Universidad de la República
Facultad de Ingeniería

Informe Final

Bruno Guguich Perdomo
Diego López Colombo
Federico Avas Bergeret

Tutor: Eduardo Cota

Abril 2010

Contenido

Índice de Figuras

Abstract

Existen en la actualidad numerosos grupos trabajando con el fin de estandarizar las comunicaciones inter-vehicularse (VANET – *Vehicular Ad-Hoc Network*). Estas propuestas requieren cambios en el hardware y protocolos de los sistemas inalámbricos tradicionales para dar cumplimiento a los estrictos requisitos de latencia y soportar los rápidos cambios de topología que caracterizan a este tipo de redes. Un estudio del arte de estas redes es presentado en el capítulo 3.

En este trabajo se presenta una implementación de un sistema de comunicación inter vehicular que utiliza la inyección de paquetes en dispositivos 802.11 comerciales configuradas en modo monitor. Se describe el funcionamiento del sistema y presentan los resultados obtenidos en las pruebas realizadas sobre el mismo. Éstas últimas demuestran que es posible realizar implementaciones de bajo costo utilizando equipos comerciales y obteniendo resultados muy aceptables.

El prototipo Vo!zila brinda servicios de comunicación de datagramas a bajo nivel, tablas de ubicación y servicios básicos de propagación de paquetes. Se trata de una base sobre la cual se puede extender el desarrollo, implementando servicios de más alto nivel y orientados a aplicaciones particulares.

1. Introducción

El auge de las telecomunicaciones y la búsqueda de la conectividad, está llevando a que hoy en día se haga hincapié en la interconexión de todos los elementos de la vida diaria. Originalmente se realizó esta conexión por medios cableados, los que están siendo sustituidos y extendidos cada vez más por medios inalámbricos. Ahora que todo es “wireless”, el siguiente desafío es la movilidad.

Las tecnologías celulares están avanzando rápidamente en este campo pero con la necesidad de un *backbone* que soporte la red inalámbrica. En virtud de reducir costos, lograr independencia y mayor simplicidad es que se está trabajando mucho en soluciones tipo ad-hoc, donde la comunicación entre los distintos nodos se puede realizar sin la necesidad de un nodo central que oficie de controlador.

En este contexto resulta natural extender las comunicaciones inalámbricas a vehículos de todo tipo, como un siguiente paso en la búsqueda de movilidad. De esta manera se abre un nuevo capítulo en el mundo de las telecomunicaciones, dando lugar a una infinidad de nuevas aplicaciones en áreas muy diversas como ser control del tráfico, seguridad, diagnostico remoto de fallas, operación de vehículos de carga en espacios reducidos, etc.

Esto implica un replanteo en los mecanismos utilizados al establecer la red ya que en el mundo vehicular aparecen nuevas restricciones. Las altas velocidades relativas entre los distintos nodos implican que el tiempo disponible para la comunicación entre ellos es muy bajo. Además, las constantes variaciones en la topología generan situaciones que no han sido contempladas en el desarrollo de las tecnologías inalámbricas actualmente en uso.

Comenzamos el trabajo estudiando distintas tecnologías de radio utilizadas para la comunicación entre dispositivos móviles. En particular se desarrolló el estudio para APRS, STDMA, Bluetooth y 802.11. Cada una de ellas fue diseñada con un propósito diferente y por lo tanto encontramos grandes diferencias. Culminamos la sección mencionada anteriormente con un repaso de 802.11 analizando las distintas características de la tecnología y evaluando por qué parece ser el principal candidato para este fin.

En el capítulo siguiente se hace un relevamiento de las actuales tendencias en comunicación inter-vehicular desarrollando principalmente los dos proyectos con más fuerza en los últimos años como son Car-2Car y GeoNet en Europa y el desarrollo de WAVE en Estados Unidos. Se detallan los datos sobre la conformación e interdependencia de varios grupos así como también detalles de sus implementaciones. Finalizamos el capítulo con un extenso informe sobre los protocolos de ruteo propuestos observando ventajas y desventajas de cada uno.

Inmediatamente después se desarrollan las experiencias obtenidas en el inicio del proyecto cuando la aproximación a la resolución del problema se basaba en las comunicaciones en modo ad-hoc. Diferentes carencias de performance y otros inconvenientes explicados en ésta sección nos llevaron a cambiar la página y pasar un nivel inferior trabajando directamente con la capa MAC e inyectando paquetes directamente a la interfaz inalámbrica.

En final de este documento se realiza la descripción de nuestro prototipo, las conclusiones de las experiencias prácticas y se plantean posibles trabajos futuros. Además see anexa la especificación detallada del código realizado.

2. Informe de tecnologías

2.1. APRS, STDMA y Bluetooth

Antes de empezar con el desarrollo del prototipo realizamos el estudio de las tecnologías existentes que pensamos podrían aportar ideas interesantes. Buscamos tecnologías inalámbricas en uso, que tuvieran la particularidad de funcionar sin la necesidad de infraestructura fija que hiciera las veces de controlador. Dentro de éstas encontramos APRS (*Automatic Packet Reporting System*), STDMA (*Self-Organising Time Division Multiple Access*) y las más recientes Bluetooth y 802.11.

Empezamos el estudio con tecnologías de mayor antigüedad, que hacen uso de las bandas de VHF y HF, como en el caso de APRS. Estas tecnologías tienen la particularidad de tener un gran alcance de radio (por la frecuencia en la que trabajan) pero también la desventaja de proveer bajo ancho de banda. Esta última propiedad, es la que hoy en día tratan de maximizar todos los sistemas actualmente en uso y constante desarrollo como lo son Bluetooth y 802.11.

Otro tema de gran importancia a la hora de las comunicaciones inalámbricas es el control de acceso al medio. Por tratarse siempre de canales de radio compartidos, cada uno de estos estándares resuelven este tema de maneras diferentes, con ventajas y desventajas en cada caso. Estudiamos estas tecnologías para encontrar las más apropiada a utilizar como base de nuestro prototipo. Con este estudio también intentamos incorporar a la base escogida en la medida de lo posible aspectos convenientes de las otras tecnologías.

En el ANEXO A, se encuentra el estudio detallado de APRS, STDMA y Bluetooth. No obstante, a continuación se describen brevemente las principales características y conclusiones del estudio realizado. Como se verá en la descripción del prototipo implementado, 802.11 es de gran relevancia y por tanto un estudio más detallado se presenta en 2.2.

APRS

El sistema APRS le introdujo al existente packet radio una serie de aplicaciones que lo volvieron muy popular entre los radioaficionados, así como para su uso oficial por parte de agencias gubernamentales. El mismo está implementado en las distintas bandas de radiocomunicaciones asignadas a radioaficionados en HF y VHF, donde combinando la gran propagación de estas bandas con las altas potencias de transmisión que se pueden manejar, se logra un gran alcance.

APRS es un muy buen ejemplo de redes Ad-Hoc de largo alcance, en donde es posible propagar información en forma periódica así como realizar comunicaciones punto-punto. Su forma de resolver el camino de propagación de mensajes a partir de los alias es una solución muy interesante para el enrutamiento de paquetes en redes dinámicas, sin la necesidad de la gestión externa para definir los caminos.

En su forma actual, APRS se utiliza para la comunicación entre vehículos envíando mensajes con información, posición, y todo tipo de contenido de interés general, con una gran área de cobertura. Estas redes APRS se caracterizan por tener pocos nodos y estar los mismos separados grandes distancias. El bajo ancho de banda provisto hace que esta solución sea un tanto limitada para aplicaciones de tiempo real, sobre todo cuando se tienen muchos vehículos participando en alcance de radio, pudiendo llegar rápidamente a la saturación del servicio. Mientras tanto, su método de propagación por alias fue tenido en cuenta para nuestro prototipo.

STDMA

El sistema “Self-Organising Time Division Multiple Access” (STDMA) plantea una forma de acceso al medio compartido usando división en el tiempo y sin necesidad de un nodo central de coordinación. De ahí su nombre “Self-Organising” que hace referencia a la capacidad de los nodos de auto gestionar el acceso al medio, coordinando con el resto de la red. STDMA está actualmente implementado en sistemas de radio de VHF como el AIS (uso marítimo) y VDL modo 4 (uso aeronáutico).

A pesar de la gran funcionalidad de este sistema, no parece adecuado para su uso masivo en vehículos terrestres. Sólo 250 vehículos resulta bastante limitado tomando en cuenta la gran cobertura del VHF. Otra limitación en la ciudad se tiene en la recepción de las señales GPS (necesarios para la sincronización de los nodos), que pueden llegar a bloquearse totalmente por períodos muy largos (edificios altos, túneles, estacionamientos, etc).

Bluetooth

Bluetooth tiene como principales objetivos posibilitar la transmisión de voz y datos entre diferentes dispositivos móviles o fijos prescindiendo de cables, mediante un enlace por radiofrecuencia globalmente libre (2,4 GHz). Ofreciendo además la posibilidad de crear pequeñas redes inalámbricas de tipo ad hoc, en la cual los dispositivos pueden establecer por si solos la red.

Esta tecnología está especialmente diseñada para equipos portables lo que implica dispositivos pequeños, de bajo consumo y precio, aspectos que pueden resultar críticos según la aplicación.

La capacidad de los dispositivos Bluetooth de establecer redes de tipo ad-hoc es imprescindible a la hora de pensar en redes entre dispositivos móviles, donde tal vez, no se disponga de infraestructura fija para gestionar la red. Resulta interesante la manera en que bluetooth resuelve este problema asignando un maestro que organiza la red, oficiando de alguna manera de access point, y cuyo rol puede ser tomado por cualquier unidad en la red ad-hoc formada (piconet). Puede asignarse este rol al dispositivo que tenga más recursos por ejemplo, o al que tenga más información que desee difundir.

El alcance puede llegar a ser de 100 metros en el mejor de los casos. Éste es el de los dispositivos clase 1 aunque son los clase 3, con un alcance máximo de 10 metros, los que se han extendido masivamente. Esto es debido a que el espíritu de la tecnología y para lo cual fue diseñada era, en un principio, sustituir el cableado entre periféricos, lo que implica enlaces inalámbricos de corto alcance. Para redes vehiculares, el alcance es un factor determinante dado que 100 metros de alcance en el mejor de los casos, según el desplazamiento relativo de los vehículos, puede implicar tiempos transmisión muy bajos.

Analizando los tiempos de conexión, los cuales pueden tener un promedio de 5 a 7 segundos aproximadamente, está claro que bluetooth no es un buen candidato para establecer redes en que la topología cambie rápidamente, como es el caso de los vehículos en movimiento. Esto puede afectar el alcance de la aplicación final imponiendo grandes limitaciones. Por otro lado, debe tenerse presente la capacidad de las redes bluetooth de conformar scatternets como una forma de extender el alcance de la red si se dispone de una alta densidad de nodos, como es el caso por ejemplo de zonas céntricas.

2.2. 802.11

El estándar 802.11 fue ratificado en 1997 y desde ese entonces las redes locales inalámbricas (WLAN – *Wireless Local Area Network*) han aumentado exponencialmente y ya son parte de nuestra vida cotidiana. En facultad, el trabajo, al ir a un café para chequear el correo en la notebook e incluso en nuestras casas. Ya no son sólo las notebook los dispositivos que utilizan las WLAN sino que también se han agregado los teléfonos móviles y PDAs.

Las WLANs nos brindan una manera sencilla y libre de cables para interconectarnos con redes fijas corporativas, personales y a Internet. Inicialmente su desventaja en comparación con las redes fijas tradicionales era la velocidad, pero su gran practicidad e inserción al mercado han provocado que el I+D en este sector vaya cada día más allá.

Las primeras redes inalámbricas como Aloha, ARDIS y Ricochet proveían velocidades de menos de 1 Mbps mientras que el 802.11 de 1997 alcanzaba los 2 Mbps. Poco tiempo después surge 802.11b en el año 1999 aumentando la apuesta, ésta vez a 11 Mbps luego se desarrollan los estándares 802.11a y 802.11g los cuales son una fuerte competencia para las LANs tradicionales con tasas de 54 Mbps y terminando actualmente en el implementaciones del borrador para la versión *n* capaz de alcanzar 100 Mbps de manera estable.

En los comienzos, varias empresas encontraron atractiva esta tecnología y como resultado los proveedores pudieron aumentar los volúmenes de fabricación reduciendo de esta manera los costos. Por ende más empresas y hogares se acercaron a esta tecnología a precios razonables.

Este documento no pretende ser una guía sobre WLANs ni mucho menos, simplemente lo consideraremos como un repaso inicial seguido de un pequeño informe que orienta el uso de 802.11 para redes de transporte inteligentes como son las VANETs haciendo una mención al último estándar en desarrollo 802.11p y WAVE (*Wireless Access in Vehicular Environment*) enfocado claramente para redes tipo ad hoc en donde la gran movilidad de los vehículos y la poca utilidad de los access points agregan muchísimas dificultades a este tipo de redes.

2.2.1. Aspectos Generales

En esta sección se describen los principios generales de las WLAN, para profundizar más adelante en las capas MAC y PHY ya que son las que sufren mayores modificaciones al adaptar esta tecnología a redes vehiculares.

Las WLAN resultan ser bastante flexibles a la hora del diseño. Tres tipos básicos de topologías se pueden implementar:

- Independent Basic Service Sets (IBSSs)
- Basic Service Sets (BSSs)
- Extended Service Sets (ESSs)

En donde se entiende *service set* como un grupo lógico de dispositivos. Las WLAN proveen acceso a red por medio de broadcast en una portadora de radio frecuencia. Las estaciones receptoras filtran las distintas transmisiones usando el SSID (*Service Set ID*).

Los tres grupos se diferencian básicamente por su forma de acceder a la red y de comunicarse entre distintas estaciones. En el caso de una WLAN BSS, todos los clientes se comunican a través de un access point en todo momento. Por otro lado, varias infraestructuras BSSs interconectadas forman un ESS.

Finalmente, los BSSs independientes o Ad Hoc se crean cuando los clientes individuales se conectan entre sí sin la intervención de un AP. En el uso cotidiano, hasta hace poco tiempo este tipo de conexiones usualmente se realizaban para intercambios puntuales de información y los enlaces eran de corta duración entre unos pocos clientes. Actualmente con el fervor de las redes mesh y las VANETs, éste tipo de WLANs se está haciendo más frecuente y revolucionando la industria ya que lo que el cliente quiere es aún más movilidad.

2.2.2. Mecanismos de Acceso al Medio

En 802.11, el acceso al medio se regula con un sistema similar a Ethernet. Este sistema: CSMA/CA (*carrier sense multiple access with collision avoidance*) es un mecanismo del tipo “escucho antes de transmitir”. En Ethernet, las colisiones pueden ser detectadas fácilmente ya que dos estaciones (STA) transmitiendo al mismo tiempo elevan el nivel de señal en el cableado indicando a las STA que hubo una colisión. Como 802.11 se implementa con un solo transceptor, no se tiene la posiblidad de escuchar mientras se transmite. Por lo tanto, se debe evitar la colisión en vez de detectarla.

Como analogía se puede comparar CSMA/CD a una conferencia telefónica, cada participante espera a que nadie esté hablando para hablar. Si dos o más hablan al mismo tiempo, interrumpen en el momento que detectan la colisión y lo intentan nuevamente mas tarde. En este sentido CSMA/CA es más ordenado por lo siguiente:

Antes de hablar, cada participante debe indicar por cuánto tiempo va a hablar, para darle una idea a los demás de cuánto tiempo deben esperar antes de intervenir. Nadie puede hablar hasta que haya transcurrido el tiempo mencionado anteriormente. Los participantes no saben si están siendo escuchados hasta que reciben confirmación de la contraparte. Si existe una colisión, esta no puede ser detectada hasta que pasa el tiempo de espera de la confirmación. En caso de colisión los participantes esperan un tiempo aleatorio antes de volver a transmitir.

A continuación se describen algunos elementos claves para comprender mejor el sistema de acceso al medio de 802.11.

Carrier sense

Antes de transmitir una STA debe censar el medio para ver si está en uso. Esto se realiza con dos métodos: chequear en la capa física si la portadora está presente y usar una función virtual de censado de portadora llamada NAV (*Network Allocation Vector*).

Puede ocurrir que a pesar de que a nivel físico el medio esté libre, siga reservado por medio de NAV por otra STA. El NAV es un temporizador (*timer*) que es actualizado por las tramas transmitidas. La trama 802.11 contiene un campo de duración, y éste valor indica un tiempo suficientemente largo como para incluir la confirmación del mensaje enviado. Por lo que cuando una estación transmite, las demás actualizan el timer y se abstienen de transmitir hasta que este haya decrementado a 0. Este timer se actualiza solamente si el valor escuchado es mayor al guardado.

DCF

La función de coordinación de distribución o DCF por sus siglas en inglés es el mecanismo designado como obligatorio por la IEEE para el acceso al medio. En este modo de operación, una STA debe esperar un tiempo específico luego de que el medio está libre. Este tiempo es denominado DIFS (DCF interframe space) y una vez que expira, el medio está disponible para transmitir. Es altamente probable que dos STA quieran transmitir inmediatamente luego de que otra STA haya finalizado su transmisión ya que han estado sensando el canal ocupado por cierto tiempo. Esto ocasionaría colisiones muy frecuentes y como no podemos detectarlas, ésta colisión no sería detectada hasta que expire el tiempo de espera del ACK. Por eso se intenta evitar las colisiones, para ello el algoritmo de *random backoff* elige un numero aleatorio para la ventana de contención (CW). Este valor está comprendido entre 0 y CWmax (valor determinado en el estándar) y refiere a la cantidad de *timeslots* que la STA tiene que esperar para poder transmitir.

El siguiente diagrama de flujo puede ayudar a entender mejor el mecanismo:

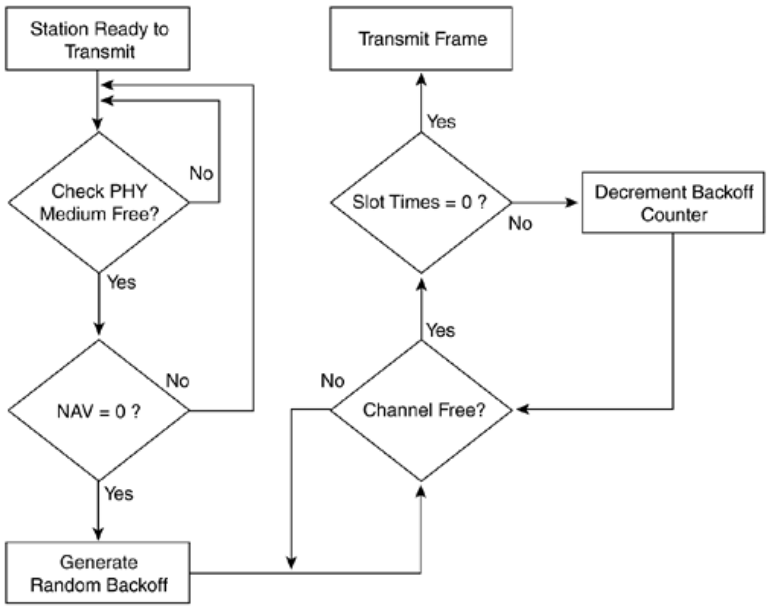

Figura 1 - Diagrama de flujo de DCF

El Reconocimiento (ACK)

Como mencionamos antes, la STA transmisora debe recibir un ACK de la receptora para considerar la transferencia exitosa. En principio se podría pensar que este reconocimiento pueda ser demorado porque el medio se encuentra ocupado. Por su importancia, esta trama recibe un tratamiento especial. Se permite que eviten el proceso de RB y esperaran un tiempo corto luego de que se indica que es necesaria su transmisión. El intervalo mencionado es conocido como SIFS (Short InterFrame Space), el cual es más corto que el DIFS por dos timeslots. Esto garantiza que la STA receptora tiene las mejores posibilidades de acceder al medio antes que cualquier otra STA.

RTS/CTS y el problema del nodo escondido

El conocido problema del nodo escondido surge cuando dos nodos que no tienen alcance de radio entre sí, interfieren la recepción de un tercero (en alcance de radio de ambos) cuando transmiten en simultáneo. Al no estar en alcance de radio, ambos nodos sensan el canal como libre al momento de transmitir. Sin embargo, en el tercer nodo sí están llegando las dos señales cuya superposición hace imposible la decodificación.

Se implementa entonces el sistema de RTS (Request To Send) y CTS (Clear To Send). El sistema funciona de la siguiente manera: en el caso anterior, antes de transmitir el primer nodo reserva el medio enviando un RTS el cual contiene la duración del mensaje a transmitir y por ende actualiza el NAV de todas las STA en rango. Luego el AP confirma (o no) esta reserva con un mensaje CTS que también contiene la duración por lo que esta vez el nodo escondido, que sí está en alcance del AP, actualiza su NAV y la colisión se evita. La trama RTS pasa por el proceso DCF como cualquier otra. En cambio, al igual que el ACK, la trama CTS evita este sistema y transmite luego del SIFS.

Fragmentación de tramas

La fragmentación de tramas es una función de la subcapa MAC diseñada para aumentar la eficiencia y confiabilidad de transmisión a través del medio inalámbrico. La idea detrás de esta función es que fragmentos más pequeños tienen mayor probabilidad de ser enviados de manera exitosa. Además, como cada fragmento recibe un ACK, en caso de error sólo se reenvía ese último fragmento, lo que aumenta el throughput efectivo del medio.

Vale la pena destacar que el administrador de red puede definir el tamaño del fragmento y que la fragmentación sólo ocurre para tramas unicast. Las tramas broadcast o multicast nunca son fragmentadas. Las tramas fragmentadas sólo realizan una iteración del mecanismo DCF.

También vale la pena mencionar otro de los mecanismos de acceso al medio, PCF (P*oint Coordination Function*) que no es usada mucho debido a que aumenta el overhead. A diferencia de DFC, en este método las STA no pueden acceder libremente al medio para enviar datos. Sólo lo podrán hacer cuando el PC (*Point Coordinator*) lo permita. El PC accede al medio también por DCF, pero un TS antes que las STA, esto le permite acceder al medio siempre primero y controlando qué estaciones transmiten. Se lo podría considerar como algo similar a un director de tránsito.

Asociación a un BSS

- Proceso de exploración

Cuando una STA está bajo la presencia de uno o más APs, generalmente envía un mensaje de prueba o sonda (*probe*) en todos los canales que tiene permitido usar. Este mensaje se envía a la menor tasa posible y contiene información sobre la STA como las tasas de datos que soporta y el SSID. Cuando el AP recibe el pedido de sondeo mencionado anteriormente y luego de que éste pasa exitosamente un chequeo, responde a este sondeo con un mensaje similar. Este último mensaje contiene un timestamp utilizado por el cliente para sincronizarse con el AP, el tiempo entre beacons, capacidades de capa MAC y PHY, el SSID, tasas de datos soportadas, etc. Después de haber detectado la trama anterior, el cliente es capaz de determinar la potencia de recepción que obtiene de la señal del AP y la asociación se resuelve con este valor pero en general se deja a cargo del fabricante esta decisión.

