
Nodo 5	%H=96,49 %A=100 %S=100 %D=100 TH=27,1ms TS=9,2ms TD=2,9ms TA=30,6ms	%H=95,34 %A=100 %S=100 %D=100 TH=60,7ms TS=71,2ms TD=76,1ms TA=78,3ms	%H=78,26 %A=95 %D=88,8 TH=43,5ms TD=227ms TA=155ms	%H=95,65 %A=100 %S=100 TH=32,3ms TS=23ms TA=32,3ms	**********

Tabla 31. Experimento dinámico 8 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Nodo 3 y 5 envían a dos nodos

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=100 %A=100 %S=100 TH=100ms TS=219ms TA=80ms	%H=98,3 %A=100 %S=100 TH=35,6ms TS=52,5ms TA=39,6ms	%H=100 %A=100 %S=100 TH=65,5ms TS=34,4ms TA=35ms	%H=83,05 %A=84,9 %S=89,6 TH=501ms TS=601ms TA=535ms
Nodo 2	%H=100 %A=100 %S=100 %D=88,88 TH=4ms TS=7ms TD=4,3ms TA=5,2ms	**********	%H=97,67 %A=100 %S=100 %D=100 TH=7ms TS=124ms TD=59,4ms TA=10,7ms	%H=96 %A=100 %S=100 %D=100 TH=15,7ms TS=26,4ms TD=6,6ms TA=14ms	%H=90,69 %A=97,56 %S=100 %D=100 TH=536ms TS=735ms TD=527ms TA=678ms
Nodo 3	%H=94,82 %A=100 %S=100 %D=100 TH=10ms TS=4,3ms TD=44,5ms TA=19,7ms	%H=92,85 %A=100 %S=100 %D=100 TH=113ms TS=98ms TD=128ms TA=61ms	**********	%H=96 %A=100 %S=91,6 %D=100 TH=17,5ms TS=8,1ms TD=3,5ms TA=11,3ms	%H=66,66 %A=85,18 %D=87,5 TH=410ms TD=279ms TA=553ms
Nodo 4	%H=95,83 %A=88 %S=91,6 TH=69,4ms TS=91,5ms TA=93,4ms	%H=96 %A=100 %S=91,6 TH=128ms TS=279ms TA=132ms	%H=96,15 %A=95,83 %S=91,66 TH=99,7ms TS=136ms TA=93,7ms	**********	%H=88,46 %A=95,65 %S=75 TH=797ms TS=861ms TA=829ms
Nodo 5	%H=91,22 %A=100 %S=100 %D=100 TH=40,4ms TS=17,2ms TD=105,6ms TA=40,2ms	%H=97,61 %A=100 %S=100 %D=100 TH=82,3ms TS=107ms TD=104ms TA=168ms	%H=76,05 %A=95,12 %D=88,8 TH=60,4ms TD=239ms TA=189ms	%H=95,45 %A=100 %S=100 TH=22ms TS=12ms TA=11,3ms	**********

Tabla 32. Experimento dinámico 9 Tam.Paquete=128B, tasaHello=1sec, tasaStatus=2sec, tasaData=4sec. Nodos 2,3,5 envían a dos

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=68,83 %A=41,66 %S=61,76 %D=41,17 TH=1723ms TS=1815ms TD=1877ms TA=1708ms	%H=70,29 %A=95,74 %S=95,74 %D=100 TH=68,1ms TS=58,5ms TD=111ms TA=88ms	%H=91,11 %A=100 %S=100 %D=100 TH=40ms TS=53ms TD=83ms TA=53ms	%H=47,11 %A=35,95 %S=39,58 %D=50 TH=2000ms TS=2200ms TD=2100ms TA=2257ms
Nodo 2	%H=98,64 %A=100 %S=100 %D=91,66 TH=32,3ms TS=10,2ms TD=20,1ms TA=27,9ms	**********	%H=94,73 %A=100 %S=100 %D=100 TH=58,7ms TS=66,8ms TD=67,3ms TA=67,3ms	%H=100 %A=100 %S=100 %D=100 TH=26,7ms TS=45,7ms TD=36,7ms TA=26,3ms	%H=44,73 %A=58,82 %S=62,96 %D=53,84 TH=1999ms TS=2220ms TD=2299ms TA=2222ms
Nodo 3	%H=93,06 %A=100 %S=100 %D=100 TH=23,5ms TS=25,7ms TD=15,5ms TA=13ms	%H=66,23 %A=48,61 %S=45,16 %D=62,5 TH=1680ms TS=1560ms TD=1723ms TA=1675ms	**********	%H=91,48 %A=44,73 %D=55,55 TH=33,6 TD=1118ms TA=1765ms	%H=44,16 %A=46,83 %D=50 TH=1740ms TD=3481ms TA=3411ms
Nodo 4	%H=95,5 %A=95,12 %S=90 %D=80 TH=43ms TS=56,5ms TD=45,5ms TA=57,8ms	%H=78,26 %A=33,33 %S=47,61 %D=50 TH=1750ms TS=1780ms TD=1880ms TA=1650ms	%H=81,25 %A=97,72 %D=100 TH=63,6ms TD=135ms TA=163ms	**********	%H=39,58 %A=48,83 %S=47,61 %D=50 TH=1880ms TS=1975ms TD=2500ms TA=2036ms
Nodo 5	%H=88,88 %A=63,33 %S=100 %D=54,44 TH=31ms TS=5,4ms TD=1854ms TA=1655ms	%H=66,21 %A=57,57 %S=66,66 %D=45,45 TH=1680ms TS=1620ms TD=1460ms TA=1650ms	%H=76,72 %A=54,54 %D=58,33 TH=55,7 TD=1546ms TA=1823ms	%H=93,02 %A=100 %S=100 %D=100 TH=26ms TS=35,7ms TD=32,9ms TA=15,5ms	**********