- Autenticación y Asociación

Básicamente existen dos métodos de autenticación que no se detallarán en esta etapa del proceso por no ser necesarios aún. Éstos son el de autenticación abierta o de clave compartida (*shared key*).

La asociación permite a un AP mapear un puerto lógico o identificador de asociación (AID) a una STA. Este proceso es iniciado por la STA con una trama de pedido de asociación. En la trama se dan parámetros relativos al modo de bajo consumo de potencia, resultado de la asociación, etc.
Los métodos de ahorro de energía no serán descriptos en este documento.

2.2.3. Formatos de las tramas MAC

Existen 3 categorías de tramas:
- Control, para el correcto intercambio de datos
- Administración, para facilitar la conectividad, autenticación y status.
- Datos

La trama de control es un valor de 2 bytes con 11 subcampos que se detallan en la figura. La descripción de los subcampos queda bastante clara con sus respectivos nombres y os posibles valores pueden tomar se encuentran en el estándar.

B0 B1	B2 B3	B4 B7	B8	B9	B10	B11	B12	B13	B14	B15
Protocol Version	Type	Subtype	To DS	From DS	More Frag	Retry	Pwr Mgt	More Data	Protected Frame	Order
Bits : 2	2	4	1	1	1	1	1	1	1	1

Figura 2 - Trama de control

Las tramas de control son: Power Save poll, RTS, CTS, ACK, Contention Free End (CF-End) y CF-End+CF Ack.

Las tramas de administración son:

- Beacon
- Probe Request
- Probe Response
- Authentication
- Deauthentication
- Association Request
- Association Response
- Reassociation Request
- Reassociation Response
- Disassociation
- Announcement traffic indication

Y se construyen con la siguiente estructura:

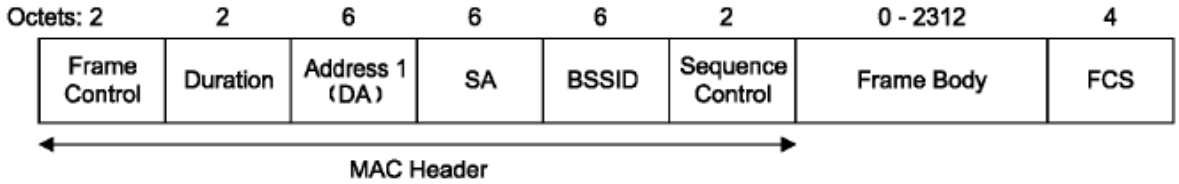

Figura 3 - Trama de administración

Las tramas de datos, al igual que las de administración son precedidas de un campo de control. Los posibles tipos son:

- Data
- Null data
- Data+CF-Ack
- Data+CF-Poll
- Data+CF-Ack+CF-Poll
- CF-Ack
- CF-Poll
- CF-Ack+CF-Poll

Las distintas combinaciones de tramas con CF son requeridas para la operación en el modo PCF mencionado anteriormente.

2.2.4. Capa física

Haremos una mención ahora de las distintas tecnologías para la capa física (PHY de ahora en más) que han ido evolucionando a lo largo del tiempo. Con distintas tecnologías de modulación se logran diversas tasas de transferencia a pesar de usar en la mayoría de los casos, la misma capa MAC. Entre otras cosas, la capa PHY se encarga de proveer una forma de transmitir las tramas que proporciona la capa MAC así como de sensar el medio para comprobar si está ocupado.

Hay que destacar que para 802.11 en las capas MAC y PHY nos encontramos con 2 subcapas. Para el caso de la capa 1, estas son: PLCP = Physical Layer Convergence Procedure y PMD = Physical Medium Dependant.

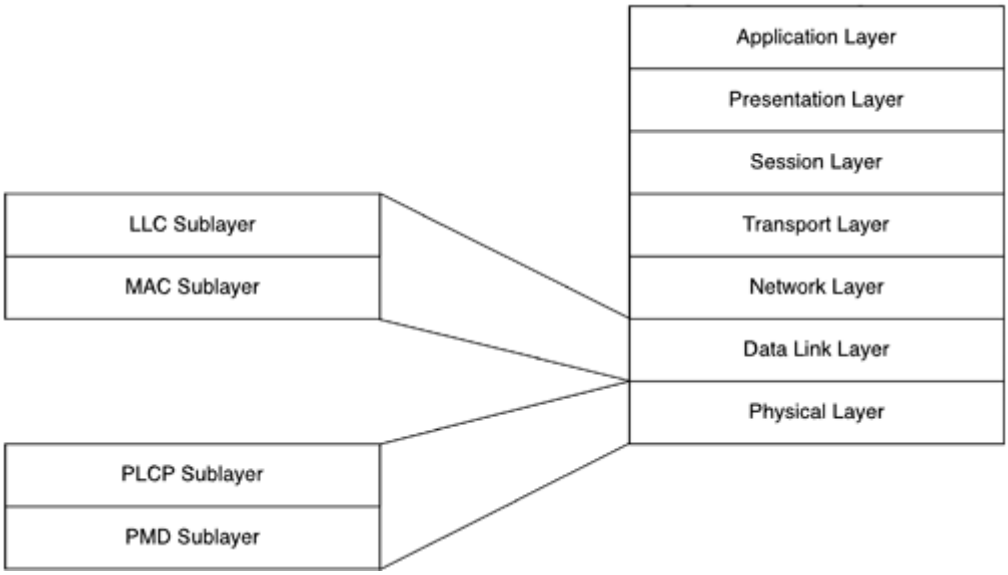

Figura 4 - Diagrama de sub-capas

La PLPC realiza el *handshake* entre la capa MAC y la subcapa PMD quien es la que físicamente transmite la trama. Provee funciones con las cuales la MAC pide el inicio de una transmisión a la capa PHY y con la cual esta última responde el fin del envío de trama. El funcionamiento de esta subcapa es sencillo y se puede describir con el siguiente diagrama:

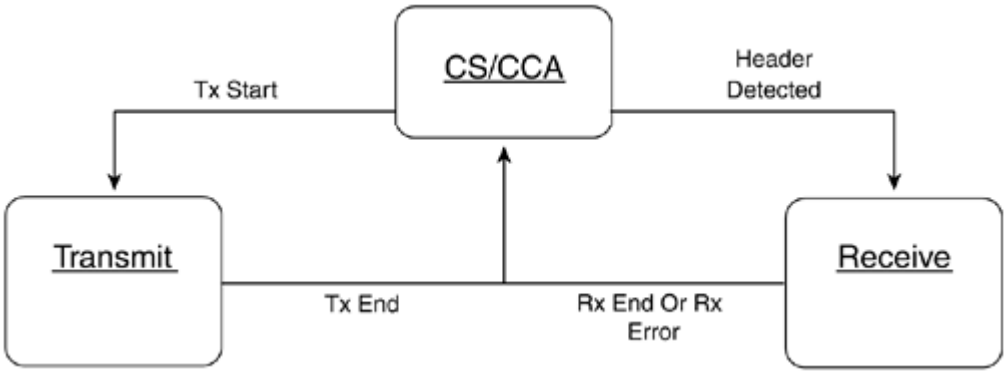

Figura 5 - Sub-capa PLPC

Para comprender mejor las distintas PMD de cada capa PHY tenemos que entender primero los distintos bloques con los que está construida.

Existe un bloque de "*Scrambling*" que está encargado de "blanquear" la señal, es decir mezcla el stream de 1s y 0s a transmitir de manera que no queden grandes cadenas de 0s o 1s continuas. En el receptor el mismo bloque realiza el proceso inverso. La mayoría de estos métodos pueden sincronizarse entre Rx y Tx por sí solos. El proceso de *scrambling* aumenta la eficiencia del uso del espectro al homogeneizar la densidad espectral de potencia de la señal.

Al igual que en cualquier sistema de transmisión digital, es necesario codificar los valores que serán enviados para permitir transmitir a altas velocidades en un canal ruidoso. Entre los codec más comunes se encuentra el código de convolución que es implementado fácilmente en HW con retardos y sumadores.

Para aumentar aún más la inmunidad frente al ruido, también existe un bloque de "*Interleaving*". Esta técnica protege contra las ráfagas de errores que caracterizan a los

sistemas inalámbricos. Al cambiar el orden en qu ese transmiten los bits, se evita que las ráfagas afecten a bits consecutivos del mensaje y de esta forma los mecanismos de corrección de errores resultan más efectivos.

2.2.5. Movilidad

Como mencionábamos al inicio de este documento, una vez que tenemos a los dispositivos conectados en forma inalámbrica, el paso siguiente es la movilidad. Un escenario donde tenemos a los dispositivos móviles asociados a nodos de infraestructura fija, implica que el móvil deba cambiar de Access Point a medida se va desplazando. Esto es muy similar a lo que sucede en las redes de telefonía celular y en 802.11 se denomina Roaming.

Es importante recalcar que para evitar loops en capa MAC, un móvil no puede estar al mismo tiempo asociado a más de un AP. Esto implica que el roaming en 802.11 es del tipo "*break before make"*, debiendo entonces liberar la conexión con el primer AP antes de poder pasar al segundo. En determinadas aplicaciones orientadas a conexión, o de tiempo real, esto puede implicar pérdida de datos, ya que durante la transición no hay como llegar al móvil, y al finalizar la misma, el móvil puede encontrarse en una subred diferente.

Descubrir el AP candidato para el cambio

Quien decide realizar el cambio de AP es el mismo dispositivo móvil y hay varios algoritmos que puede utilizar para determinar que la conexión con su actual AP se está degradando, o simplemente puede esperar a perder totalmente la conexión. Para que sea posible el roaming, debe existir otro AP con el mismo SSID perteneciente por tanto al mismo dominio de roaming, y el móvil debe descubrirlos.

Para descubrir los AP candidatos para el cambio, el móvil puede tomar dos caminos: "*preemtive AP discovery*" o "*roam-time AP discovery*".

En el primero, el móvil busca por AP dentro del domino de roaming incluso antes de perder la conexión con su AP actual. Este método tiene la ventaja que una vez que se tome la decisión de cambiar de AP (o se pierda la conexión con el primero), el móvil ya va a tener los datos de su nuevo AP y el tiempo invertido en realizar el cambio será minimizado. Como desventaja, tenemos el problema que mientras el móvil está buscando nuevos AP, se ve obligado a dejar la escucha del canal actual perdiendo entonces la posibilidad de recibir datos en ese momento.

En el segundo método, el móvil espera hasta perder la conexión con su AP para salir a buscar AP nuevos. Obviamente esto alarga más los tiempos de asociación con el nuevo AP, pero no se tiene el overhead de realizar la búsqueda mientras se está activamente en AP original.

A su vez, en cada caso la búsqueda de nuevos AP se puede hacer en forma activa o pasiva. En una búsqueda activa el móvil transmite en cada canal en búsqueda de la respuesta de un AP, mientras que en la pasiva simplemente escucha por los beacons de los mismos. Nuevamente encontramos ventajas y desventajas en cada caso, principalmente por el lado del tiempo que insume cada uno, así como el consumo energético que implica.

En una búsqueda pasiva se debe permanecer más tiempo en cada canal para permitir que llegue el beacon del AP, en vez de solicitarlo activamente. Esto igual tiene la ventaja de que al evitar la transmisión se ahorra energía, algo que puede tornarse crítico en ciertas aplicaciones.

El procedimiento ideal

El procedimiento más adecuado para realizar el roaming implicaría los siguientes pasos cuando el móvil pasa del AP1 al AP2:

1. AP1 detecta que el móvil no está recibiendo las tramas y comienza a almacenarlas
2. El móvil se asocia al AP2
3. El AP2 envía una trama por la red cableada indicando que el móvil ahora está asociado a este nuevo AP. Esta comunicación tiene 2 finalidades:
    a. AP2 envía la trama colocando la dirección MAC del móvil en el campo de originador del mensaje. Esto causará que los switches actualicen sus tablas ARP al recibir esta MAC en una nueva boca.
    b. AP1 recibe la trama y confirma entonces que el móvil ha dejado de estar bajo su dominio, enviándole todos los datos almacenados a AP2 para que lleguen al móvil. La forma en que AP2 le indica a AP1 sobre la aparición del móvil no está definida en la norma 802.11 y se utilizan por tanto formatos específicos de cada fabricante.

2.2.6. Evolución Histórica

Para las primeras versiones de 802.11 se implementaron dos métodos para la capa PHY: *Frequency Hopping Spread Spectrum* (FHSS) y *Direct Sequence Spread Spectrum* (DSSS) ambas en la banda de 2.4 GHz (2.402 a 2.480 GHz). FHSS separa esta banda en 79 canales no solapados de 1 MHz sobre los cuales transmisor y receptor saltan (*hopping*) transmitiendo los símbolos a 1 MHz con patrones de saltos predefinidos. La modulación para el caso anterior es GFSK que es nada más que una señal FSK que pasa luego por un filtro Gaussiano haciendo las transiciones de frecuencia más suaves. En DS en cambio, se toma la señal original y se le aumenta el ancho de banda al hacer un XOR con una señal de mayor frecuencia (22 MHz) de esta manera por cada bit a transmitir, se obtiene una secuencia de 22 “chips” que dependen de la secuencia de expansión; en recepción se realiza la operación inversa y se obtiene el bit enviado originalmente. Para este caso se utiliza BPSK o QPSK para obtener tasas de 1 y 2 Mbps respectivamente, para simplificar el hardware se utiliza el método diferencial de PSK.

Sobre 1999, la IEEE publica el borrador del estándar de 802.11b, que simplemente optimiza la modulación de DS cambiando la codificación, esta vez utilizando CCK (Complementary Code Keying) obteniendo tasas de 11 Mbps pero manteniendo aún la compatibilidad con la versión anterior. Otros pequeños cambios en las estructuras de las tramas son implementados pero no son relevantes para este documento.

El sistema de modulación es muy similar al de la versión anterior, solo que en este caso el código de chipeo es complejo. Para el caso de 5.5 Mbps por ejemplo, se toman símbolos de 4 bits (b0.b1.b2.b3) con los dos últimos bits del símbolo se elige la secuencia de chipeo de 8 bits mencionada anteriormente y, con los dos primeros bits se selecciona la rotación de fase de DQPSK.

En paralelo con la norma anterior (802.11b) surge 802.11a. Este sistema es completamente distinto a los anteriores ya que utiliza OFDM y canales de 20 MHz logrando tasas de hasta 54 Mbps en la banda libre de los 5 GHz y dejando un importante precedente para la emergente versión n que se busca estandarizar en estos días.

OFDM se utiliza para disminuir la Interferencia Inter-Simbolica (ISI por sus siglas en ingles) ya que secciona el canal en N subcanales de menor ancho de banda y transmitiendo símbolos a una tasa mucho menor por cada uno de los subcanales. De esta manera en detección

disminuye el error ocasionado por ISI al aumentar la ventana de tiempo en que está disponible el símbolo; esto junto a una modulación en 64 QAM logra los 54 Mbps.

Varios años más tarde, en el 2003 para ser exactos, se desarrolla 802.11g estándar ampliamente usado hoy en día que simplemente trae OFDM a la banda ISM (2.4 GHz) logrando las mismas tasas que 802.11a pero manteniendo la compatibilidad hacia atrás con b y la versión original.

2.2.7. Radiotap

Introducción

Al trabajar con las placas inalámbricas en modo monitor cobra gran importancia un encabezado "transparente" en la operación normal del WiFi. Su aplicación así como particularidades de uso ameritan un capítulo independiente que presentamos a continuación.

Cometido del encabezado Radiotap

Radiotap surge como una alternativa para enviar y recibir información del driver de la placa 802.11 directamente en los paquetes traficados por la misma. Su gran practicidad así como las posibilidades de desarrollo lo han convertido en el estándar de facto para realizar esta tarea. Esta información es incorporada al paquete como un encabezado más, en un nivel inferior al encabezado definido por la norma 802.11.

El encabezado radiotap fue diseñado con el objetivo de lograr un formato de captura independiente del hardware y extensible que pudiera soportar virtualmente todos los protocolos de radio 802.11.

Este encabezado no viaja en el aire sino que el driver de la tarjeta inalámbrica lo retira de las tramas salientes previo al envío por RF y lo agrega a las tramas entrantes. Es por esto que para una misma trama, el encabezado radiotap previo al envío es diferente del recibido en la otra punta.

En la recepción, a través de este encabezado se puede obtener información adicional de capa física que no es considerada por el encabezado 802.11. En particular, el encabezado radiotap contiene información de capa física que resulta de utilidad en el análisis de tráfico wifi y que las tarjetas inalámbricas usualmente no pasan hacia arriba en el stack, como ser: el tiempo de arribo de la trama, el bitrate, el canal, potencia, nivel de ruido, entre otros.

En la transmisión, este encabezado contiene información para configurar los parámetros físicos de la placa inalámbrica como lo son la frecuencia, potencia de transmisión, etc. De esta forma obtenemos un método sencillo y eficiente para configurar la placa inalámbrica en función de las necesidades particulares de cada paquete sin tener que recurrir a rutinas especiales que realicen esta tarea.

Radiotap esta soportado en mayor o menor medida por la mayoría de los sistemas operativos existentes, entre ellos Linux, donde a través de la biblioteca libpcap obtiene la información contenienda en el encabezado radiotap de las tramas entrantes.

Formato del encabezado

El formato de captura radiotap tiene un encabezado con la siguiente estructura (en lenguaje C):

```
struct ieee80211_radiotap_header {
        u_int8_t                it_version;             /* set to 0 */
        u_int8_t                it_pad;
        u_int16_t               it_len;                 /* entire length */
        u_int32_t               it_present;             /* fields present */
} __packed;
```

it_version es la versión en uso que actualmente es la 0
it_pad en desuso actualmente, necesario para la alineación natural de it_len
it_len indica la longitud total de los datos radiotap incluyendo el encabezado
it_present es un bitmask que indica los campos presentes

El campo it_present consta de un preámbulo estándar de 32 bits donde cada bit del 0 al 30 indica qué campos están presentes luego del preámbulo. El bit 31 permite agregar otro preámbulo opcional a continuación del preámbulo estándar con el objetivo de poder extender la cantidad de campos en el futuro. Cada preámbulo extra tiene reservado el bit 31 para poder extender el it_present en 32 bytes cada vez.

Un driver que implementa radiotap normalmente define una estructura raditap_header al comienzo de la trama recibida, seguida por los campos correspondientes y en el orden adecuado. El driver también define una macro para setear los bits del encabezado que indican qué campos han sido llenados. Inmediatamente después de recibir una trama, el driver completa los campos con la información disponible.

Aunque la combinación de campos puede variar, estos deben estar en su correspondiente orden para ser interpretados adecuadamente. El contenido de cada campo debe ir escrito en Little-endian y el largo de cada campo está establecido con anterioridad y es fijo.

El contendido de los campos debe estar naturalmente alineado, es decir, los campos de 16, 32 y 64 bits de largo, deben comenzar en sus respectivas fronteras de 16, 32 y 64 bits. Para lograrlo, es necesario rellenar los campos cuyo contenido no tenga el largo adecuado, lo que se conoce como padding.

Por ejemplo, supongamos que se quiere construir un encabezado con los siguientes campos:

```
struct rtapdata {
    uint8_t  antsignal;
    uint16_t tx_attenuation;
    uint8_t  flags;
    uint16_t rx_flags;
} __attribute__ ((packed));
```

Esta disposición de los campos provocaría que tx_attenuation y rx_flags quedaran mal alineados con su frontera natural de 16 bits. Para esto se rellenan los campos que los preceden de la siguiente manera:

```
struct rtapdata {
    uint8_t  antsignal;
    uint8_t  pad_for_tx_attentuation; // <-- added
    uint16_t tx_attenuation;
    uint8_t  flags;
    uint8_t  pad_for_rx_flags;        // <-- added
    uint16_t rx_flags;
} __attribute__ ((packed));
```

Los campos definidos hasta el momento son:

Bit number	Field	Structure	Required Alignment	Unit
0	TSFT	u64 mactime	8	Ms
1	Flags	u8 flags		Bitmap
2	Rate	u8		500 kbps
3	Channel	u16 frequency, u16 flags	2	MHz, bitmap
4	FHSS	u8 hop set, u8 hop pattern		??
5	Antenna signal	u8		dBm
6	Antenna noise	u8		dBm
7	Lock quality	u16	2	unitless
8	TX attenuation	u16	2	unitless
9	dB TX attenuation	u16	2	dB
10	dBm TX power	s8	2	dBm
11	Antenna	u8		Antenna index
12	dB antenna signal	u8		dB
13	dB antenna noise	u8		dB
14	RX flags	u16	2	Bitmap
31	reserved: another bitmap follows			
31+32*n	reserved: another bitmap follows			

Tabla 1 - Campos del encabezado Radiotap

TSFT: timestamp de 64 bits conteniendo un entero sin signo en microsegundos que indica el momento en que llega el primer bit de la trama a la capa mac.

Flags: 8 bits de banderas especificando propiedades de las tramas.

Rate: data rate en uso en unidades de 500 kbps.

Channel: consta de dos valores de 16 bits, el primero contiene la frecuencia a la que son transmitidas o recibidas las tramas mientras que el segundo es un bitmap que indica propiedades del canal en uso.

FHSS: consta de dos valores de 8 bits, está presente solo en radios que implementan frequency hopping, el primer byte contiene el hop set mientras que el segundo el patrón en uso.

dBm_antsignal: potencia de la señal de RF en la antena en dBm.

dBm_antnoise: potencia del ruido de RF en la antena en dBm.

Lock_quality: mide la calidad del Barker Code Lock, a veces llamada “Calidad de Señal” en algunas hojas de datos.

TX_attenuation: expresa la potencia de transmisión sin unidades tomando como referencia la máxima potencia de transmisión establecida de fábrica, 0 indica máxima potencia de transmisión.

dB_TX_attenuation: expresa la potencia de transmisión en dB tomando como referencia la máxima potencia de transmisión establecida de fábrica, 0 indica máxima potencia de transmisión.

dBm_TX_power: nivel de potencia absoluto medido en el puerto de la antena y expresado en dBm.

Antenna: para radios que soportan varias antenas específicas la antena en uso, la primera antena es la 0.

dB_antsignal: indica la potencia de señal de RF en dB respecto a una referencia arbitraria fija.

dB_antnoise: incida la potencia de ruido de RF en dB respecto a una referencia arbitraria fija.

EXT: bit reservado para una futura extensión de la estructura radiotap, seteando este bit se obtienen otros 32 bits de encabezado, pudiendo extender el encabezado en múltiplos de 32 bits seteando el último bit de cada bitmap.

2.2.8. Conclusiones

En el capítulo de 802.11 explicamos los principios de esta tecnología en donde podemos apreciar la originalidad que tuvo en sus inicios y cómo se ha ido moldeando mediante el agregado de funciones o mejoras en principios ya existentes, a las necesidades de los usuarios de tener cada vez más ancho de banda. A pesar de lo dicho anteriormente, también se observan las falencias de esta tecnología al intentar ir aún más lejos y acercarse a la comunicación vehicular.

802.11 fue originalmente pensado para dispositivos inalámbricos pero no en continuo movimiento como es el caso de un vehículo. Es por eso que la implementación de una VANET no sería posible con la tecnología existente, se requiere un cambio más drástico a los realizados anteriormente. Este cambio involucra principalmente un reestructura de toda la capa MAC de manera que ésta resuelva los conflictos que surgen en una situación de extrema movilidad.

3. Vehicular Ad-Hoc Networks (VANETs)

3.1. Introducción

Hace muchos años ya que los *Intelligent Transportation Systems* (ITS) han tomado un papel relevante en el contexto internacional. Tanto Estados Unidos como Japón y la Unión Europea están trabajando para estandarizar la arquitectura que establezca esta tecnología en los vehículos del futuro.

En el caso Europeo, desde el 6º Programa Marco las ITS han tenido una especial atención con una fuerte financiación por parte de éste a los grupos de trabajo. Hoy en día, con el 7º Programa Marco estos esfuerzos se siguieron incrementando y estamos cada vez más cerca de tener las primeras arquitecturas estandarizadas. La Unión Europea creó COMeSafety, un organismo dedicado a coordinar los diferentes grupos de trabajo, priorizando y realizando las gestiones necesarias con los organismos de estandarización y reguladores de comunicaciones. Tiene a CVIS como organismo de validación, quien impulsa la creación de una arquitectura abierta.

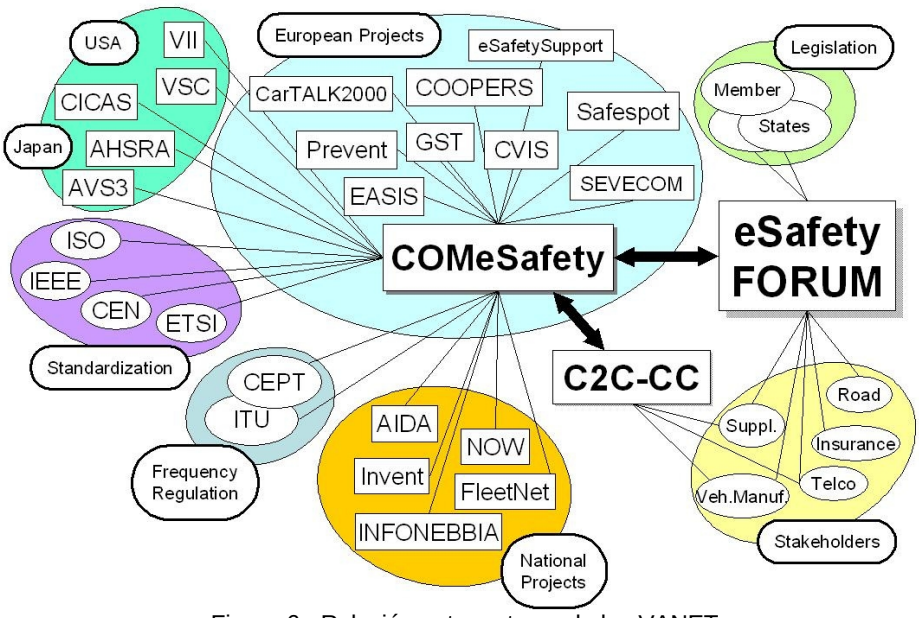

Figura 6 - Relación entre actores de las VANETs

Son muchos los grupos de trabajo que estudian el tema y cada uno intenta influenciar la estandarización hacia su lado. En este informe vamos a detallar los grupos de trabajo más activos y que al día de hoy han avanzado más, como lo son el Car-2-Car Consortium y el Proyecto GeoNet, ambos validados por CVIS. Otros grupos de trabajo interesantes son SafeSpot, que se centra en las aplicaciones de seguridad vehículo-vehículo, y Coopers, que se centra en la gestión eficiente del tráfico a través de la comunicación vehículo-infraestructura.

Todos estos grupos de trabajo vierten sus resultados a los organismos de validación y estandarización donde el objetivo es lograr la arquitectura completa para el ITS. Tanto ETSI, ISO como IEEE están trabajando en el tema.

En el caso de ISO, la arquitectura se llama CALM y corresponde al estándar ISO 21217 (TC204 WG16). En este caso, también se están tomando los resultados de Car-2-Car y GeoNet para la elaboración del estándar.

3.2. Car-2-Car

El *Car 2 Car Communication Consortium* (C2C-CC) es un grupo europeo formado por diversas entidades tanto públicas como privadas. Ha sido creado con el fin de estandarizar las interfaces y protocolos de comunicación inter-vehicular en Europa y el mundo.

Básicamente la tecnología intenta lograr una red ad hoc de vehículos y estaciones fijas capaces de interactuar entre sí de manera inteligente para lograr mayor seguridad y confort al conductor. Utilizando 802.11p, el C2C-CC intenta resolver la gran variedad de dificultades que surgen al intentar formar una red ad hoc en la cual la topología varía muy rápidamente, los vehículos se mueven a gran velocidad, y los tiempos de latencia para aplicaciones de seguridad deben ser muy bajos.

Los casos de uso tanto para aplicaciones de seguridad como para proveer información o entretenimiento, incluso para la optimización del flujo del tráfico en autopistas y ciudades son muy diversos. Es por eso que el consorcio cuenta con una gran cantidad de integrantes desde fabricantes de automóviles hasta organizaciones gubernamentales y universidades.

3.2.1. Descripción

El sistema está formado por 3 dominios: dentro del vehículo, ad hoc e infraestructuras.

Dentro del vehículo refiere a una red formada por una unidad de a bordo, OBU por sus siglas en inglés, y varias unidades de aplicación (AUs). Las AUs son dispositivos dedicados a una o un conjunto de aplicaciones que pueden estar integradas al vehículo o pueden ser removibles como un PDA o teléfono móvil.

El dominio ad hoc o VANET está compuesto por vehículos equipados con OBUs y unidades estacionarias a lo largo de las diferentes vías de tránsito (RSUs). Las OBUs están equipadas con al menos un dispositivo de comunicación inalámbrica y forman una red ad hoc móvil entre sí y con las RSUs. Todos los integrantes de la red ad hoc pueden comunicarse entre sí directamente con alcance de radio, o a través de otros entes por medio de protocolos de ruteo apropiados. El principal uso de las RSUs está orientado a la seguridad ya sea informando a los vehículos distintas situaciones como reportes de tráfico o simplemente extendiendo el alcance de la red ad hoc y son tratados como nodos estáticos de la red ad hoc.

También las OBUs se pueden asociar a nodos del dominio de infraestructura, esta vez a través de 802.11 a/b/g y de esta manera acceder a cualquier host en Internet. A modo de ejemplo, un nodo de este dominio podría ser un hot spot en una estación de carga de combustible. Este acceso a Internet se puede complementar con alguna tecnología celular como HSDPA o GPRS. En la Figura 7 se observa claramente la diferencia de dominios.

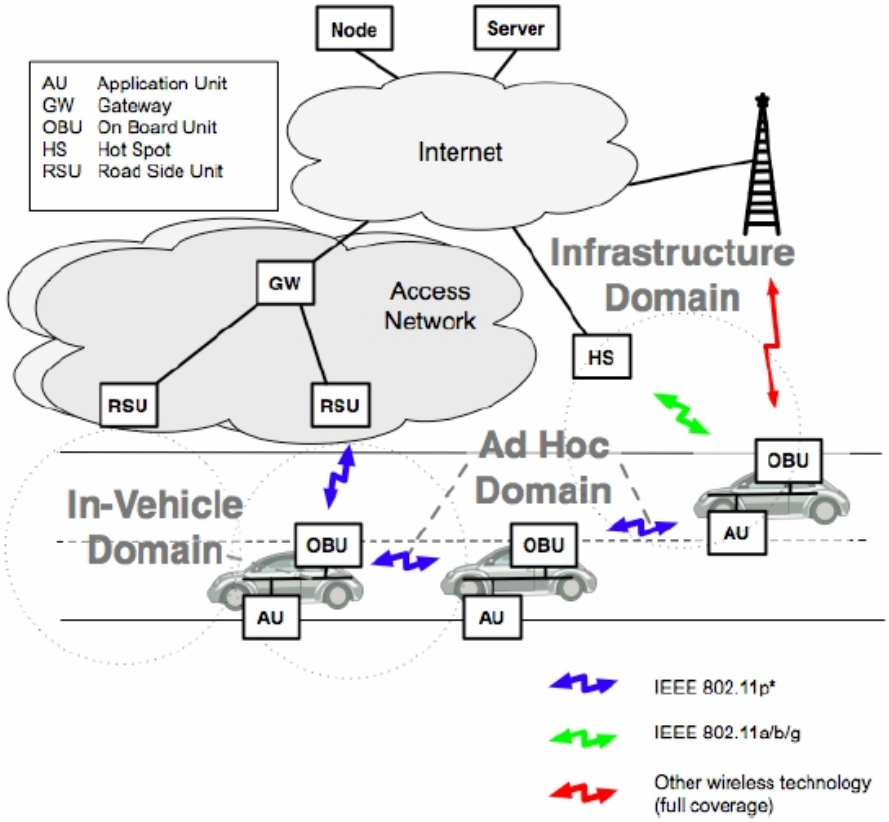

Figura 7 - Dominios de comunicación

3.2.2. Arquitectura de capas y protocolos relacionados

El C2C-CC distingue tres tipos básicos de tecnologías inalámbricas: IEEE 802.11p, tecnologías WLAN convencionales basadas en 80211. a/b/g/n y otra complementaria como GPRS o UMTS.

Encima de las respectivas capas MAC y física, la capa de red provee comunicación inalámbrica multi salto basada en direccionamiento geográfico y beaconing de los vehículos.

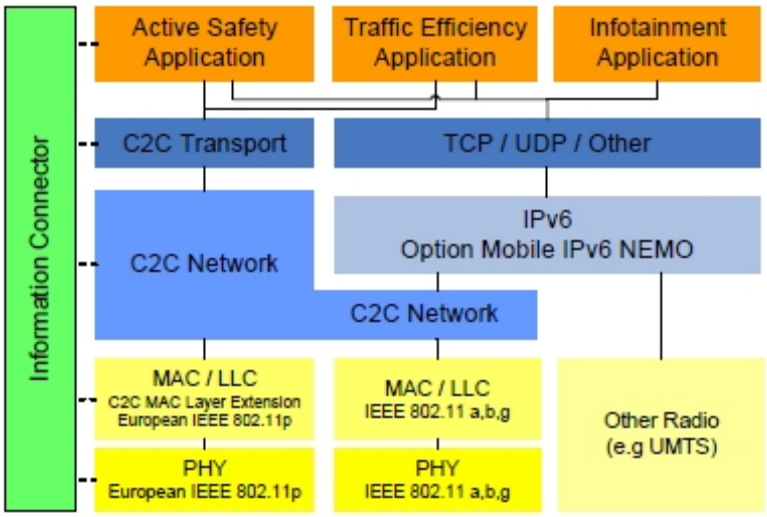

Figura 8 - Diagrama de capas de la arquitectura Car-2-Car

Como se puede observar en la figura 8 las aplicaciones no orientadas a seguridad utilizan el stack tradicional de protocolos con TCP/UDP sobre IPv6. Lo que hay que remarcar es que las aplicaciones de seguridad se comunican regularmente a través de las capas C2C de red y transporte, la capa física 802.11p y las extensiones de capa MAC IEEE1609.4.

Otro elemento interesante en la arquitectura es el Conector de Información (en verde, sobre la izquierda). Su principal tarea es proveer un mecanismo de intercambio de información entre capas.

En particular, la capa de red C2C se encarga de los protocolos de reparto de datos para aplicaciones VANET, ya que debido al limitado ancho de banda, que cada nodo haga un broadcast de cada paquete recibido no es nada eficiente. En la seccion 4.2 se detalla un estudio sobre la propagacion por inundación. En esta versión de capa de red, la información puede ser dirigida a un área puntual y recién al llegar a dicha área se distribuye la información entre los vehículos.

La capa MAC C2C está basada en el MAC IEEE 802.11 pero con simplificaciones en los servicios y algunas mejoras en la interacción entre capas. El algoritmo elegido es CSMA/CA. En particular, la capa MAC C2C define una única red ad hoc en donde todos los nodos que cumplan con el estándar son miembros a priori sin necesidad de asociación.

Para el problema de congestión se requieren características no incluidas en el estándar 802.11 como por ejemplo que dicha capa debe proveer información sobre la carga estimada del canal a las capas superiores. De acuerdo a ésta información las capas superiores deben adoptar estrategias para reducirla, como por ejemplo que la capa de aplicación decida si transmitir o no de acuerdo a la congestión existente.

3.2.3. Sistema de Radio

Como se mencionó con anterioridad, la capa física de este sistema considera el IEEE 802.11p como tecnología de radio. Este draft es derivado directamente del 802.11a pero adaptado a la comunicación vehicular removiendo, como mencionamos anteriormente, la asociación y autenticación.

Se han solicitado las siguientes bandas de frecuencia al ETSI (*European Telecomunications Standards Institute*):

- 10 MHz (5.885 a 5.895 GHz) para control y aplicaciones críticas a seguridad
- 10 MHz (5.895 a 5.905 GHz) para aplicaciones críticas a seguridad
- 3 bandas de 10 MHz (5.875 a 5.885 GHz y 5.905 a 5.925 GHz) para eficiencia de tráfico y seguridad no critica
- 2 bandas de 10 MHz (5.855 a 5.875 GHZ) para aplicaciones no relacionadas a seguridad

Por más información: *ETSI TR 102 492-1*

También se establecen máximos a la potencia de transmisión fijando este valor a 33dBm. El alcance deberá ser de entre 500 y 1000m en un salto teniendo línea de vista.

A su vez se enfatiza el control de dicha potencia de transmisión, pudiendo ser ajustable por paquete a pedido de las capas superiores. Este ajuste de potencia deberá ser dinámico y con un mínimo de 3dBm.

Para la capa MAC se decidió seguir los estándares (drafts) 802.11p y 1609.4 adoptando el algoritmo *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA). Encontramos en el curso de nuestra investigación, que este algoritmo presenta problemas al requerir cierta performance en la comunicación, en particular cuando se exigen tiempos de latencia relativamente bajos. Existen propuestas para el uso de STDMA al igual que en AIS en lugar de CSMA/CA para poder asegurar los tiempos requeridos pero no han tenido mucho auge.

Por otro lado se incita que el sistema soporte dos receptores con el principal objetivo de aprovechar al máximo los diferentes canales usados. En particular, el móvil debería ser capaz de estar monitoreando el canal de control constantemente, independientemente de si se está recibiendo un mensaje de otro móvil. Esto permite a las aplicaciones relacionadas con seguridad continuar con su función sin interrupciones.

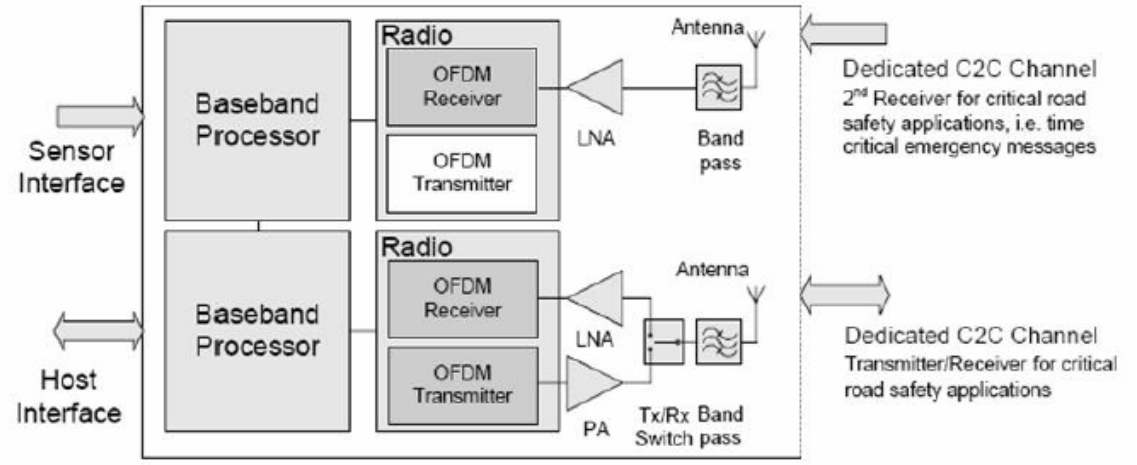

Figura 9 - Diagrama de bloques del transceptor

3.2.4. Sistema de comunicación

El sistema de comunicación es el componente principal de las OBUs y RSUs. Provee transmisión de datos inalámbrica, servicios de comunicación a las aplicaciones y permite la coordinación de la red distribuida. El sistema de comunicación incluye las capas de red y transporte en el stack de protocolos C2C-CC por encima de la capa física MAC. A continuación se describen los principios para la comunicación entre vehículos (V2V) y entre vehículo e infraestructura (V2I).

El sistema de comunicación tiene en cuenta los diferentes requerimientos de las distintas aplicaciones. Típicamente, las aplicaciones orientadas a seguridad están basadas en broadcast y direccionan hacia vehículos en cierta área geográfica, mientras que el resto de las aplicaciones se basan en comunicaciones punto a punto (unicast) y tienen menores restricciones en cuanto a confiabilidad y delay.

Al diseñar estos protocolos de comunicación hay que tener en cuenta los aspectos particulares de una red vehicular. Primero, la red carece de una instancia central para la organización y coordinación por lo que los algoritmos trabajan en forma completamente distribuida. Segundo, la alta movilidad de los nodos resulta en cambios constantes de topología lo que aumenta el overhead de señalización. Tercero, el tráfico de datos generado por los vehículos puede exceder el ancho de banda disponible con la consecuente congestión y perdida de paquetes.

Debido al uso de 802.11, que es una tecnología de corto alcance, la distribución de los vehículos en las rutas o calles tiene gran impacto en el diseño de los protocolos de comunicación. Se diferencian dos situaciones claras: congestión con una gran densidad de vehículos por un lado y grandes áreas con unos pocos vehículos por otro.

En el primer caso se requiere de un eficiente control de la carga del enlace. En contraste, en la segunda situación, como sucederá durante la introducción al mercado de ésta tecnología o en áreas rurales, a pesar de que es probable que no haya congestión del enlace, si lo es que ciertos nodos entren y salgan del alcance de radio, por lo que un ruteo eficiente y el guardar los mensajes para poder retransmitirlos (*store and forward*) será una necesidad.

3.2.5. Direccionamiento geográfico

Se definen dos tipos de direcciones geográficas. El primero trata de las coordenadas geográficas propias de cada nodo necesarias para poder realizar el “GeoUnicast”. El segundo tipo trata de áreas geográficas basadas en figuras geométricas como círculos o rectángulos, definidos por las coordenadas geográficas del centro y el radio en el caso del círculo, por ejemplo. En esta región se podrá direccionar a todos o a cualquiera de los nodos (GeoBroadcast y GeoAnycast). Vale la pena destacar que la dirección geográfica tiene un período de validez. Por eso se debe estampar una marca temporal en dicha dirección.

3.2.6. Algoritmos de encaminamiento

En principio, los algoritmos de encaminamiento están basados en los conceptos mencionados en el párrafo anterior. El C2C-CC distingue entre 4 tipos básicos: GeoUnicast, GeoAnycast y GeoBroacast, todos ellos geográficos, y Broadcast según topología (TopoBroadcast).
GeoUnicast es usado para la transmisión unidireccional de datos desde un nodo fuente a un único nodo destino, por medio de comunicación directa o de múltiples saltos.

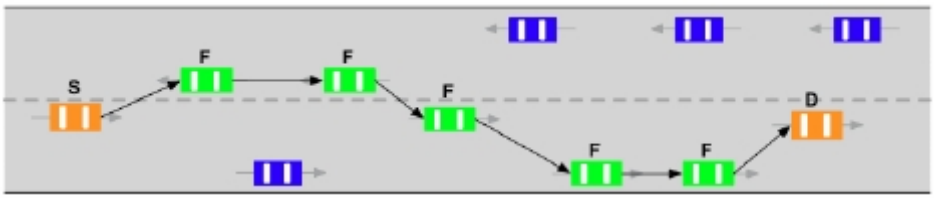

Figura 10 - GeoUnicast

TopoBroadcast se usa cuando se quiere transmitir datos desde un único nodo fuente a todos los nodos en la cobertura de la red ad hoc vehicular. Es decir, por ejemplo, dos saltos dentro de la red.

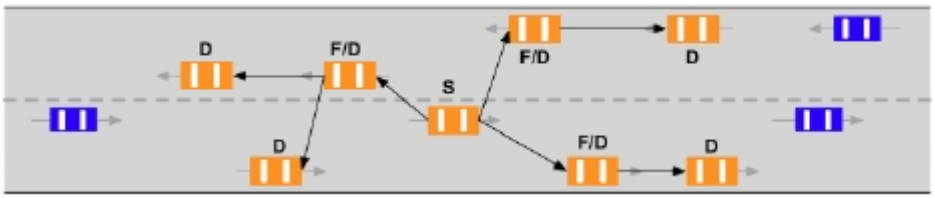

Figura 11 - TopoBroadcast

GeoBroadcast: se define un área geográfica y se hace broadcast a todos los nodos que estén dentro.

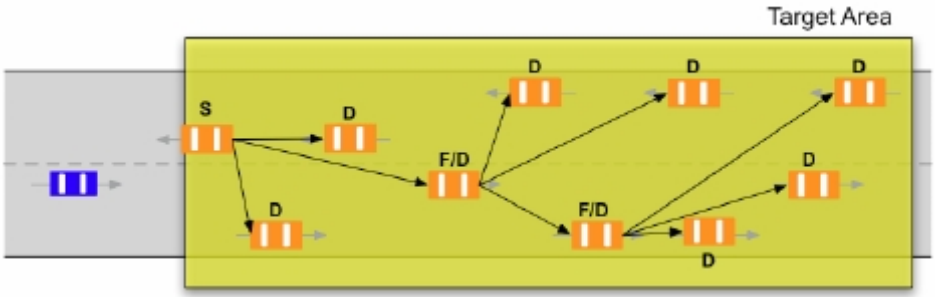

Figura 12 - GeoBroadcast - el originador está dentro del área

GeoAnycast transporta datos de un único nodo fuente a cualquiera de los nodos dentro de un área geográfica definida. A diferencia del GeoBroadcast, el GeoAnycast deja de ser propagado al llegar al área.

En particular para el GeoBroadcast se distinguen dos escenarios. En el primer caso el originador se encuentra dentro del área definida y en el segundo el paquete debe ser propagado hasta llegar a la misma. Este segundo caso se puede interpretar también como una GeoAnycast hacia un vehículo dentro del área seguido de un GeoBroadcast ordinario.