Tabla 33. Experimento dinámico 10 Tam.Paquete=128B, tasaHello=0.5sec, tasaStatus=1sec, tasaData=2sec. Todos envían a todos

ENLACES	Nodo 1	Nodo 2	Nodo 3	Nodo 4	Nodo 5
Nodo 1	**********	%H=100 %A=100 %S=100 %D=100 TH=11,3ms TS=66,7ms TD=47,2ms TA=49,4ms	%H=93,33 %A=100 %S=96,42 %D=100 TH=11,7ms TS=21,1ms TD=39,7ms TA=33,7ms	%H=100 %A=100 %S=100 %D=100 TH=12,2ms TS=33,9ms TD=25,3ms TA=21,4ms	%H=82,75 %A=73 %S=74 %D=78,6 TH=27,9ms TS=288ms TD=183ms TA=137ms
Nodo 2	%H=100 %A=100 %S=100 %D=100 TH=19,3ms TS=18,6ms TD=11ms TA=15,1ms	**********	%H=95,45 %A=95,45 %S=89,47 %D=100 TH=34,2ms TS=151ms TD=13,2ms TA=19,2ms	%H=100 %A=100 %S=100 %D=100 TH=17ms TS=27,2ms TD=15ms TA=17ms	%H=100 %A=90,47 %S=100 %D=100 TH=347ms TS=249ms TD=120ms TA=99ms
Nodo 3	%H=100 %A=100 %S=100 %D=100 TH=12,4ms TS=15,7ms TD=7,4ms TA=20,2ms	%H=100 %A=100 %S=100 %D=100 TH=70,3ms TS=64,1ms TD=33,6ms TA=79,1ms	**********	%H=100 %A=100 %D=80 TH=29,3ms TD=213ms TA=180ms	%H=88,57 %A=84,61 %D=76,92 TH=62,3 TD=257ms TA=203ms
Nodo 4	%H=100 %A=100 %S=100 %D=100 TH=44,6ms TS=47,5ms TD=83,1ms TA=53,2ms	%H=100 %A=100 %S=100 %D=100 TH=57,5ms TS=85,3ms TD=108,8ms TA=**90ms**	%H=92,3 %A=91,6 %D=100 TH=67,5 TD=96,2ms TA=144,7ms	**********	%H=92,3 %A=91,6 %S=100 %D=80 TH=116ms TS=299ms TD=281ms TA=235ms
Nodo 5	%H=93,103 %A=90,9 %S=100 %D=88,88 TH=12,3ms TS=316ms TD=106ms TA=186ms	%H=95,23 %A=100 %S=93,33 %D=100 TH=39,8ms TS=51,1ms TD=70ms TA=93ms	%H=77,77 %A=87,5 %D=88,88 TH=29,5ms TD=181ms TA=182ms	%H=100 %A=100 %S=100 %D=100 TH=17,5ms TS=27,6ms TD=5,1ms TA=26,1ms	**********

Tabla 34. Experimento dinámico 11 Tam.Paquete=128B, tasaHello=2sec, tasaStatus=2sec, tasaData=4sec. Todos envían a todos.

Creación de una red mesh

```
#!/bin/bash
echo "Lanzando script de startup para poner meshpoint
sudo iw dev wlan0 interface add mesh0 type mp
sudo iw dev mesh0 set channel 1
sudo ifconfig wlan0 down
sudo ifconfig mesh0 192.168.100.3
sudo ifconfig mesh0 netmask 255.255.255.0
sudo iw dev mesh0 mesh join pi_mesh
sudo iw mesh0 set mesh_param mesh_fwding=0
sudo iw mesh0 set mesh_rssi_threshold=77
```

Ilustración 31. Script que añade al nodo 3 a la red de tipo mesh.

Este script debía ser lanzado automáticamente al inicio del sistema. Con el intentaba que únicamente viese como vecinos a aquellos nodos que tuviesen una calidad de señal mejor que -77, obviando aquellos que no cumpliesen con ello.

Para arrancar el script al inicio del programa tuve que ejecutar la siguiente secuencia de comandos:

```
sudo cp meshScript.sh /etc/init.d
sudo chmod +x /etc/init.d/meshScript.sh
sudo update-rc.d meshScript.sh defaults
```

Al ver que no se obtuvieron los resultados obtenidos, tocó remover el script mediante el uso del comando

```
sudo update-rc.d -f meshScript.sh remove
```