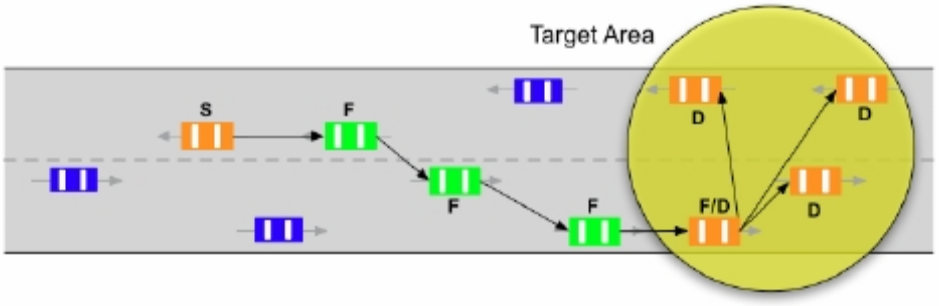

Figura 13 - GeoBroadcast - el originador está fuera del área

3.2.7. Transporte y Control de congestión

Comparado con la capa de red, en protocolo de transporte se encuentra aun en una etapa inicial. Aún no existen protocolos de transporte aplicables a una red ad hoc vehicular y es muy complejo ya sea adaptar un protocolo existente o crear uno de cero.

A pesar de que el tema se encuentra todavía bajo análisis del C2C-CC, a continuación se presentan algunos principios de diseño y posibles aproximaciones a esta ardua tarea.

En Internet, los pedidos de servicios por parte de las aplicaciones, son manejados por los protocolos de transporte quienes a su vez transfieren los pedidos a la capa de red y proveen una transferencia de datos transparente entre las aplicaciones. Funciones típicas de la capa de transporte son: multiplexado de los datos de la aplicación, recuperación de errores, transferencia confiable y ordenada, control de flujo y congestión, etc. En general los servicios de transporte deben ser transparentes y proveer poca información a las aplicaciones, efectivamente así lo hacen TCP y UDP que solamente proveen el puerto y la IP a la capa de aplicación.

Para la comunicación vehicular se han estudiado varios casos concluyendo en que se necesitan las siguientes características de la capa de transporte:

- Tipos de transporte: acordes a las necesidades de broadcast y unicast.
- Transporte libre de errores: Sin importar del tipo de transporte, tanto aplicaciones que son orientadas a seguridad como las que no lo son necesitan un sistema de transmisión libre de errores.
- Confiabilidad: Las aplicaciones orientadas a seguridad tienen requisitos muy exigentes en cuanto a confiabilidad.
- Multiplexado: Puede ocurrir que varias aplicaciones quieran acceder simultáneamente a el sistema de comunicación, en este caso el protocolo de transporte debe poder multiplexar los datos de las distintas aplicaciones
- Retardo y validez de posición: Los paquetes pueden tener importancia espacial y temporal siendo inválidos luego de determinado período de tiempo o fuera de cierta área. Esto impone restricciones adicionales en el uso de buffers, retransmisiones y señalización punto a punto.
- Prioridad de los paquetes: Debido a la naturaleza del sistema existe una clara diferenciación en cuanto a prioridades. Por ejemplo los paquetes relacionados con seguridad son los más importantes, lo que hace necesario que los nodos traten de manera diferenciada a los paquetes cuando se almacenan en un buffer, se descartan o encaminan.
- Agregación de datos de distintas aplicaciones: típicamente, los datos de aplicaciones para seguridad tienen pequeño payload por lo que agregar varios paquetes de distintas aplicaciones puede reducir la carga de la red.
- A la inversa que en el caso anterior, es posible que las aplicaciones tengan demasiados datos como para mandar en un único paquete. Por eso debe ser posible segmentar los mismos para enviarlos y poder rearmarlos en el destino.

3.2.8. Protocolo de capa de red:

Componentes principales:

- Tabla de ubicaciones: es la base de datos central de cada nodo. Contiene una lista de nodos conocidos a un salto de distancia o nodos no vecinos. Se tratan las entradas teniendo en cuenta que tienen cierto tiempo de vida y que necesitan ser actualizadas o removidas. Una entrada incluye la dirección de C2CNET, dirección MAC, dirección IPv6, posición geográfica, velocidad y dirección con marca de tiempo. Se utilizan banderas para indicar el tipo de nodo, disponibilidad de acceso a Internet, etc.

- Armado / desarmado de paquetes: refiere a la generación y proceso del encabezado del paquete cuando el paquete es enviado, encaminado o recibido. Incluye el acceso a la tabla de ubicaciones.

- Beaconing: usado para advertir la presencia de un nodo a sus vecinos. Usado para actualizar la tabla de ubicaciones.

- Encaminamiento: para la re distribución de paquetes hacia el o los destinos. Basados en los algoritmos descriptos anteriormente.

- Servicio de ubicación: parte del ruteo basado en posición. Permite ubicar cierto nodo usando su identificador.

- Manejo de prioridades: incluye clasificación de paquetes, posicionamiento en la cola y organización de los mismos en base a prioridad.

- Control de congestión en capa de red incluye mecanismos de control de potencia de transmisión, indicadores de carga del canal, descarte de paquetes de acuerdo a prioridad, control de velocidad y otros.

La capa de red provee puntos de acceso a las diferentes capas según el siguiente diagrama:

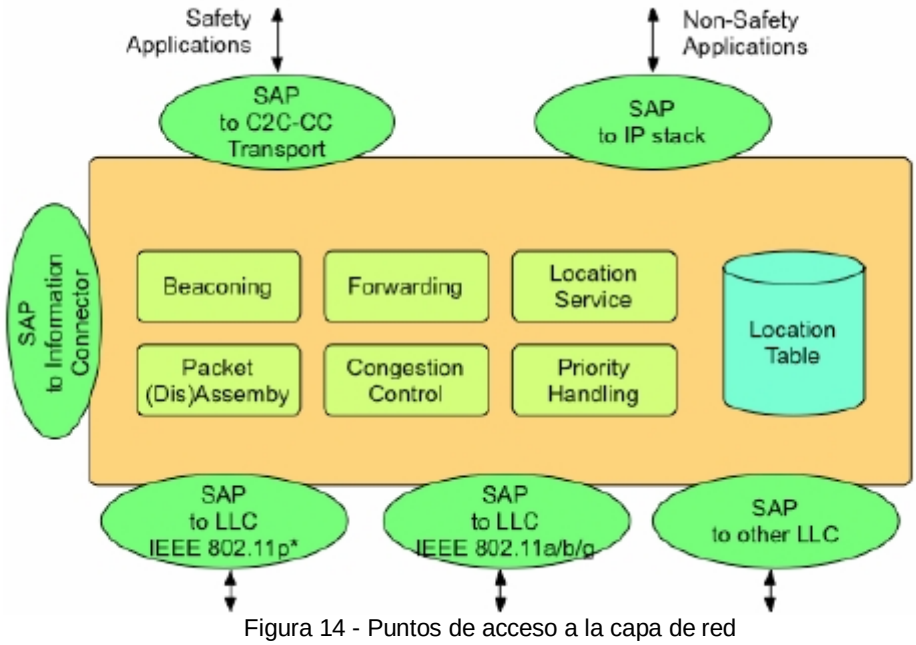

Figura 14 - Puntos de acceso a la capa de red

3.2.9. Seguridad y privacidad

La seguridad y privacidad son indispensables para que el sistema funcione. Toda la información circulando por la red debe ser correcta y confiable. La privacidad de los participantes debe estar asegurada. De lo contrario un usuario con acceso inalámbrico podría ingresar a la red alimentando información falsa a los otros integrantes y pudiendo entonces con malicia provocar accidentes o redirigir un vehículo puntual argumentando que hay un embotellamiento más adelante con el fin de robar a los integrantes o secuestrar el vehículo. Además, es importante brindar seguridad al usuario, ya que si éste no se siente seguro, no utilizará el sistema.

Los estándares actuales de seguridad para redes inalámbricas son adecuados y van a ser usados, pero se requiere de una ampliación a la seguridad en este sentido. Las discusiones en torno a seguridad son muchas y se requieren varios niveles de aprobación, ya sea por las diferentes entidades gubernamentales, como por el usuario mismo. El siguiente diagrama intenta describir la magnitud de este tema y las diferentes aristas por donde se está atacando.

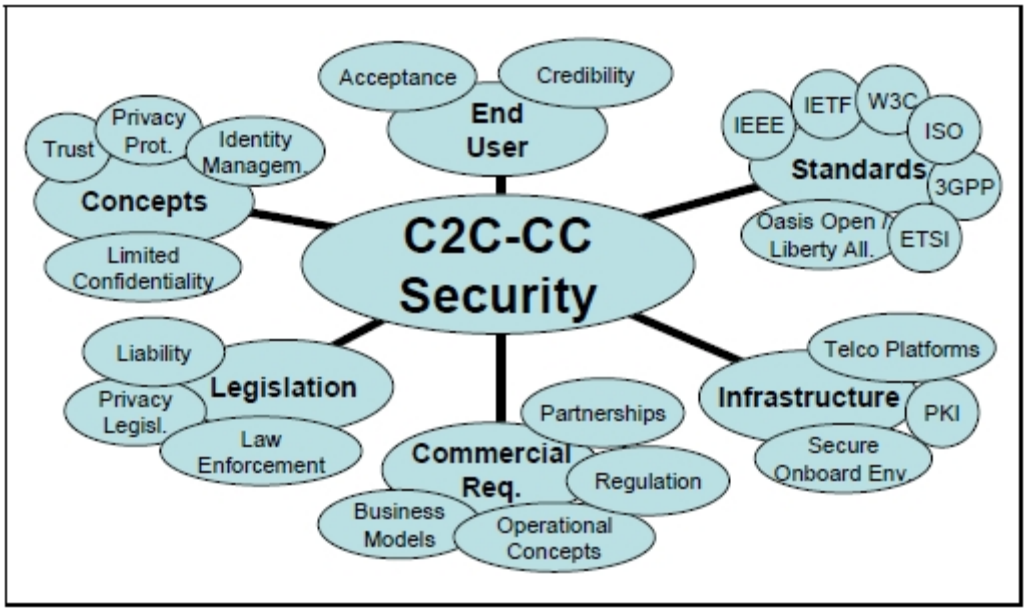

Figura 15 - Actores que intervienen en la seguridad de Car-2-Car

Surge entonces un dilema, por un lado aumentar la seguridad es un requisito necesario para que el proyecto tenga éxito. Por otro, los mecanismos de seguridad agregados aumentan el overhead y necesidad de pre procesar la información antes de poder utilizarla, lo que consecuentemente aumenta los retardos. Por ello los algoritmos no deben ser tan exigentes, haciéndolos menos seguros. Se está trabajando al día de hoy en este dilema y se busca obtener un punto intermedio en donde se cumpla un mínimo de seguridad con una máxima exigencia en la latencia.

3.3. GeoNet

Con una estrecha relación al Consorcio Car-2-Car (C2C-CC), GeoNet orienta sus esfuerzos a combinar el *geonetworking* provisto por C2C-CC con IPv6 y de esta forma lograr un stack de protocolos único para los Sistemas de Transporte Inteligentes (ITS). GeoNet llama a esta combinación *IPv6 geonetworking*.

GeoNet es un proyecto co-financiado por el 7º Programa Marco de la Unión Europea, el cual realiza especial énfasis en las ITS a través del ICT-2007.6.1 (*ICT for intelligent vehicles and mobility services*). Los resultados de GeoNet son incorporados a los esfuerzos de estandarización de las ITS en ETSI, ISO e IETF.

Comenzando su trabajo en febrero de 2008, GeoNet cumplió con su planificación de 2 años y recientemente (en febrero de este año) hizo públicos sus resultados. En el presente capítulo presentamos un resumen de las especificaciones de GeoNet las cuales al día de hoy son la aproximación más concreta a una VANET.

3.3.1. ¿Por qué IPv6?

Ya hemos mencionado anteriormente en el documento la estructura propuesta por Car-2-Car. GeoNet enfoca sus esfuerzos en una pequeña porción de esa estructura, que a su vez resulta ser una de las más importantes para garantizar el éxito en el despliegue de esta tecnología.

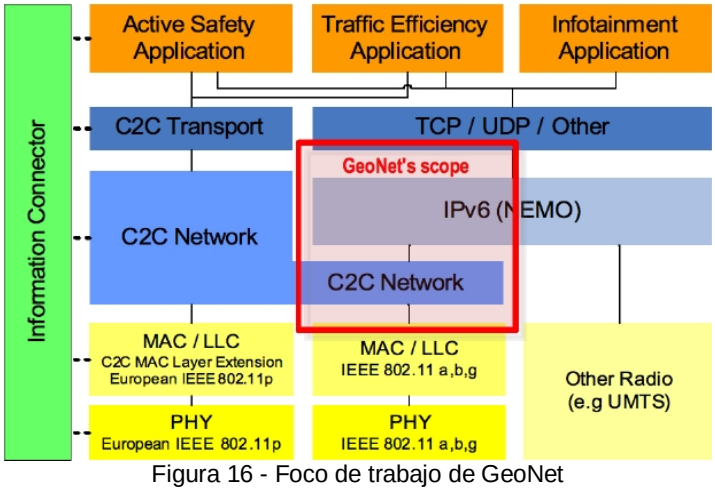

Figura 16 - Foco de trabajo de GeoNet

Como mencionamos, C2C-CC permite a través de su *geonetworking* la comunicación entre vehículos, aún cuando no están en alcance de radio. Sin embargo, para alcanzar un nodo fuera de alcance, se necesitan los nodos intermedios a través de los cuales el mensaje sea propagado. Es de esperar que muchas aplicaciones requieran la comunicación con nodos aún más distantes, o con servicios que se encuentran directamente en Internet, y es aquí que comienza a tomar gran relevancia la incorporación del protocolo IP a este sistema de comunicación.

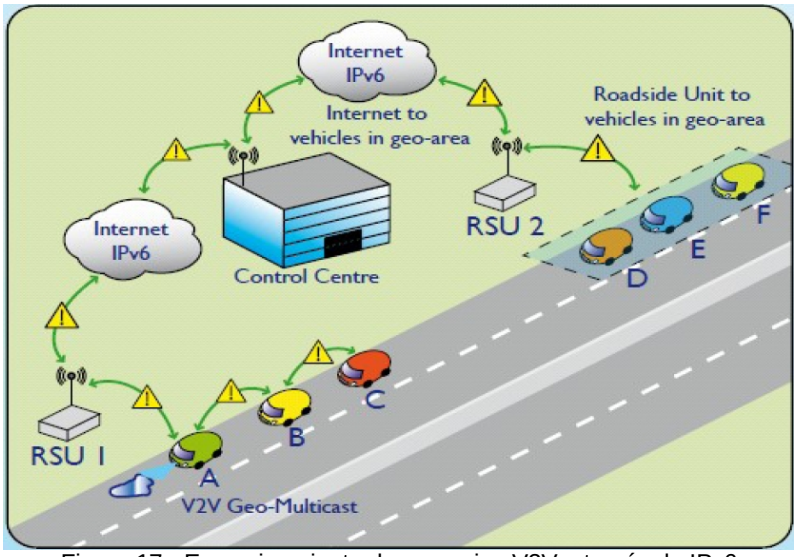

Figura 17 - Encaminamiento de mensajes V2V a través de IPv6

Por otra parte, el soporte a IP permite correr aplicaciones estándar en forma transparente y sin necesidad de realizar modificaciones sobre las mismas. Estas aplicaciones no necesitan por tanto preocuparse por el *geonetworking* ni la arquitectura que está por debajo. IP es naturalmente una capa que separa a las aplicaciones de la forma física de transmitir la información. Gracias al gran despliegue que tiene hoy en día IP, el soporte de dicho protocolo es crucial para un despliegue masivo, permitiendo la comunicación entre vehículos de aplicaciones de variados orígenes y no tan sólo las diseñadas para ITS.

Al soportar IP, el vehículo pasa a estar conectado a Internet pudiendo requerir varias direcciones (por ejemplo, si cuenta con varias dispositivos físicos de conexión – GSM, UMTS, 802.11p, etc). En 1997 había 600 millones de vehículos en el mundo y se estima que al ritmo actual de crecimiento serán 1.200 millones para el 2030. Con sus 4.300 millones de direcciones sumadas a su limitada capacidad para mantener la continuidad en las sesiones, es imposible pensar en IPv4 para esta aplicación. Serían demasiados vehículos para realizar NAT, por ejemplo. Es por esto que cualquier implementación de IP sobre redes vehiculares necesariamente debe ser con IPv6, donde gracias a sus direcciones de 128 bits tenemos una dirección para casi cualquier elemento que queramos conectar a Internet. A su vez, IPv6 provee funcionalidades adicionales fundamentales para las VANETS como lo son la autoconfiguración y la continuidad de la sesión aún en movilidad (NEMO, por ejemplo).

3.3.2. Clasificación de comunicaciones

Al estudiar la comunicación entre nodos a través de GeoNet, debemos tener en cuenta los extremos involucrados en el enlace.
Los extremos de un enlace pueden ser tanto un vehículo (OBU), un dispositivo fijo en la ruta (RSU), como cualquier otro nodo en la Internet. Esto da lugar a 3 escenarios de comunicación:

- Vehículo – Vehículo
- Vehículo – Infraestructura
- Vehículo – Internet

De la misma forma, se clasifica el tipo de enlace entre nodos adyacentes, por lo que podemos tener una comunicación entre vehículos que en determinado punto intermedio pasa a través de infraestructura o Internet.

Otra clasificación importante es en rango de destino. Este es un punto donde se vuelve muy interesante la convivencia entre IPv6 y *geonetworking*.

En IPv6:

- Unicast: el mensaje está dirigido a un nodo individual.
- Multicast: el mensaje está dirigido a múltiples destinatarios.
- Anycast: el mensaje está dirigido al primer nodo que lo reciba dentro de un rango predefinido de destinatarios.

En *geonetworking*:

- GeoUnicast: el mensaje está dirigido a un nodo en una posición específica
- GeoAnycast: el mensaje está dirigido al primer nodo que lo reciba dentro de un área predefinida.
- GeoBroadcast: el mensaje está dirigido a todos los nodos que se encentren dentro de un área predefinida.
- TopoBroadcast: el mensaje está dirigido a todos los nodos a x saltos de distancia.

Veremos más adelante cómo se pueden traducir los rangos de IPv6 en rangos de *geonetworking* y así realizar la integración transparente de aplicaciones que nada saben de *geonetworking* a una red basada en posición.

3.3.3. Módulos funcionales

La arquitectura de GeoNet cuenta con módulos y puntos de acceso a servicios (SAP) que conectan los distintos módulos.

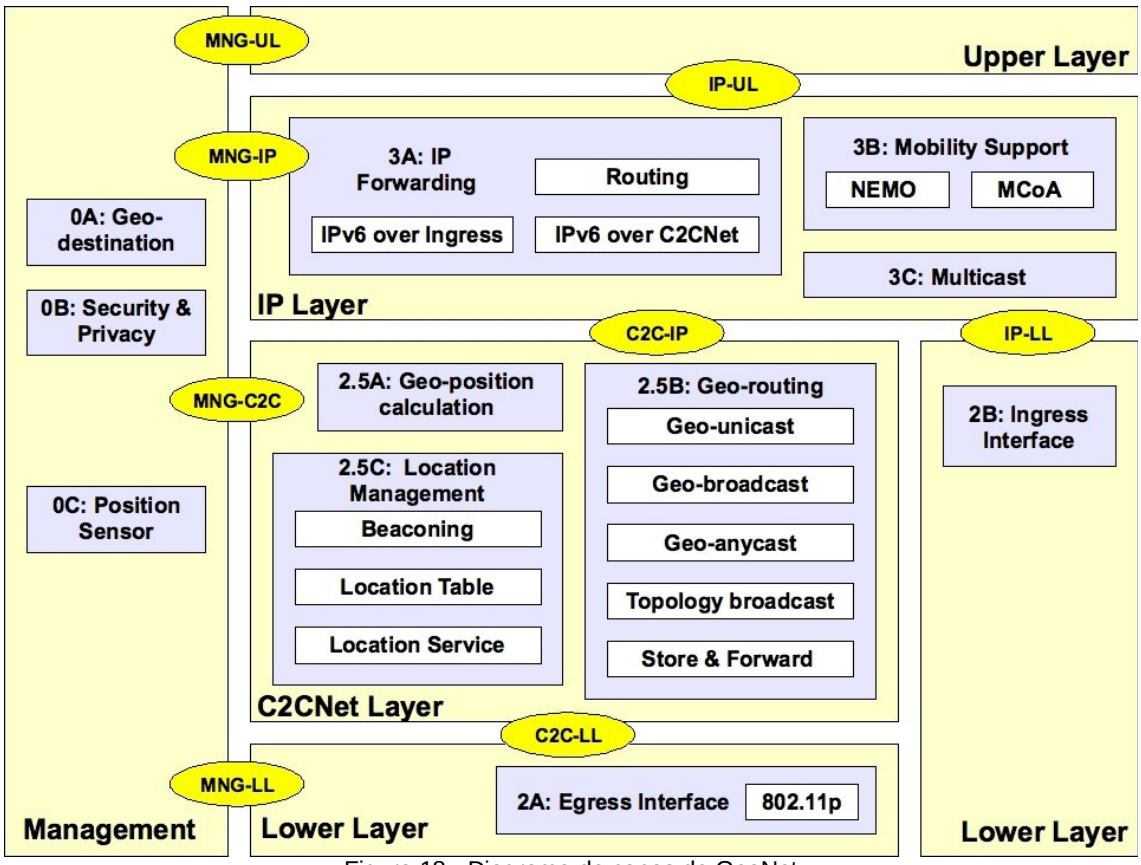

Figura 18 - Diagrama de capas de GeoNet

En la figura 18 se identifican los distintos módulos que componen GeoNet así como las funciones que realiza cada uno. En amarillo se resaltan los SAP que conectan los distintos módulos. Nuevamente recalcamos las ventajas de que las aplicaciones se comunican con una capa IP, que hace las funciones de interfaz entre estas y la comunicación geo-basada.

Módulos de gestión

En estos módulos encontramos funciones de interconexión entre las distintas capas así como brindar servicios para todas las capas. Por ejemplo, el módulo 0A se encarga de conectar la información geográfica entre la capa de aplicación y la capa C2CNet, mientras que el módulo 0C es quien provee a los distintos módulos de información sobre la posición actual del vehículo. Es interesante en que este último punto sea manejado por un módulo independiente puesto que los orígenes de la información de posición pueden ser muy variados (GPS, Galileo, Inercial, etc).

Módulos IP

Esta capa es responsable de ensamblar y reenviar los paquetes IPv6.
Su módulo de IP Forwarding está a su vez dividido en 3 sub-módulos que se encargan de las distintas interfaces hacia las capas más bajas: C2CNet (para *geonetworking*), Ingress (que provee una interfaz IPv6 para otros dispositivos dentro del vehículo o RSU) y Routing (que se encarga de enrutar los paquetes provenientes de las otras dos interfaces).

Cuenta a su vez con un módulo dedicado a la movilidad, que se encarga de mantener la conectividad y sesiones en caso de tener acceso a Internet.
Finalmente, el módulo de Multicast es el encargado de administrar las suscripciones a grupos multicast y realiza la fundamental tarea de conectar los mundos de multicast y geocast, proveyendo a IP Forwarding de la información necesaria para adaptar las tablas de ruteo a este fin.

Módulos de C2CNet

Esta capa es responsable del *geonetworking*.
El módulo 2.5B se encarga de enrutar los paquetes en la red haciendo uso de los métodos GeoUnicast, GeoBroadcast, GeoAnycast, TopoBroadcast y Store and Forward.

De la misma forma, este módulo se encarga de generar los paquetes C2CNet donde van encapsulados los paquetes IP. Para ello debe determinar el siguiente salto a nivel de C2CNet, identificándolo con su identificador de C2CNet.

Las funciones descritas más arriba necesitan información sobre la posición de los nodos destinatarios y vecinos. Para ellos cuenta con la información proveniente del módulo 2.5A que se encarga de calcular la posición de todos los nodos involucrados, y del módulo 2.5C que realiza el descubrimiento de vecinos así como encontrar la posición de un nodo en la red a partir de su identificador.

Módulos de capa superior

Aquí encontramos las aplicaciones y los servicios de capa de transporte. En caso de aplicaciones diseñadas para el geocast, cuentan con los servicios del módulo 0A para realizar la comunicación de información relativa a la posición directamente a la capa de C2CNet.

Módulos de capa inferior

Aquí encontramos las interfaces con las distintas capas físicas que pueden ser utilizadas. Se diferencian las interfaces Ingress y Egress, donde la primera es una interfaz a dispositivos dentro del vehículo o RSU y la segunda es efectivamente la interfaz hacia afuera. Si bien para la interfaz de Egress GeoNet se enfoca en 802.11p, también es posible utilizar otras tecnologías de radio.

3.3.4. Comunicación a través de GeoNet

Tomando en cuenta los módulos descritos antes, una comunicación punto-punto entre dos nodos IP implicaría a las siguientes entidades:

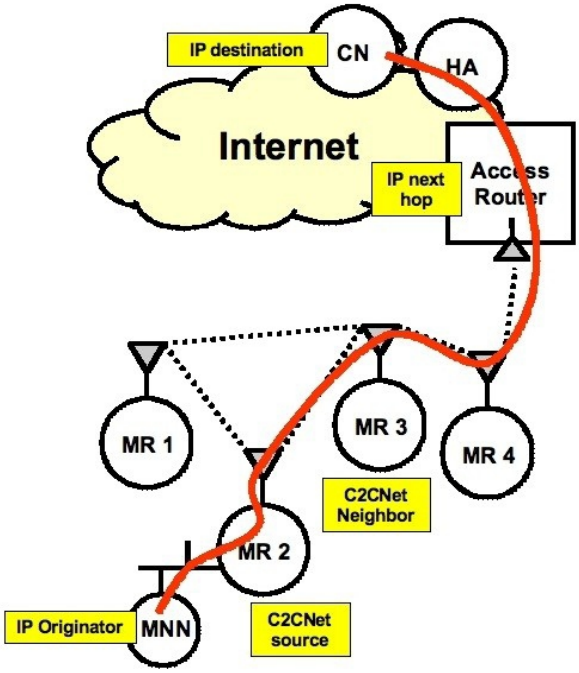

Figura 19 - Encaminamiento de mensajes en GeoNet

Vemos en la figura 19 que el paquete pasa por las siguientes entidades:

- Originador IP: Puede ser incluso un dispositivo dentro del vehículo, conectado al dispositivo GeoNet a través de la interfaz de Ingress.
- Originador C2CNet: Es el primer punto en el que el paquete entra a la red C2CNet.
- Vecino C2CNet: Pueden ser varios saltos a nivel de C2CNet a través de los cuales el paquete se va geo-enrutando.
- Próximo salto IP: Este es nodo IP que figura en la tabla de rutas del originador como el próximo salto. Es importante notar acá que no siempre un próximo salto IP está directamente conectado, sino que pueden haber varios saltos C2CNet entre medio.
- Destinatario IP: El destinatario del paquete.

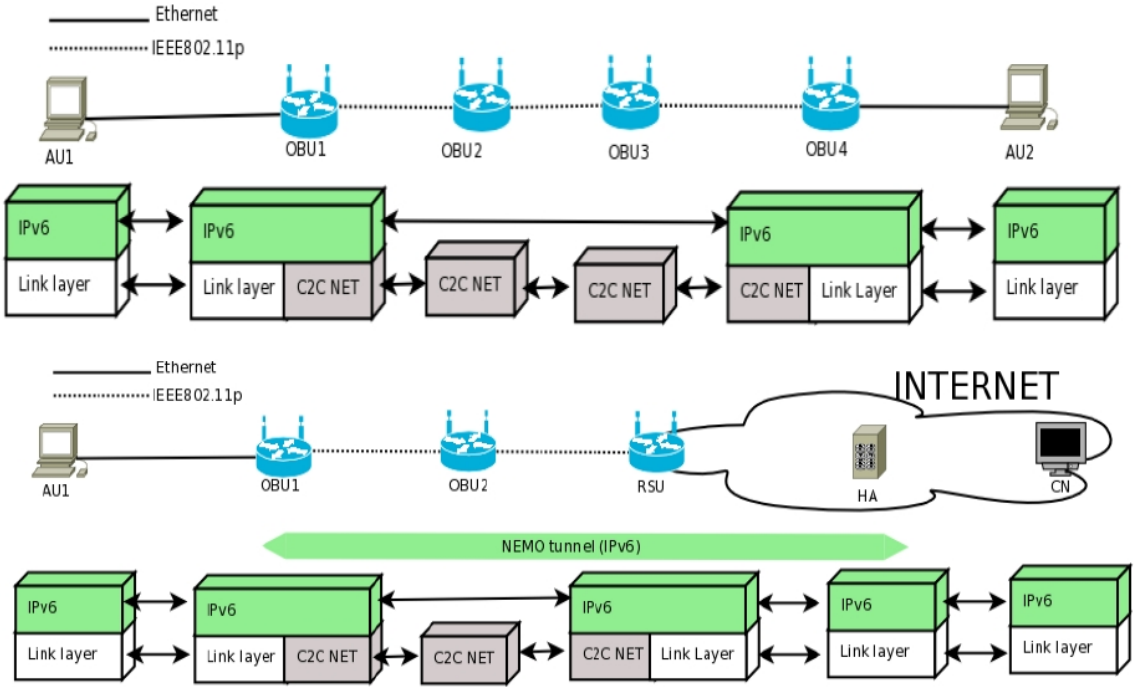

Figura 20 - Encaminamiento de tramas IPv6 a través de C2CNet

Con todas estas entidades en juego, el paquete queda encapsulado de la siguiente forma:

Figura 21 - Encabezados en GeoNet

3.3.5. Implementación y algoritmos

Si bien las implementaciones de Car-2-Car Consortium y GeoNet son privadas y de acceso restringido a los miembros del consorcio, con la publicación de los resultados de GeoNet se hicieron públicos los lineamientos generales en cuanto a la conformación de los paquetes y algoritmos. Esta es la primera vez que este tipo de información se hace pública bajando a tierra años de documentos genéricos.

Encabezado común de C2C

El C2C-CC propone en su demostración del año 2008 un encabezado común que viaja en todos los paquetes de la C2CNet. Este encabezado contiene la identificación del nodo y así como la posición actual, conocido como vector de posición. Este encabezado está presente en todos los paquetes y la información del mismo se reescribe salto a salto con la información del último nodo que retransmitió el paquete. De esta forma cada vez que un nodo recibe un paquete, está obteniendo la información actualizada de la posición de ese vecino desde el cual le está llegando el paquete.

Beacon y tabla de ubicación

Cada nodo mantiene en memoria una tabla de ubicación con la información actualizada de todos los nodos con los que tiene comunicación. Para cada nodo se cuenta con la información contenida en el encabezado común de C2C, sea:

- Identificador MAC
- Identificador C2CNet
- Timestamp
- Posición
- Velocidad y dirección

Esta información se actualiza a través del encabezado común C2C con la llegada de cada paquete y con los paquetes de beacon. A su vez, los nodos son retirados de la lista si no se recibe información actualizada en un plazo predefinido.

El beacon es un paquete enviado solamente con el fin de anunciar la presencia de un nodo en la red. Naturalmente, el beacon contiene tan sólo el encabezado común C2C y no es retransmitido por los nodos adyacentes. Podemos decir entonces que cuando un nodo envía o retransmite un paquete C2CNet, le agrega gratuitamente la información de beacon. Esto último es muy interesante, pues C2CNet hace uso de este hecho para variar la cadencia de envío de beacons y así controlar la carga de la red. Si un nodo está siendo muy activo en la transmisión o retransmisión de paquetes, entonces ya está anunciando su posición y no necesita enviar beacons. Por el contrario, si un nodo está en silencio, necesita enviar beacons para que sus vecinos no lo retiren de la tabla de vecinos.

Al reducir la cadencia de beacons se reduce dramáticamente la carga en la red, evitando así congestión y colisiones.

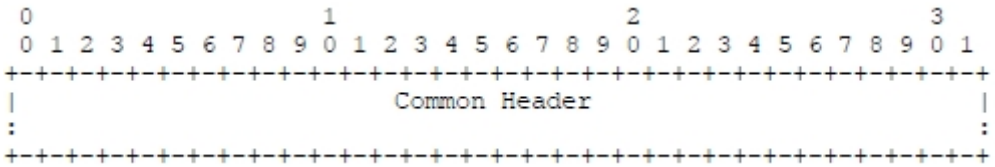

Figura 22 - Paquete de Beacon

Al recibir un paquete de beacon, el nodo primero comprueba si el originador ya se encuentra en la tabla de vecinos y en caso afirmativo actualiza la información. De lo contrario, crea una nueva entrada en la tabla de vecinos con la información del nodo originador.

Servicio de ubicación

Para poder geo-enrutar un paquete a través de varios saltos, es imprescindible conocer su posición geográfica. Se hace necesario contar entonces con un servicio que se encargue de ubicar un nodo en la red conociendo tan sólo su identificador de C2CNet.

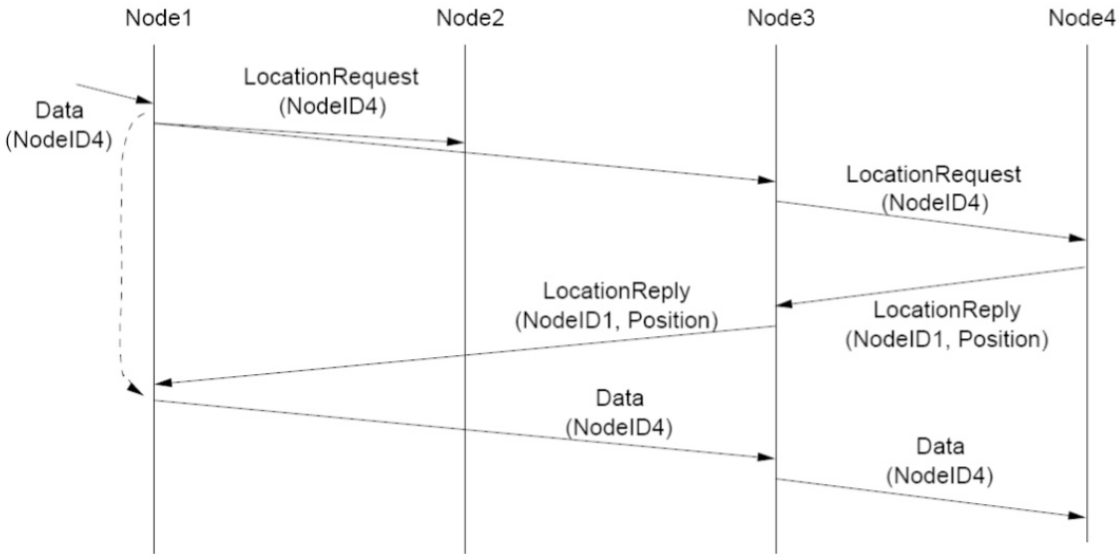

Figura 23 - Secuencia de mensajes en el servicio de ubicación

El nodo que desea conocer la ubicación de otro en la red, envía por broadcast un *location request*, un paquete donde solicita al nodo con identificador *nodeID4*, responder con esta información. Naturalmente, el paquete que se envía contiene la información necesaria para que una vez que se alcanza el nodo en cuestión, este pueda geo-enrutar la respuesta. Una vez que el paquete llega al nodo solicitado, éste responde al originador con su información de posición.

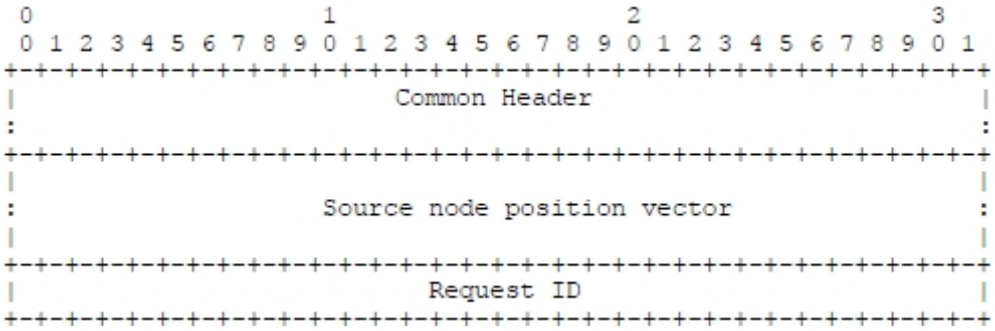

Figura 24 - Paquete de Location Request

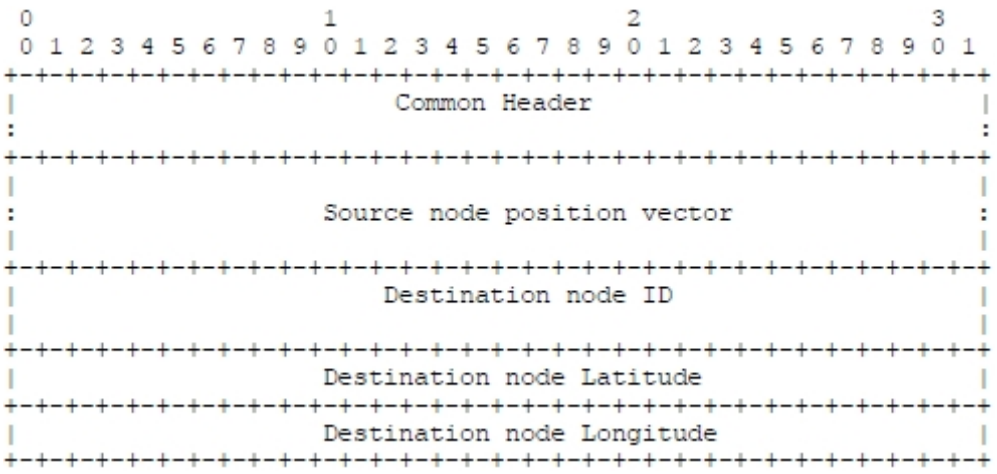

Figura 25 - Paquete de Location Reply

GeoUnicast

Los paquetes GeoUnicast son reenviados nodo a nodo hasta llegar al destinatario. Para el reenvío se utiliza el mecanismo Greedy Perimeter Stateless Routing (GPSR) descrito en este documento. En caso de presentarse un punto donde GPSR no es posible, el algoritmo se recupera enrutando por el perímetro de dicha área.

El algoritmo para el originador del paquete es el siguiente:

1. S genera un paquete GeoUnicast P, dirigido a D.
2. Si en la tabla de ubicaciones D aparece como un vecino directo, entonces el paquete se envía directamente a él.
3. De lo contrario, si no se conoce la posición de D, se solicita la búsqueda del mismo al servicio de ubicación.
4. Si S tiene vecinos directos entonces:
5. Calcular para cada vecino i, la distancia dist(i,D)
6. Enviar P al vecino cuya distancia a D sea menor siempre que ésta sea menor a dist(S,D)
7. De lo contrario, colocar P en el buffer de Store and Forward.

En el lado receptor, cuando el nodo j recibe el paquete P desde el repetidor F el algoritmo es el siguiente:

1. Si fallan los controles de seguridad, descartar P.
2. Chequear si P ya fue procesado anteriormente verificando el identificador del originador y el time stamp. Si ya fue procesado anteriormente, entonces se descarta P.
3. j actualiza su tabla de ubicación con la información de S y F contenida en el paquete.
4. Si j = D, entonces entrega P a las capas superiores.
5. Si D pertenece a la tabla de posiciones de j como un nodo adyacente, entonces se actualiza el encabezado común con el vector de posición de j y se envía el paquete directamente a D.
6. De lo contrario, si j tiene vecinos adyacentes.
7. Calcular para cada vecino i la distancia dist(i,D).
8. Actualizar el encabezado común con el vector de posición de j y enviar el paquete al vecino cuya distancia a D sea la menor, siempre que ésta sea menor a dist(j,D).
9. De lo contrario, poner P en el buffer de store and forward.

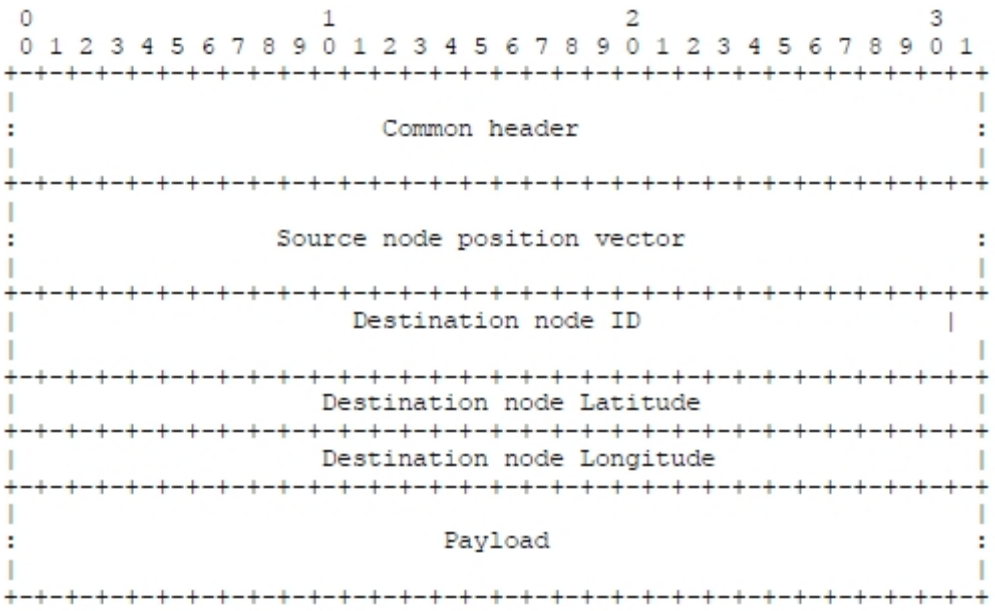

Figura 26 - Paquete de GeoUnicast

GeoAnycast

Los paquetes GeoAnycast se procesan casi idénticamente que los GeoUnicast, con la diferencia que los receptores en vez de chequear si ellos son el destinatario, chequean si ellos pertenecen al área de destino.

El algoritmo para el originador del paquete es el siguiente:

1. S genera un paquete GeoAnycast P, dirigido al área con centro en D y radio R.
2. Si S pertenece al área de destino, entonces no se hace nada.
3. De lo contrario, si S tiene vecinos directos entonces:
4. Calcular para cada vecino i, la distancia dist(i,D)
5. Enviar P al vecino cuya distancia a D sea menor siempre que esta sea menor a dist(S,D)
6. De lo contrario, colocar P en el buffer de Store and Forward.

En el lado receptor, cuando el nodo j recibe el paquete P desde el repetidor F el algoritmo es el siguiente:

1. Si fallan los controles de seguridad, descartar P.
2. Chequear si P ya fue procesado anteriormente verificando el identificador del originador y el time stamp. Si ya fue procesado anteriormente, entonces se descarta P.
3. j actualiza su tabla de ubicación con la información de S y F contenida en el paquete.
4. Si j pertenece al área definida por D y R, entonces entrega P a las capas superiores.
5. De lo contrario, si en la tabla de ubicaciones existe un nodo adyacente k que pertenece al área definida por D y R, entonces se actualiza el encabezado común con el vector de posición de j y se envía el paquete directamente a k.
6. De lo contrario, si j tiene vecinos adyacentes.
7. Calcular para cada vecino i la distancia dist(i,D).
8. Actualizar el encabezado común con el vector de posición de j y enviar el paquete al vecino cuya distancia a D sea la menor, siempre que ésta sea menor a dist(j,D).
9. De lo contrario, poner P en el buffer de store and forward.

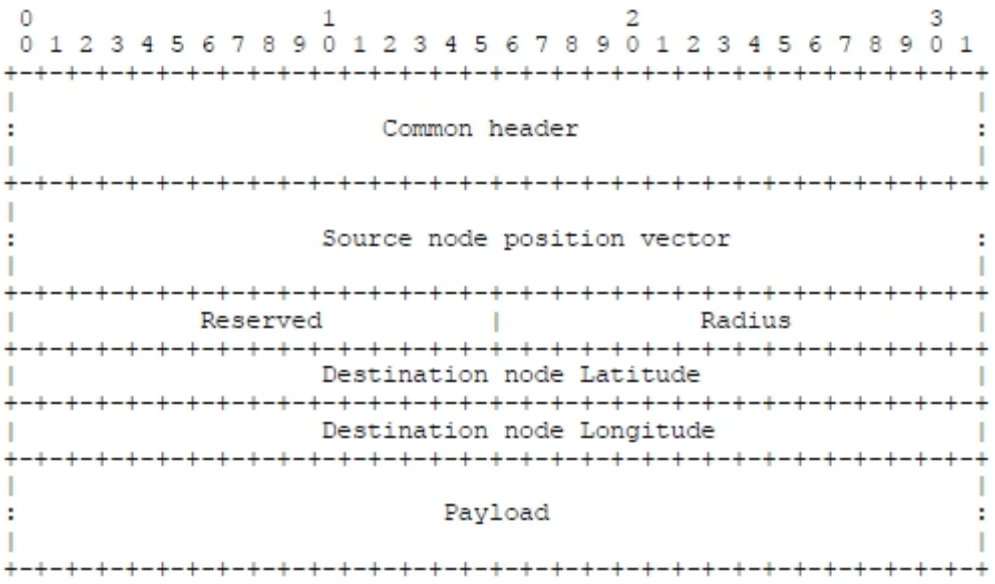

Figura 27 - Paquete de GeoAnycast

GeoBroadcast

El originador de un paquete GeoBroadcast puede encontrarse tanto fuera como dentro del área definida para propagar el paquete. En el caso que se encuentre dentro, entonces el paquete se propaga en dicha área. Caso contrario, el paquete se enruta hacia el área en cuestión con el mecanismo de GeoAnycast y una vez que alcanza el área de destino, se propaga entre todos los nodos que se encuentre ahí.

El algoritmo para el originador del paquete es el siguiente:

1. S genera un paquete GeoBroadcast P, dirigido al área con centro en D y radio R.
2. Si S pertenece al área de destino, entonces S envía el paquete en modo broadcast.
3. De lo contrario, si S tiene vecinos directos entonces:
4. Calcular para cada vecino i, la distancia dist(i,D)
5. Enviar P al vecino cuya distancia a D sea menor siempre que esta sea menor a dist(S,D)
6. De lo contrario, colocar P en el buffer de Store and Forward.

En el lado receptor, cuando el nodo j recibe el paquete P desde el repetidor F el algoritmo es el siguiente:

1. Si fallan los controles de seguridad, descartar P.
2. Chequear si P ya fue procesado anteriormente verificando el identificador del originador y el time stamp. Si ya fue procesado anteriormente, entonces se descarta P.
3. j actualiza su tabla de ubicación con la información de S y F contenida en el paquete.
4. Si j pertenece al área definida por D y R, entonces entrega P a las capas superiores y lo reenvía en modo broadcast.
5. De lo contrario, si F pertenece al área definida por D y R, descarta el paquete.
6. De lo contrario, si en la tabla de ubicaciones existe un nodo adyacente k que pertenece al área definida por D y R, entonces se actualiza el encabezado común con el vector de posición de j y se envía el paquete directamente a k.
7. De lo contrario, si j tiene vecinos adyacentes.
8. Calcular para cada vecino i la distancia dist(i,D).
9. Actualizar el encabezado común con el vector de posición de j y enviar el paquete al vecino cuya distancia a D sea la menor, siempre que ésta sea menor a dist(j,D).
10. De lo contrario, poner P en el buffer de store and forward.

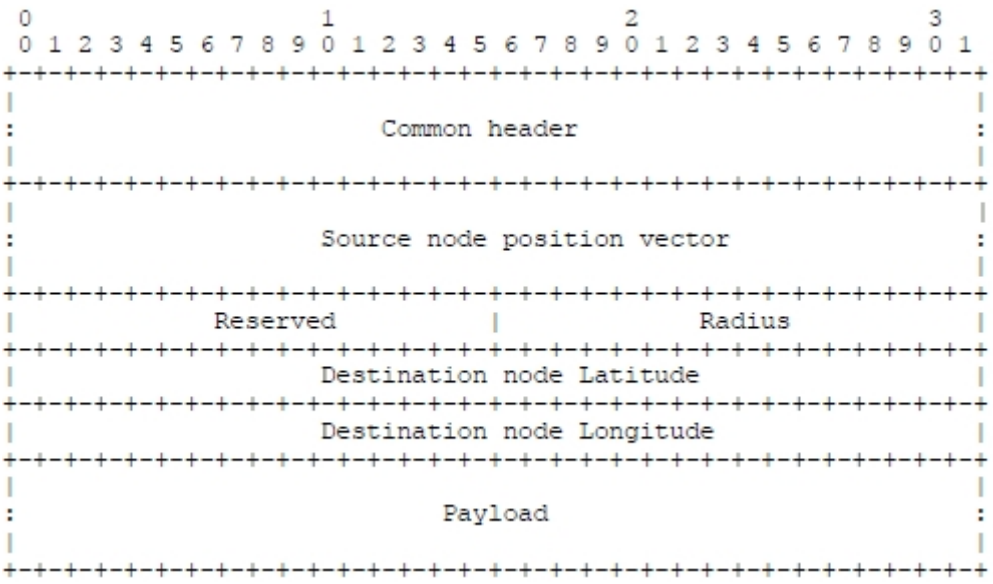

Figura 28 - Paquete de GeoBroadcast

TopoBroadcast

El TopoBroadcast es el mecanismo básico de GeoNet para realizar la inundación de información. Corresponde a un escenario punto-multipunto, donde los paquetes son propagados por un determinado número de saltos. Para controlar la cantidad de saltos, se utiliza un campo del encabezado común C2C diseñado para este fin.

El algoritmo para el originador del paquete es el siguiente:

1. S genera un paquete TopoBroadcast P, a ser propagado por N saltos. Setea por tanto el campo *hop limit* del encabezado común a N.
2. S envía el paquete en modo broadcast.

En el lado receptor, cuando el nodo j recibe el paquete P desde el repetidor F el algoritmo es el siguiente:

1. Si fallan los controles de seguridad, descartar P.
2. Chequear si P ya fue procesado anteriormente verificando el identificador del originador y el time stamp. Si ya fue procesado anteriormente, entonces se descarta P.
3. j actualiza su tabla de ubicación con la información de S y F contenida en el paquete.
4. j entrega P a las capas superiores y decrementa en 1 del valor de *hop limit* del encabezado común de P.
5. Si el nuevo valor de *hop limit* es cero entonces descarta el paquete.
6. De lo contrario, actualiza el encabezado común con el vector de posición de j y reenvía el paquete en modo broadcast.

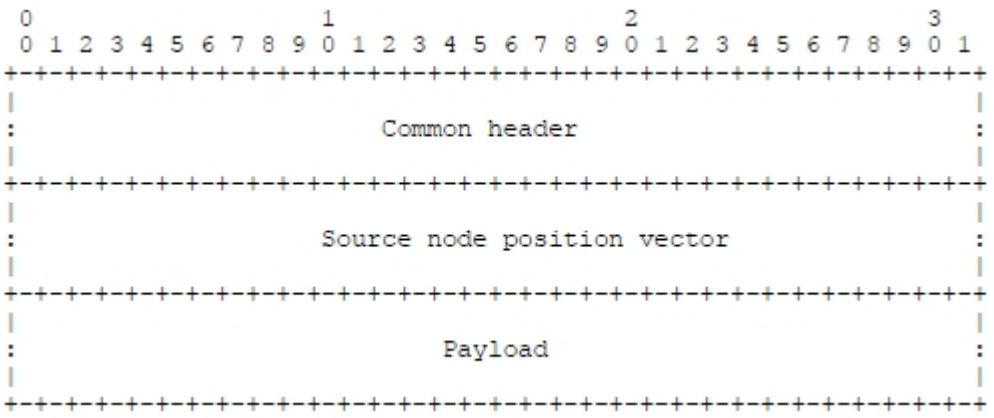

Figura 29 - Paquete de TopoBroadcast

El buffer de Store and Forward

Vimos en los algoritmos anteriores que cuando no se lograba encontrar un candidato a quien enviar un paquete, el mismo era guardado en el buffer de Store and Forward. En este buffer los paquetes aguardan por un cambio en la topología de la red que favorezca el camino para enrutar los mismos. En caso que expire el tiempo máximo de espera sin que el paquete pueda ser encaminado adecuadamente, el mismo se descarta.

Los cambios de topología se dan cuando ingresa un nuevo vecino a la tabla de ubicación, y cuando los miembros de dicha tabla cambian significativamente su posición relativa. Cuando suceden estos hechos, los paquetes almacenados se envían a las entidades que originalmente no pudieron enviarlos (GeoUnicast, GeoAnycast, etc) donde correr nuevamente el algoritmo correspondiente intentando salir con éxito.

3.3.6. Conversión IP – C2CNet

Como adelantamos, GeoNet provee una conversión transparente entre el mundo IP y el *geonetworking*. Los destinos IP deben ser convertidos a destinos C2CNet para poder ser geo-enrutados.

Destino	Capa IPv6	Capa C2CNet
Un nodo específico	Unicast	GeoUnicast
Nodos en determinada área	Multicast	GeoBroadcast
Nodos a x saltos de distancia	Multicast	TopoBroadcast
Un nodo en determinada área	Anycast	GeoAnycast

Tabla 2 - Relación de destinos IPv6 con destinos C2CNet

En la tabla 2 vemos el mapeo de destinos IPv6 en destinos C2CNet. Dicha conversión se realiza de la siguiente manera:

- Unicast: Se arma un paquete GeoUnicast cuyo destino en la capa C2CNet corresponde al nodo que figura como próximo salto IP en la tabla de ruteo IP. Es importante ver aquí que el geo-ruteo se va realizando entre saltos IP y no entre el originador y destino final.
- Multicast: En caso de tratarse de un GeoBroadcast, se arma un paquete de este tipo para ser propagado en el área donde se encuentran los nodos IPv6 multicast. Existen varias formas de pasar la información geográfica desde IPv6 a C2CNet. GeoNet opta por transmitir esa información en el Group ID del encabezado IPv6. Para esto es necesario la coordinación entre las dos capas, a través del módulo 0A para mapear Group IDs de IPv6, con áreas geográficas de C2CNet.

- Multicast: En caso de tratarse de múltiples destinos a una distancia de x saltos del originador, entonces se opta por un paquete TopoBroadcast. En este caso sólo se necesita conocer la cantidad de saltos.
- Anycast: Se crea un paquete GeoAnycast utilizando los mismos mecanismos que para GeoBroadcast.

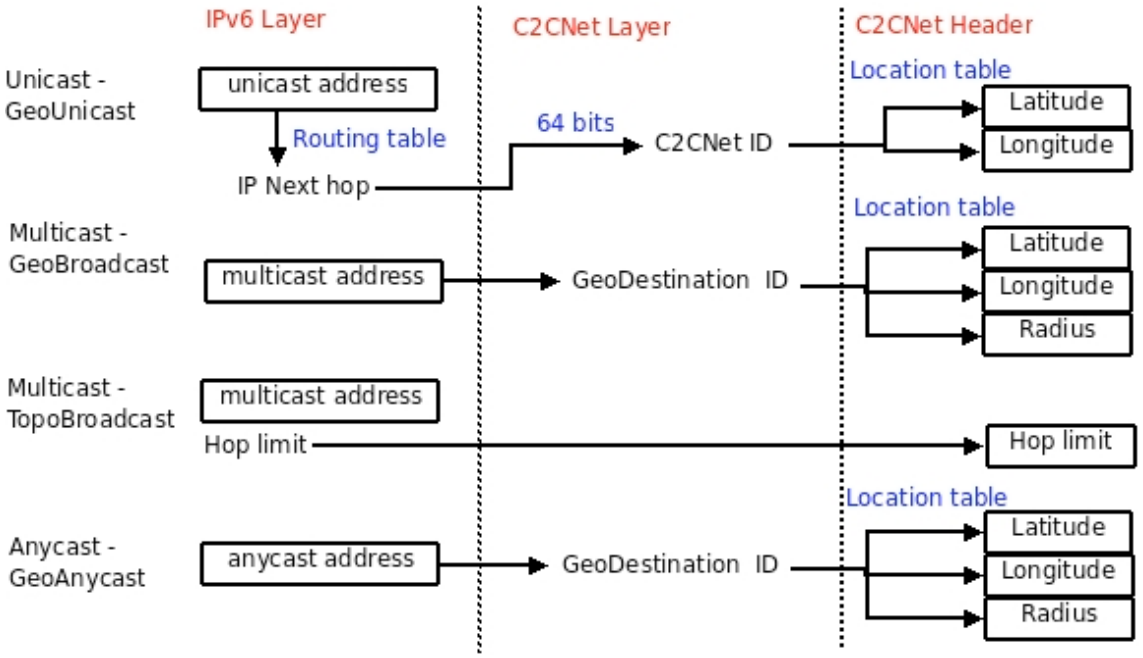

Figura 30 - Mapeo de direcciones IPv6 en direcciones C2CNet

3.4. Wireless Access Vehicular Environment (WAVE)

WAVE es un conjunto de estándares que tienen como objetivo definir un sistema de comunicación inalámbrico capaz de proveer servicios a vehículos y todo tipo de entidades de transporte. Estos servicios incluyen aquellos reconocidos por la arquitectura ITS (Intelligent Transportation System) así como también muchos otros contemplados por la industria automotriz y de trasporte.

El sistema WAVE está pensado para soportar comunicaciones tanto vehiculo-infrastructura como vehiculo-vehiculo. WAVE busca satisfacer los exigentes requerimientos del mundo vehicular como son bajos tiempos de conexión, constantes cambios en la topología, baja latencia en las comunicaciones, imprescindible para mensajes relacionados a la seguridad, entre otros.

Se provee de un canal de control común para señalización, donde se hace uso de mensajes cortos especializados (WSM – WAVE Short Message) para servicios de alta prioridad y mensajes de control del sistema, y varios canales de servicio capaces de soportar tanto mensajes WSM como tráfico IPv6, utilizados para aplicaciones de propósito general y de baja prioridad.

Los principales estándares involucrados en la arquitectura WAVE son el ieee1609.1, ieee1609.2, ieee1609.3, ieee1609.4, e ieee802.11p, todos ellos en desarrollo. Cada uno represente un bloque de la arquitectura y se describen a continuación. En la siguiente figura se muestra un diagrama de la arquitectura del sistema:

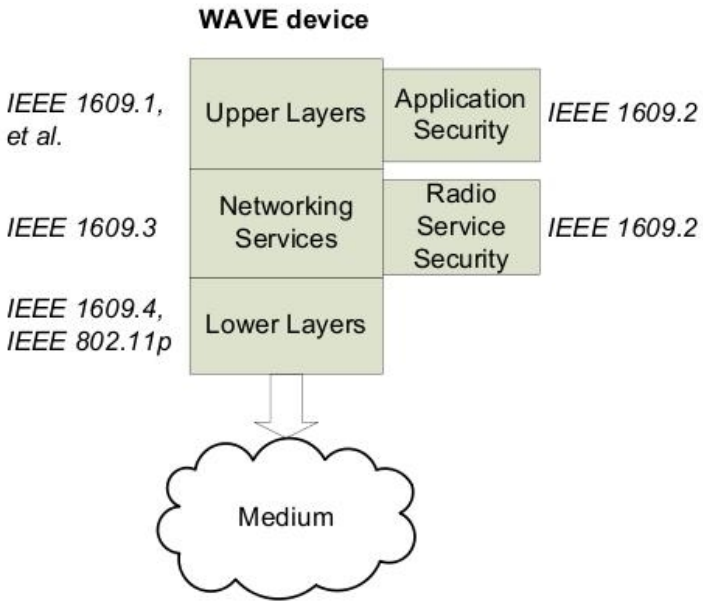

Figura 31 - Diagrama de capas de WAVE

3.4.1. Capas PHY y MAC

El estándar **ieee802.11p** pertenece al grupo de protocolos wifi, y cumple las funciones de capa física y buena parte de las funciones de capa mac. Está especialmente diseñado para comunicaciones inter-vehiculares y presenta las siguientes particularidades:

- define un modo de operación especifico para comunicaciones vehiculares
- el transmisor emplea OFDM en la banda ISM libre de 5.9 Ghz
- define canales de 10 y 20 MHz de ancho de banda
- define especificaciones de frecuencia más ajustadas para el transmisor
- agrega requerimientos de rechazo de canales adyacentes para al receptor
- permite comunicaciones con y sin un WBSS (WAVE Basic Service Set)
- define un rango extendido de temperaturas en las cuales debe ser capaz de operar

El estándar **ieee1609.4** es una extensión al estándar ieee802.11p con el objetivo de lograr una operación del sistema en múltiples canales. Posibilita mecanismos efectivos para controlar operaciones de capas superiores (ieee1609.3) a través de múltiples canales sin la necesidad de conocer datos de capa física.

En particular se definen 2 tipos de canales, uno de control (CCH – Control Channel) y uno de servicios (SCH – Service Channel). El primero se utiliza para transmitir mensajes de tipo WSM (WAVE Short Message), mensajes característicos del estándar de alta prioridad, principalmente orientados a aplicaciones de seguridad. También en este canal es posible anunciar servicios WAVE. El segundo tipo de canal es donde se hacen efectivos esos servicios una vez solicitados en el CCH, soportando el manejo tanto de mensajes WSMP (WAVE Short Message Protocol) como de mensajes IP. Existen varios canales de servicio.

El canal CCH es monitoreado a intervalos regulares para poder atender sin demoras mensajes de alta prioridad. Se realiza un ranurado en el tiempo en intervalos de 50 ms y se asigna a cada canal un intervalo de manera alternativa como se muestra en la figura 32.

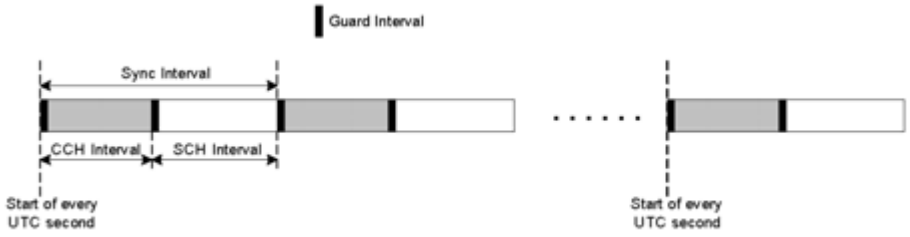

Figura 32 - Intervalos de canales CCH y SCH

Es imprescindible que los dispositivos estén sincronizados por lo que se debe utilizar un GPS como referencia de tiempo. En caso de ser necesario, se puede extender el intervalo correspondiente al canal de control sobre el intervalo contiguo correspondiente al canal de servicios.

En Estados Unidos ya existe un rango de frecuencias asignado para las comunicaciones DSRC (Direct Short Rango Communication) el cual consta de 7 canales de 10 Mhz de ancho de banda como muestra la siguiente figura:

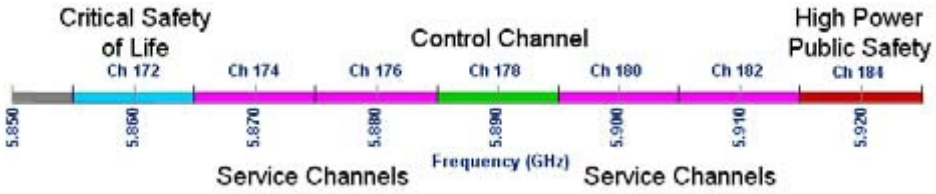

Figura 33 - Asignación de frecuencias para DSRC en EE.UU.

La existencia de varios canales de servicios distintos en el espectro permite atender varios servicios simultáneamente una vez que han sido apropiadamente coordinados en el canal de control.

Las funcionalidades provistas por ieee1609.4 son las siguientes:

- Channel routing: controla el ruteo de paquetes provenientes de la capa LLC a su canal designado, ya sea de control o de servicios, sin necesidad de realizar operaciones de coordinación de canal por parte de la capa mac.

- User priority: ieee1609.4 soporta una variedad de aplicaciones seguras y no seguras con hasta 8 niveles de prioridad predefinidos. Estos son utilizados por algoritmos de contención a la hora de ganar el acceso al medio, siendo los de mayor prioridad los mensajes relacionados a la seguridad. Existe un buffer para cada uno de estos 8 niveles donde los paquetes son encolados. A su vez, estos buffers tienen 3 parámetros característicos:

    - AIFS (Arbitration Inter-Frame Space) – Tiempo mínimo entre que el canal esta libre y se puede comenzar a transmitir
    - CW (Contention Window) – Ventana para implementar un backoff aleatorio
    - TXOP (Transmit Oportunity) limit – Tiempo máximo durante el cual se puede transmitir luego de haber obtenido un TXOP. Si el límite es 0 solo se podrá trasmitir un MSDU.

  Mediante el uso de estos parámetros, los paquetes provenientes de los diferentes buffers participan primero en un mecanismo de contención interno para luego realizar otro externo y así ganar el acceso al medio.

- Channel coordination: coordina los intervalos activos de cada canal acorde a las operaciones de sincronización de la capa mac para que los paquetes se transmitan en el canal adecuado. Es necesario para soportar intercambio de datos entre dispositivos que no son capaces de monitorear el canal de control a la vez que intercambian datos en canales de servicios. Cuando un dispositivo se une a una WBSS (WAVE BSS) es imprescindible la sincronización y la coordinación de canal para asegurar que todos los dispositivos están monitoreando el CCH adecuadamente, donde se transmiten los mensajes de mayor prioridad relacionados a seguridad vehicular.

- MSDU data transfer: servicios de capa mac para la transferencia de datos ya sean de control o de servicio, mediante IPv6 o WSMP. Pueden enviarse tramas WSMP directamente entre dos nodos sin conexión previa en el canal de control mientras que para comunicaciones en un canal de servicio primero el proveedor del servicio debe anunciarlo en el canal de control, generando una WBSS.

3.4.2. Capa de red

Por encima de la capa mac, se encuentra la capa de red, cuyos servicios están definidos en el estándar **ieee1609.3**. La siguiente figura muestra las capas que abarca:

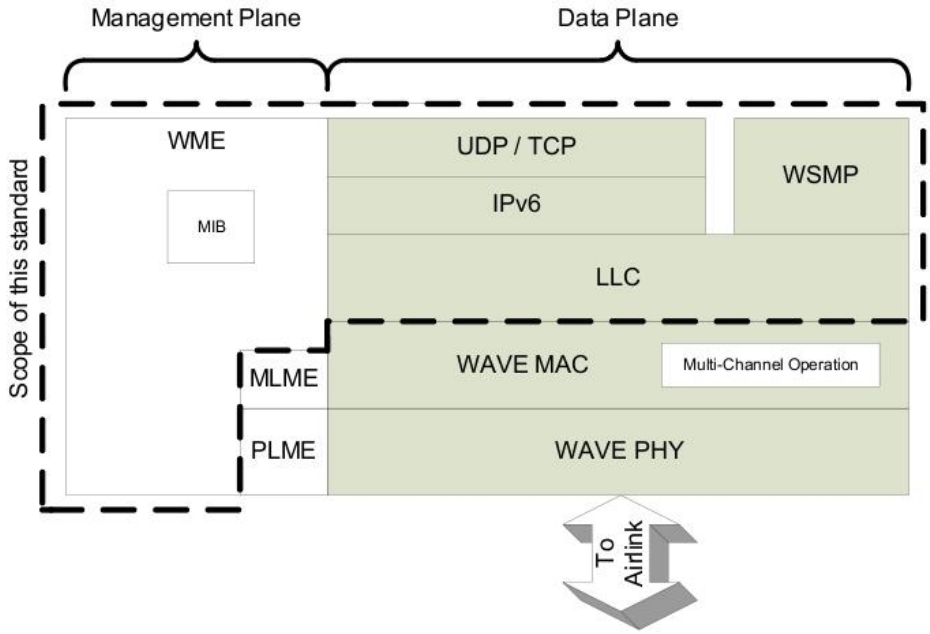

Figura 34 - Capas que abarca 1609.3

También se distinguen en la figura dos áreas definidas como el plano de datos y el plano de administración. El plano de datos contiene los protocolos y hardware usados en la transmisión de datos, y se encarga generalmente del tráfico generado o destinado a aplicaciones, aunque soporta también tráfico entre planos de administración de diferentes maquinas o tráfico entre el plano de administración y el de datos.

El plano de administración en cambio lleva a cabo tareas de configuración y mantenimiento del sistema. Sus funciones emplean servicios del plano de datos para intercambiar tráfico de administración entre dispositivos. Se definen entidades específicas de administración para ciertas capas como son PLME (Physical Layer Management Entity) y MLME (Mac Layer Management Entity), además de WME que es una colección más general de los servicios de administración.

Se ve entonces que el estándar ieee1609.3 consiste en las capas intermedias del plano de datos y la totalidad del plano de administración. En resumen, se hace cargo de los siguientes servicios:

- Servicios de datos:
  - LLC (Logical Link Control)
  - IPv6
  - UDP y TCP
  - WSMP (WAVE Short Message Protocol)

- Servicios de administración:
  - Mecanismos de registro para las aplicaciones
  - Administración de WBSS (WAVE Basic Service Set)
  - Monitoreo del uso del canal
  - Configuración IPv6
  - Monitoreo del RCPI (Received Channel Power Indicator)
  - Mantenimiento del MIB (Management Information Base)

Los dos protocolos posibles de comunicación a nivel de red son WSMP o IPv6. WSMP (WAVE Short Message Protocol) está diseñado específicamente para operaciones del entorno vehicular, siendo muy eficientes en la utilización del canal. Los mensajes WSM pueden ser transmitidos en cualquier canal y permite a las aplicaciones controlar directamente parámetros de capa física como ser el canal utilizado y la potencia de transmisión, a diferencia de IPv6 que controla estos parámetros a través de los diferentes perfiles que define el protocolo.

El formato de los WSM es el siguiente:

1	1	1	1	1	4	2	variable
WSM Version	Security Type	Channel-Number	Data Rate	TxPwr_Level	Provider Service Identifier	WSM Length	WSM Data

Figura 35 - Formato de los WAVE Short Messages

Las aplicaciones pueden elegir el intercambio de datos en el contexto de un WBSS o no. Si se establece un WBSS, es posible utilizar tanto WSM como IPv6 sobre algún canal de servicio (aunque el anuncio y coordinación de una WBSS se realiza en el canal de control). Si se decide no establecer un WBSS, la comunicación está limitada a mensajes WSM en el canal de control.

El dispositivo que anuncia un WBSS y por ende los respectivos servicios asociados (pueden existir más de un servicio asociado a un mismo WBSS) se denomina “proveedor”. Cualquier dispositivo que se una a ese WBSS para utilizar sus servicios se denomina “usuario” (pueden haber varios usuarios utilizando distintas combinaciones de servicios dentro del mismo WBSS).

Al operar sin un WBSS, la aplicación prepara mensajes WSM con una primitiva tipo request y los manda a la dirección MAC de broadcast en el CCH. Los dispositivos receptores registran la aplicación a través de un número único que la identifica, el PSID. Dado que en este punto ya se conoce la existencia y dirección del proveedor se puede continuar con el intercambio en el CCH usando unicast o broadcast según corresponda.

Si se opera dentro de un WBSS, esta es anunciada por el proveedor junto con los servicios ofrecidos al igual que antes, pero con un tipo de mensaje especializado llamado WSIE (WAVE Service Information Element), conteniendo información detallada de los servicios ofrecidos, parámetros de capa física, de ruteo, prioridad, entre otros, como se observa en la figura 36.

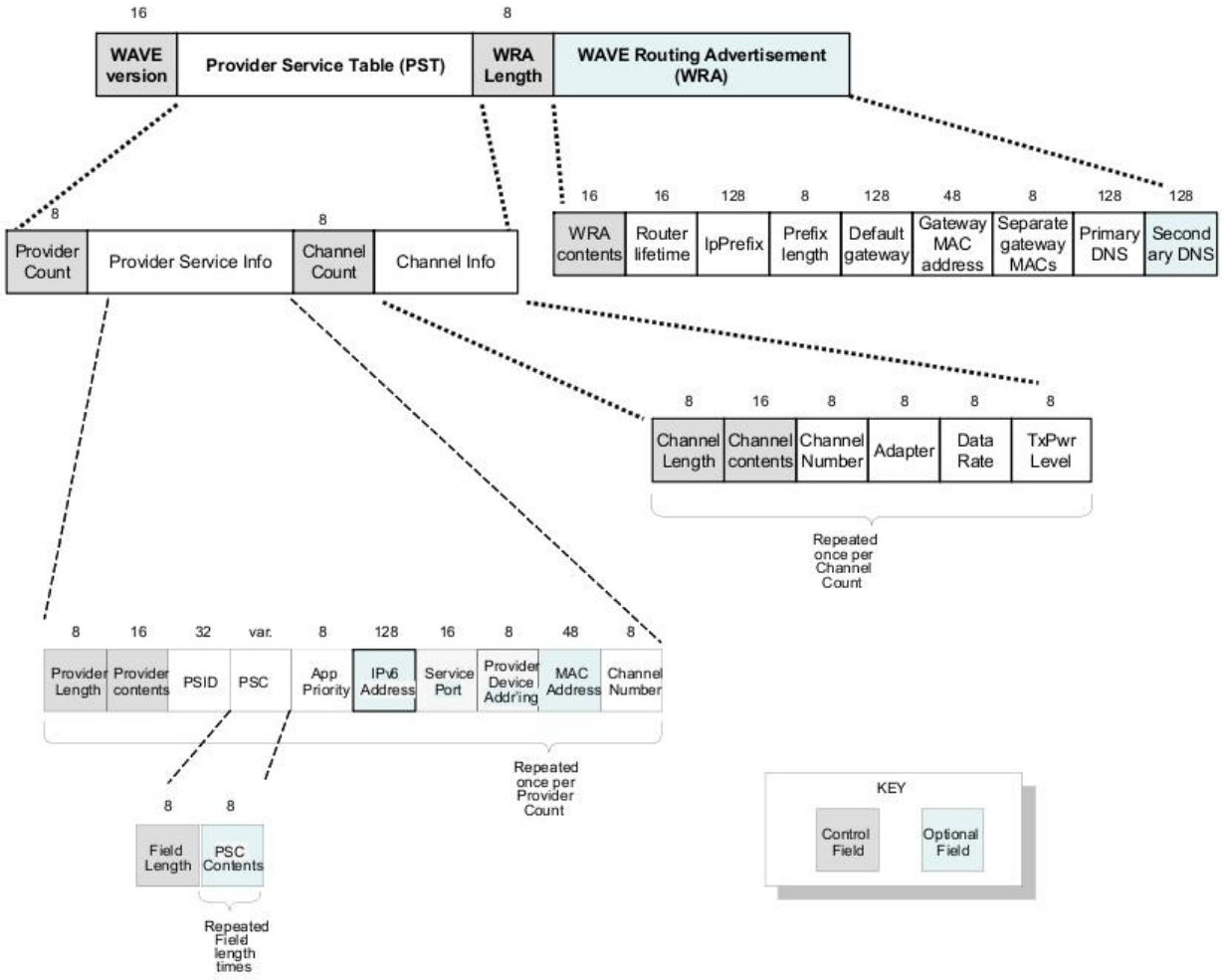

Figura 36 - Formato de la trama WAVE Service Information Element

3.4.3. Capa de aplicación

Sobre la capa de red, el estándar **ieee1609.1** define ciertas aplicaciones específicas capaces de proveer interoperabilidad entre las distintas aplicaciones de alto nivel que utilicen WAVE con el fin de simplificar el sistema a bordo de los vehículos.

Se definen dos tipos de entidades: RSU (Road Side Unit) y OBU (On Board Unit). Las RSU representan la infraestructura fija idealmente dispuesta a lo largo de la ruta, mientras que las OBU son unidades móviles montadas en los vehículos. Típicamente las RSU tienen una aplicación que provee un servicio y la OBU tiene una aplicación cliente que hace uso de ese servicio, aunque una OBU puede también proveer servicios. Pueden existir además aplicaciones remotas a las RSU que brinden servicios a las OBUs.

El estándar ieee1609.1 define dos aplicaciones específicas, el Resource Manager (RM) que reside en las RSU, y su contrapartida, el Resource Command Processor (RCP) que reside en las OBU. Las aplicaciones remotas a la RSU se denominan Resource Manager Aplication (RMA) y se comunican con el RCP de los vehículos (OBUs) mediante el RM de las unidades fijas (RSUs). Se contempla además que las OBUs puedan brindar servicios a otras OBUs por lo que deberán contar en estos casos con su correspondiente RM como lo establece la arquitectura.

El estándar describe como el RM multiplexa los servicios de múltiples RMA, cada una de las cuales se comunica con múltiples OBUs. El propósito de estas comunicaciones es proveer acceso por parte de las RMA a los recursos de las OBU, como lo son la memoria, la interfaz de usuario, e interfaces a otros sistemas onboard controlados por el RCP, de manera consistente, interoperable y a tiempo para satisfacer los requerimientos de las RMA.

Las distintas aplicaciones mencionadas se muestran en el siguiente esquema:

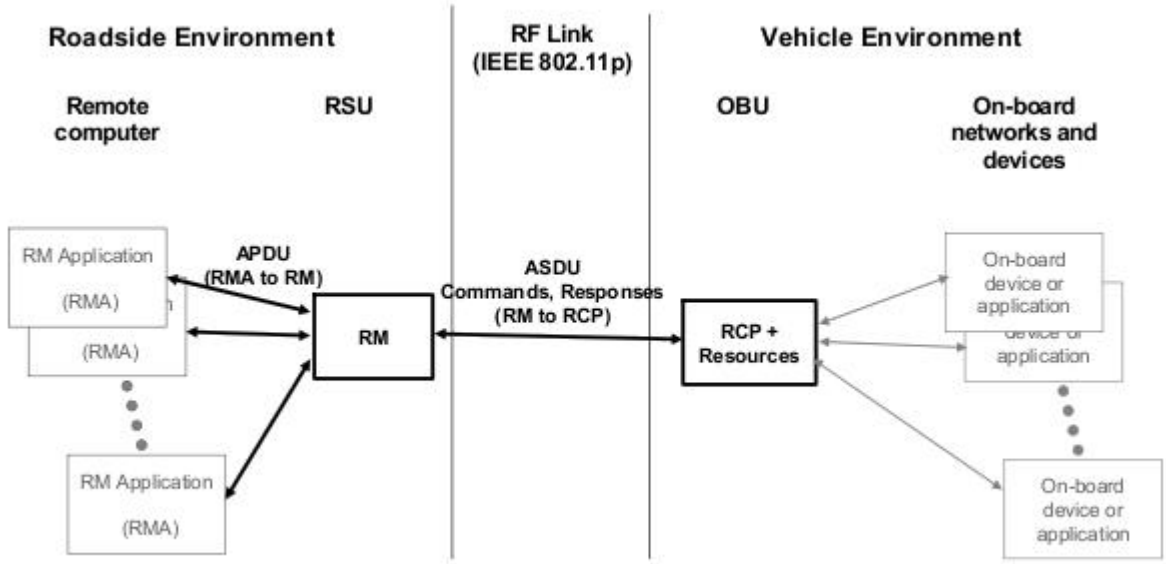

Figura 37 - Aplicaciones descritas en 1609.1

Cada RMA debe registrarse con la RM que desea interactuar y le pasa la lista de recursos a los cuales desea tener acceso. El RM contiene una lista con los recursos solicitados por las

diferentes RMA, y difunde esta información a la espera de OBUs que puedan atender esos pedidos. El RCP de la OBU se encarga de detectar y dar acceso a los recursos solicitados si corresponde.

El intercambio de comandos y respuestas entre el RMA y el RM se realiza mediante mensajes encapsulado en formato APDU (Aplication Protocol Data Unit) sobre la red cableada. Al llegar al RM, este extrae la carga útil, y la encapsula en mensajes UDP que luego transmite a través de un enlace RF al RCP del OBU correspondiente. Este interpreta los comandos recibidos y manda de vuelta las respuestas en mensajes UDP. Por último, el RM extrae la carga útil y la encapsula nuevamente en mensajes APDU para dirigirlas hacia la RMA que solicitó la información. El formato APDU se describe en el estándar.

En resumen, el estándar especifica:

- Los servicios provistos por el RM al RMA
- Cómo son usados los servicios provistos por ieee1609.3 para que el OBU detecte la presencia de un RM o un RMA.
- Cómo el RCP reconoce y responde a la presencia de un RM y a las RMA asociadas a este
- El manejo de recursos de memoria de las OBU
- El set de comandos disponibles por las RMA para administrar estos recursos, así como también, cómo son intercambiados los comandos y respuestas entre las RMA, RM, y RCP
- El uso de recursos especializados de lectura/escritura que permiten la transferencia de datos con otros equipos del OBU controlados por el RCP

3.4.4. Seguridad en las comunicaciones

Ya sea debido al carácter crítico que pueden tener los mensajes relacionados a la seguridad vehicular, o simplemente para mantener la confidencialidad de la información difundida, es necesario contar con mecanismos que aseguren el intercambio seguro y confiable de mensajes entre las diferentes entidades.

El estándar **ieee1609.2** (originalmente denominado ieee1556) define los servicios usados para proteger los mensajes de ataques tales como acceso a la información por personas no autorizadas, alteración de los mensajes, o repetición de los mismos, entre otros. Se definen funciones de seguridad tanto para la capa de red como para la capa de aplicación. Aparte de los servicios típicos de seguridad como son la confidencialidad, autenticidad e integridad, el sistema WAVE impone que se provean mecanismo para mantener el anonimato del usuario final, ya que información propia puede llegar a ser difundida, y también porque se puede llegar a tener en memoria información de otros usuarios.

Para cumplir con estos requisitos, el estándar utiliza ampliamente los mecanismos conocidos de criptografía como ser: algoritmos simétricos, algoritmos asimétricos, funciones hash, y certificados y autorizaciones digitales. La utilización de cada uno de ellos depende de los requerimientos y restricciones de cada situación en particular, ya que, por ejemplo, existe un compromiso entre velocidad de codificación-decodificación y nivel de seguridad, que determina que un algoritmo sea mejor que otro según la situación. Si bien este tema es de fundamental importancia, supera el alcance de este docuemento.

4. Ruteo en VANETs

4.1. Introducción

A medida que las comunicaciones inter vehiculares van tomando fuerza, surgen necesidades de contar con algoritmos de ruteo robustos. Recordemos que las VANET no sólo proveen un medio que podría hacer el tránsito vehicular más seguro y eficiente sino que además proveería al usuario servicios del tipo infotainment como acceso web, video streaming, etc. Se necesita entonces un ruteo eficiente antes de poder escalar los actuales proyectos a redes VANET eficientes y de gran tamaño.

Vale la pena mencionar que numerosos algoritmos han sido propuestos en Mobile Ad-hoc Networks (MANETs) para resolver este problema y muchos de ellos han demostrado ser efectivos. En una primera aproximación uno podría pensar que esos mismos algoritmos podrían ser trasladados a las redes vehiculares ya que estas últimas son nada más que un caso particular de MANETs, pero eso sería incorrecto. Si bien es cierto que ambos tipos de redes comparten muchos aspectos como el ancho de banda limitado, necesidad de comunicación multi-salto entre nodos móviles, etc, existen diferencias fundamentales que impiden que los algoritmos implementados para MANETs sean eficientes, o incluso que lleguen a funcionar en VANETs. Entre estas diferencias están:

Topología muy dinámica

Debido a la gran velocidad entre vehículos en movimiento y al corto alcance de radio que los mismos tienen, la topología varia muy rápidamente y consecuentemente también lo hacen los caminos formados.

Energía y capacidad de procesamiento o almacenamiento.

Una gran diferencia con las redes MANET es que los vehículos tienen una fuente de energía potente y recargable. Esto junto al espacio disponible nos permite alojar procesadores potentes y memoria suficiente para nuestros propósitos. En cambio las MANET se caracterizan por dispositivos portátiles pequeños que requieren que sus baterías duren el máximo tiempo posible como notebooks, PDAs, etc. Es fundamental aprovechar esta ventaja al máximo realizando por ejemplo una mayor cantidad de procesamiento local que en comparación con las MANET.

Comunicación geográfica

La mayoría de los vehículos modernos tienen la opción de contar con un GPS integrado y sistemas de navegación que nos permiten conocer exactamente la ubicación, velocidad y rumbo de cada nodo. Con esta información podríamos predecir o estimar una zona en donde es altamente probable que el nodo se encuentre un tiempo después de recibido el primer mensaje, facilitando entonces la predicción de rutas al mismo.

Variabilidad del ambiente de comunicación

En general es posible obtener características particulares del entorno en el cual las VANET se comunican. Es decir, en una ruta nacional la comunicación es unidimensional lo que simplifica muchas tareas. También en ciudades tenemos restricciones sujetas a la estructura de calles e intersecciones. Al contar con posicionamiento y mapas podemos aprovechar esta situación conocida a la hora de diseñar el algoritmo.

Retardos exigentes

A pesar de que no se exigen velocidades de transmisión muy altas en VANETs si es necesario cuidar los retardos ya que en aplicaciones orientadas a seguridad pasa a ser más importante el delay máximo y no tanto el promedio.

Movimiento restringido

Todos los vehículos tienen limitaciones en el movimiento. A diferencia de notebooks o PDAs en un campus universitario, los vehículos deben someterse a las limitaciones de las rutas y calles. Así como también a los semáforos y velocidades máximas permitidas. Estos datos nos pueden ayudar a predecir la posición de un vecino un tiempo después de haber recibido la información de posición del mismo.

Antes de continuar con este capítulo presentamos un árbol en donde se clasifican los distintos métodos de ruteo organizados temporalmente y con vínculos de influencias en su desarrollo.

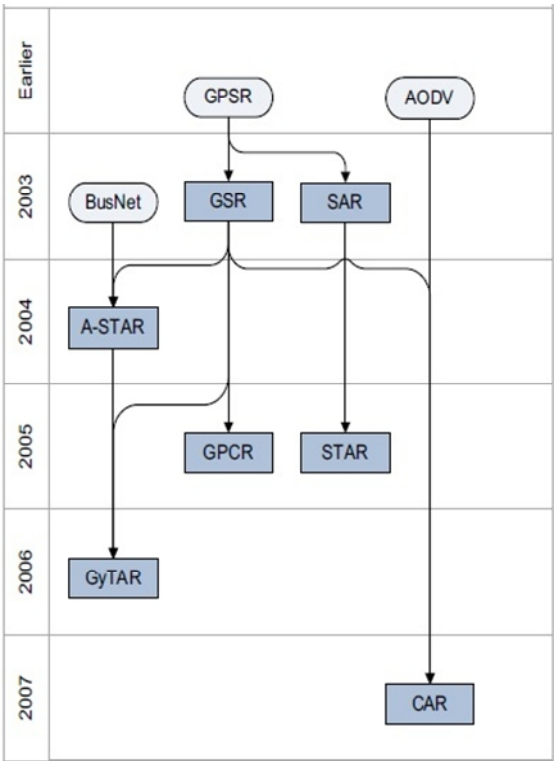

Figura 38 - Línea de tiempo de protocolos de ruteo

Comenzamos con un análisis de los problemas que tiene simplemente realizar el broadcast por inundación, como método de propagación de paquetes y luego se describen rápidamente los protocolos mencionados haciendo más énfasis en uno u otro de acuerdo a su relevancia.

4.2. Ineficiencia de broadcast por inundación

La propagación de información a través de broadcast adquiere especial importancia en redes ad hoc, como es el caso de las VANET. La gran movilidad de los nodos hace necesario recurrir con frecuencia a la propagación de mensajes a través de broadcast, por ejemplo, para intercambiar información de ruteo, o directamente enviar una señal de alarma a todos los nodos.

La forma más sencilla de implementar broadcast es la que se conoce como inundación. El nodo que quiere transmitir un mensaje de tipo broadcast lo hace a todos los vecinos que están en alcance de radio, estos reciben el mensaje y lo retransmiten, y así sucesivamente, obteniéndose de esta manera la propagación de información a través de toda la red.

Sin embargo, el broadcast por inundación es muy ineficiente. En una red con n nodos, este mecanismo implica realizar n transmisiones, aún si n-1 nodos están en alcance de radio del nodo originador del mensaje, lo que se resolvería eficientemente con una sola transmisión.

En particular si se utiliza CSMA/CA como es el caso de 802.11, la inundación presenta principalmente 3 inconvenientes:

- Redundancia de broadcast: cuando un host decide retransmitir un mensaje de broadcast a sus vecinos, pero estos ya han recibido el mensaje

- Contención: cuando un host manda un mensaje de broadcast, todos los vecinos que lo escuchen y quieran retransmitir el mensaje, lo intentarán al mismo tiempo, provocando contenciones al tratar de acceder al medio compartido

- Colisiones: debido a la ausencia del diálogo RTS/CTS en mensajes de broadcast, y la ausencia de detección de colisiones (CD), estas ocurrirán frecuentemente

A continuación se muestra un análisis extraído de “The Broadcast Storm Problem In A Mobile Ad Hoc Network” sobre cada uno de estos puntos, además de plantear 5 soluciones posibles.

4.2.1. Análisis de broadcast redundante

En la figura 39 se muestran dos esquemas de ruteo óptimo, el nodo blanco genera el mensaje, el gris es el único que lo retransmite. En (a), difundir la información toma 2 transmisiones mientras que tomaría 4 si se hiciera inundación. En (b), toma 2 transmisiones en vez de 7 si se hiciera inundación.

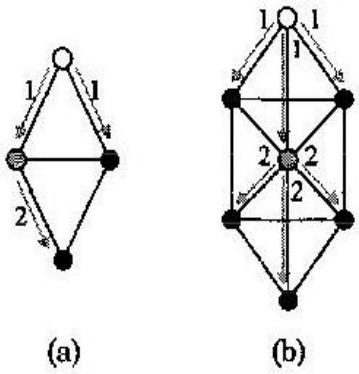

Figura 39 - Ruteo óptimo

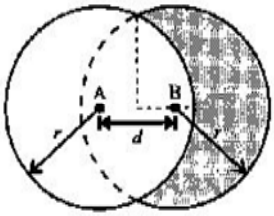

Figura 40 – Alcance aportado por una retransmisión

Considero el escenario de figura 40, en el que A manda un mensaje y B decide retransmitirlo. Sean S_A y S_B las áreas de alcance de A y B respectivamente. El área adicional que puede ser beneficiada con la retransmisión de B es el área gris, llamémosle $S_{A\cap B}$. Sea r el radio de los círculos y d la distancia entre A y B, entonces $|S_{B-A}| = |S_B| - |S_{A\cap B}| = \pi r^2 - INTC(d)$, donde INTC(d) es la intersección de dos círculos de radio r cuyos centros distan una distancia d.

Si d = r, el área de cobertura $|S_{B-A}|$ es la máxima y equivale a

$$\pi r^2 - INTC(r) = r^2\left(\frac{\pi}{3} + \frac{\sqrt{3}}{2}\right) \approx 0.61\pi r^2$$

Esto implica que la retransmisión sólo aportará de 0 a 61 % de cobertura adicional. Se puede hallar el valor promedio de $\pi r^2 - INTC(d)$ para un punto B situado de manera aleatoria en S_A integrándolo la expresión sobre el círculo de radio x centrado en A, con x en [0,r]:

$$\int_0^r \frac{2\pi x[\pi r^2 - INTC(x)]}{\pi r^2} dx \approx 0.41\pi r^2$$

Esto implica que, en promedio, la retransmisión del nodo B aportara un 41% de cobertura adicional.

Si un nodo C estuviera en S_{B-A}, éste escucharía la transmisión de A y podría escuchar la retransmisión de B antes de retransmitir. Si aún así, desea hacerlo, se puede hallar mediante simulación que, en promedio, aportará un 19% de cobertura.
En general, mediante simulaciones se puede obtener la cobertura media de un host que desee retransmitir el mensaje luego de haberlo escuchado k veces, como se muestra en la figura 41,

donde cf(nk) es la cobertura adicional esperada. Se puede ver que para k≥4 la cobertura adicional esperada está por debajo de 0.05%.

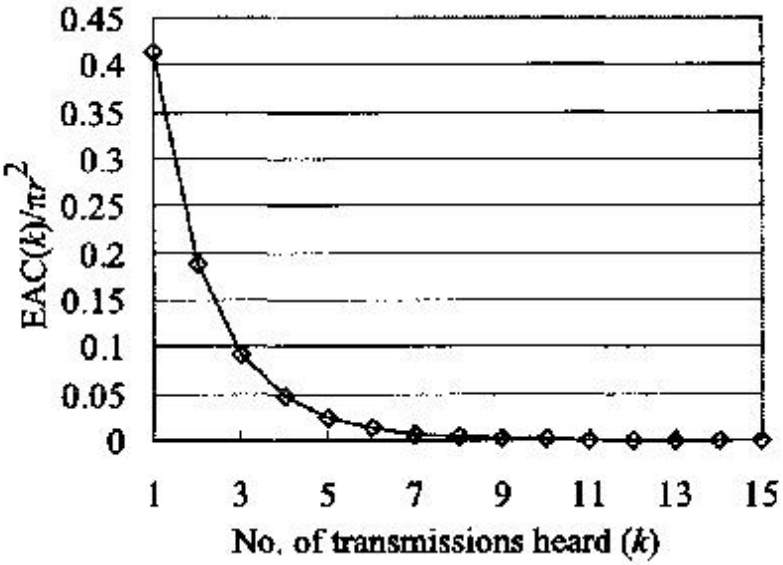

Figura 41 - Aporte medio de cobertura luego de escuchar k transmisiones

4.2.2. Análisis de contención

Considero el caso en el que el nodo A transmite un broadcast y n nodos lo reciben. Si todos esos nodos tratan de retransmitir, habrá contenciones entre algunos de ellos en su lucha por ganar el acceso al medio.

Considero n=2, lo que equivale a un nodo A que transmite y dos nodos B y C que reciben. Si coloco a B en algún lugar aleatorio en el alcance de A, C debe estar en $S_{A\cap B}$ para que haya contención al retransmitir, por lo que la probabilidad de contención es $\frac{|S_{A\cap B}|}{\pi r^2}$.

Sea x la distancia entre A y B, entonces puedo hallar la probabilidad esperada de contención integrando la expresión anterior sobre el círculo de radio x desde 0 hasta r:

$$\int_0^r \frac{2\pi x(INTC(x)/\pi r^2)}{\pi r^2} dx \approx 59\%$$

Se puede obtener un resultado más general mediante simulación, es decir, dados n host, cuál es la probabilidad de que k de esos n experimenten contención. La figura 42 muestra como esta probabilidad aumenta rápidamente a 80% parar n≥6.

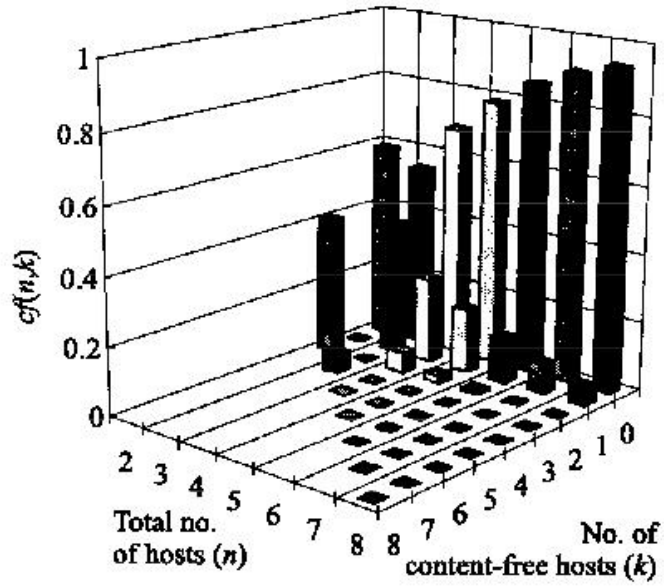

Figura 42 - Contención debido a demasiadas retransmisiones

4.2.3. Análisis de colisiones

El mecanismo CSMA/CA requiere que el host inicie el procedimiento de backoff. Es altamente probable que dos host quieran transmitir inmediatamente luego de que otro host haya finalizado su transmisión ya que han estado sensando el canal ocupado por cierto tiempo. Esto ocasionaría colisiones muy frecuentes y como no podemos detectarlas, esta colisión no sería detectada hasta que expire el tiempo de espera del ACK. Por eso se intenta evitar las colisiones, para ello el algoritmo de random backoff elige un numero aleatorio para la ventana de contención (CW). Este valor está comprendido entre 0 y CWmax (valor determinado en el estándar) y refiere a la cantidad de timeslots que el host tiene que esperar para poder transmitir.

Aún con este procedimiento, cuando un host X transmite un broadcast que es escuchado por varios vecinos, existen varias razones para que ocurran colisiones. No se utiliza el dialogo RTS/CTS en mensajes de broadcast, por lo que no hay solución al problema del nodo oculto provocando más colisiones. A su vez, al no existir detección de colisiones las tramas se trasmitirán en su totalidad aún si han colisionado significando una pérdida de tiempo útil de transmisión.

4.2.4. Posibles soluciones

Una primera solución al problema pude ser utilizando un esquema probabilístico. Cuando se recibe un mensaje broadcast, un host retransmitirá con cierta probabilidad P. Se debería agregar además cierto retardo aleatorio para evitar colisiones y contenciones.

Otra solución sería utilizar un esquema basado en conteo. Cuando un host quiere retransmitir un mensaje, este probablemente quedará bloqueado momentáneamente debido a que el medio está ocupado (procedimiento de backoff además de que puede haber varios mensajes encolados esperando acceder al medio). El host puede de esta manera escuchar varias veces el mismo mensaje antes de poder retransmitirlo. Si se lleva la cuenta de cuantas veces es

escuchado se puede tomar un numeró limite C tal que si la cantidad de veces que se ha escuchado el mensaje supera a C no se transmite.

Una tercera solución podría ser un esquema basado en distancias. Si un host ha escuchado un mensaje broadcast, sería bueno que lo retransmitiera solo si la distancia entre él y el host que originó el mensaje, no fuera demasiado chica, ya que como se vio anteriormente, la cobertura adicional obtenida sería muy baja. Es necesario fijar una distancia mínima a partir de la cual retransmitir. Además se precisa conocer esa distancia, que puede ser estimada a través de la potencia recibida.

Una alternativa más eficiente sería basada es localización. Si se dispone de datos de localización (como los que entrega un GPS por ejemplo) se podría estimar con mucha mayor precisión la cobertura adicional obtenida para poder así decidir si retransmitir o no.

Un último esquema posible es el basado en clusters como muestra la figura 43. Un clusters es un grupo de host en el cual existe un head que tiene alcance a un salto con todos los del grupo, y uno o varios gateways, host en la frontera del área de cobertura del head que son capaces de comunicarse con gateways de otros clusters. Se pretende de esta manera evitar una retransmisión por parte de los nodos que no sean gateways ni head. Se podría agregar a este mecanismo, alguno de los anteriores para evitar por ejemplo que dos gateways cercanos retransmitan el mismo mensaje.

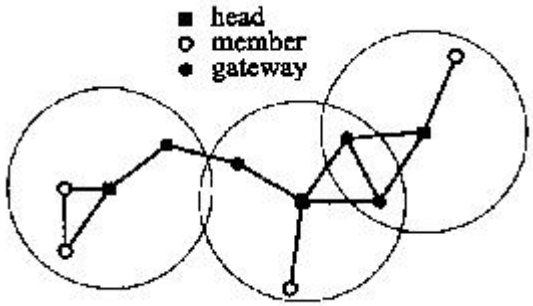

Figura 43 - Clusters

4.2.5. Conclusiones

Se desprende de esta sección que la transmisión de mensajes broadcast mediante inundación en redes móviles de tipo ad hoc es muy ineficiente. Se vuelve por lo tanto de vital importancia encontrar una estrategia útil para la propagación de mensajes broadcast ya que son la base, entre otras cosas, de la mayoría de los algoritmos de ruteo propuestos hoy día para redes de este tipo. Se plantean además 5 soluciones a este problema que deberán ser analizadas con el fin de implementar un broadcast lo más eficiente posible. En las siguientes secciones de ruteo en VANETs se desarrollarán algunas de las soluciones planteadas y se presentaran otras propuestas por quienes desarrollaron los algoritmos.

4.3. Antecedentes en MANET

Entre los algoritmos de ruteo más conocidos en MANETs destacamos en particular AODV y DSR descriptos brevemente a continuación.

4.3.1. Automatic On-Demand Distance Vector Routing (AODV) [48][50]

Este algoritmo desarrollado en conjunto por Nokia, la Universidad de California y la de Cincinnati es del tipo reactivo, es decir, solo establece una ruta en caso de necesitarla. A diferencia de la mayoría de los protocolos de Internet en donde se definen las rutas independientemente del uso de los caminos (proactivos).

AODV periódicamente envía mensajes del tipo HELLO para descubrir sus vecinos manteniendo una tabla actualizada. Cuando requiere comunicarse con un nodo no vecino, realiza un broadcast de reconocimiento RREQ (*route request*). Otros nodos encaminan este mensaje y guardan la dirección de quien recibieron el RREQ, y en caso de que éste nodo sea vecino del destinatario o que ya tenga una ruta hacia el mismo, también envía hacia atrás un RREP (*route reply*) quedando entonces establecida la ruta.

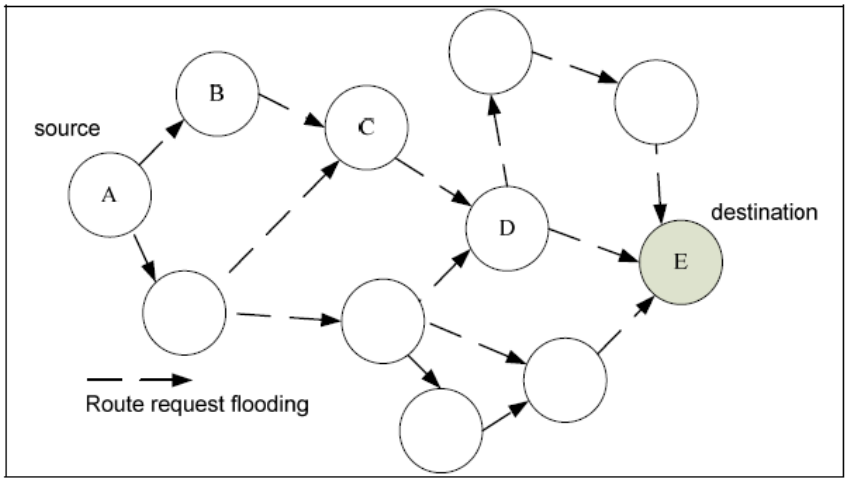

Figura 44 - AODV Route Request (RREQ)

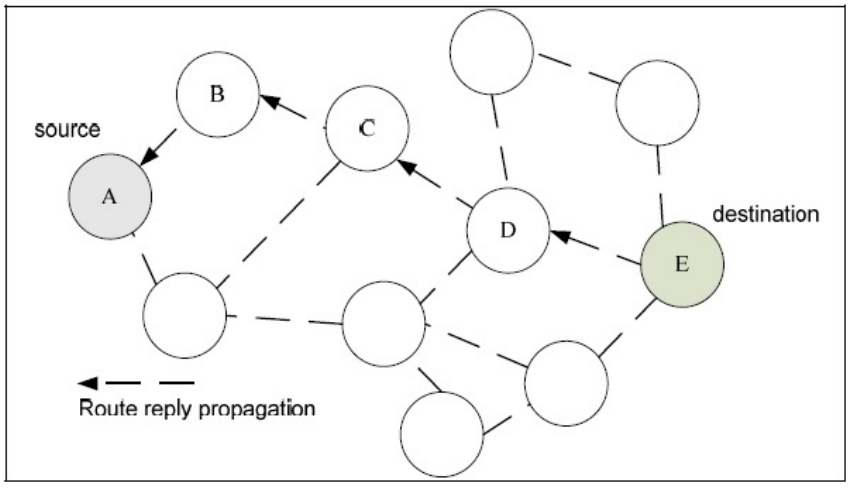

Figura 45 - AODV Route Reply (RREP)

Como modo de protección contra la inundación, los paquetes RREQ tienen un tiempo de vida limitado. En caso de que en un primer intento no se descubra una ruta, se re envía el RREQ con un tiempo de vida mayor. Para actualizar las rutas y borrar rutas viejas se utiliza el mensaje RERR en el cual se comunica a los vecinos estas variaciones por ejemplo cuando se pierde alcance de radio con un tercero.

A pesar de su amplio desarrollo para MANETs varios estudios [13] [51] [52] demuestran que estos algoritmos no funcionaran correctamente en VANETs. En [53] se evalúa AODV usando seis vehículos demostrando que AODV no puede encontrar, mantener y actualizar las rutas. Como ejemplo extremo se muestra que es casi imposible para TCP terminar el *three way handshake*. Por ende es necesario modificar estos algoritmos para poder superar los inconvenientes encontrados.

Mejorando AODV con métodos predictivos se crean PRAODV y PRAODV-M que utilizan la velocidad y ubicación del nodo para predecir la vida de una ruta. PRAODV establece una nueva ruta antes de que la que está en uso expire mientras que PRAODV-M elige la ruta con mayor tiempo de vida esperado en lugar de la que tenga menor número de saltos. A pesar de haber mejorado un poco el algoritmo original estos algoritmos aun dependen de una buena predicción, aumentan un poco el overhead y siguen careciendo del rendimiento esperado en una VANET.

4.3.2. Dynamic Source Routing (DSR) [11]

Diseñado para redes mesh este algoritmo también es reactivo pero utiliza source routing en lugar de depender de las tablas de rutas de cada nodo intermedio. Para lograr esto, cada nodo guarda la dirección de los demás nodos en el entorno al procesar los mensajes de descubrimiento. Al necesitar incluir las direcciones de todos los saltos en el paquete, para caminos largos el overhead puede ser importante especialmente en IPv6.

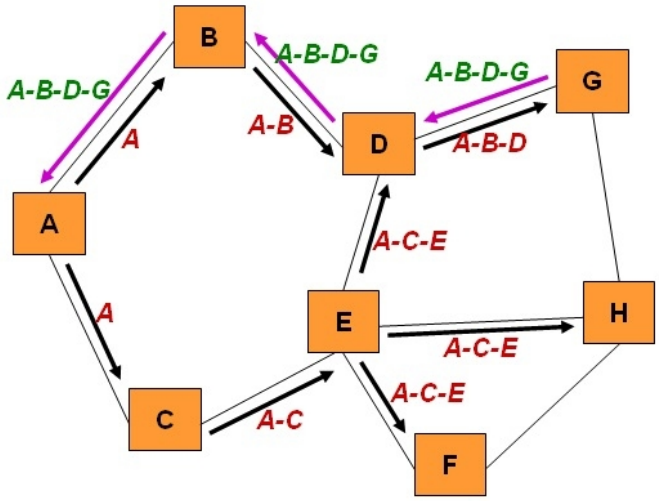

Figura 46 - DSR Route Discovery

Si no se quiere utilizar *source routing*, DSR incluye una opción que permite que todos los paquetes sean encaminados salto a salto. Toda la información de rutas es mantenida en cada nodo y actualizada constantemente.

Entre las principales diferencias de éstos dos protocolos mencionados, destacamos que DSR tiene acceso a mayor información de ruteo en comparación con AODV. Esto es porque en DSR en una única consulta de rutas, el nodo originador aprende la ruta a cada nodo intermedio y a su vez cada nodo intermedio aprende sobre los siguientes nodos de la ruta. En cambio en AODV estás funcionalidades no existen, lo que ocasiona que el algoritmo deba depender del descubrimiento de rutas más a menudo, generando más overhead en la red.

4.4. Ruteo basado en posición

Entendemos ruteo basado en posición a aquellos protocolos de ruteo que requieren de algún método de localización del nodo (como GPS) para funcionar. Comparaciones y estudios de performance apuntan a que en general el ruteo basado en posición es más eficiente cuando se trata de VANETs. En MANETs, Location Aided Routing (LAR) introduce el uso de la ubicación de los nodos, o al menos una aproximación de esta. LAR logra reducir el área de inundación a una más pequeña denominada request zone logrando entonces disminuir considerablemente la cantidad de mensajes de descubrimiento.

4.4.1. Greedy Perimeter Stateless Routing (GPSR) [11] [28]

Entre los algoritmos bajo esta categoría nos encontramos con GPSR. Este protocolo hace uso de dos algoritmos: *Greedy Forwarding y Perimeter Forwarding*.

Como su nombre lo describe, el “encaminamiento codicioso” busca enviar el paquete hacia el nodo que está más cerca del destino dentro de los que están en alcance de radio. Este proceso se repite hasta llegar a destino.

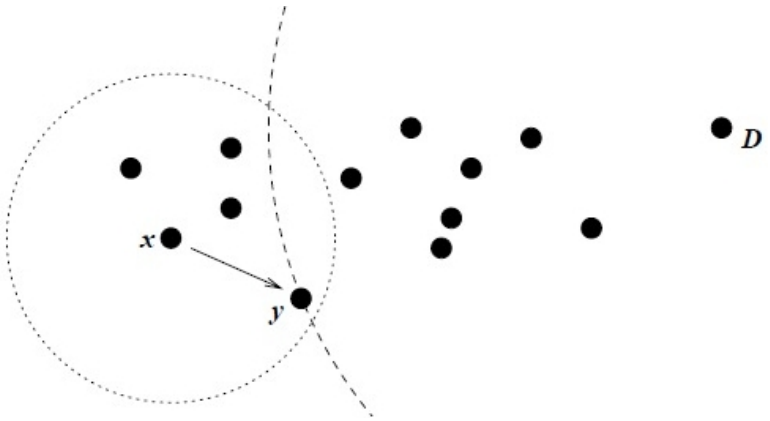

Figura 47 - Greedy Forwarding

Un beacon informa periódicamente la información de los nodos en alcance de radio a través de la dirección de broadcast de capa 2. La posición se transmite como un par de coordenadas (x,y) en punto flotante de 4 bytes por valor. Para evitar sincronización de los beacon de distintos nodos se deja un jitter de 50% en el intervalo del mismo.

Este comportamiento hace que el protocolo sea proactivo (al contrario de AODV y GSR) ya que sin que nadie realice un pedido, el nodo está continuamente enviando los beacon. Para minimizar el costo asociado al ocupar el canal enviando beacons y no datos, GPSR realiza *piggyback* con todos los paquetes de datos adjuntando la información de posición. Es necesario entonces que las interfaces de todos los nodos se encuentren en modo promiscuo

para poder recibir los datos de posición sin importar si el paquete de datos está dirigido a ellos. Esto es muy similar a lo realizado en nuestro prototipo como se puede ver en la sección de descripción del mismo. Se logra entonces reducir el tráfico de beacons en los lugares en donde se esté encaminando paquetes. El algoritmo podría adaptarse para ser completamente reactivo si en lugar de establecer un beacon periódico se crearan paquetes de pedido de posición y respuesta de posición.

Usar este algoritmo de ruteo en donde sólo es conocida la posición de los vecinos en alcance de radio lleva a que en ciertas topologías se encuentren problemas. Topologías como la de la figura 48 hacen que el algoritmo falle ya que no existe un nodo que esté más cerca del destino en comparación con el nodo originador. A pesar de que se observan dos rutas claras al destino, el algoritmo falla.

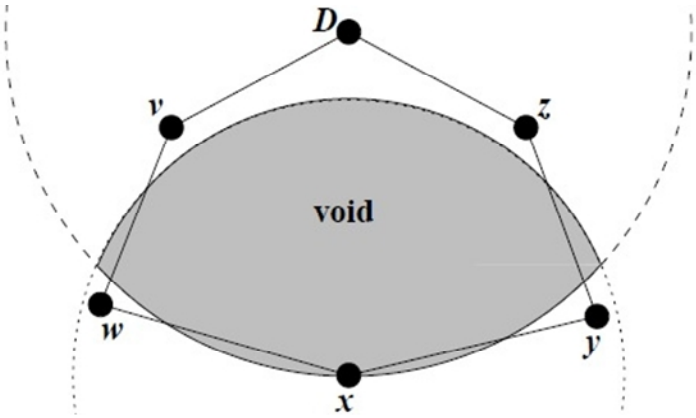

Figura 48 - Máximo local en greedy forwarding

Para resolver este problema se aplica la regla de la mano derecha o perimetral, con la cual para llegar al destino se circula por la izquierda del originador. Es decir se toma como una grafica plana y se recorre la misma utilizando la regla antes mencionada hasta que exista un nodo para hacer *greedy forwarding*. En el ejemplo los paquetes de ida y vuelta entre x y D recorrerían la siguiente secuencia: x **à** w **à** v **à** D **à** z **à** y **à** x.

GPSR sufre de un gran problema al enfrentarse a entornos urbanos, el sistema de *greedy forwarding* no funciona correctamente al haber muchos obstáculos como edificios y además la performance se ve degradada al cambiar entre encaminamiento codicioso y perimetral ya que se requiere una mayor distancia y número de saltos. Para este tipo de situaciones más complejas se han propuesto varias soluciones.

En [4] se realiza una comparación entre los protocolos AODV, DSR y GPSR. Como mencionamos al principio de esta sección es fundamental contar con un modelo de movilidad acertado ya que es este factor uno de los principales en afectar la performance de los distintos algoritmos de ruteo. Para este estudio se hace uso del simulador MITSIM [4] desarrollado en el Massachusetts Institute of Technology considerado de los mejores en el ámbito académico.

Se observa en la figura 49, la gran mejora de GPSR respecto a los protocolos estudiados anteriormente. En el caso del overhead esto se explica porque GPSR sólo depende de los beacon con sus vecinos directos y al simular la VANET con una cantidad fijas de nodos, el overhead por beacons es casi constante. En cambio en los algoritmos reactivos se observa un mayor overhead al aumentar la distancia ya que se requieren más paquetes de descubrimiento y mantenimiento de rutas que a distancias menores. Otro factor importante a destacar es el uso de *Greedy Forwarding* que influye directamente en la cantidad de saltos necesarios para llegar al destino.

Vale la pena recordar que para nuestro prototipo decidimos no implementar un algoritmo de ruteo como los mencionados aquí debido a la gran complejidad de los mismos y el tiempo acotado con el que contamos. Simplemente se agregó una funcionalidad de encaminamiento por broadcast y detección de paquetes repetidos utilizando *source routing*. Los estudios realizados sobre la performance se limitaron a la perdida de paquetes en alcance de radio logrando buenos resultados.

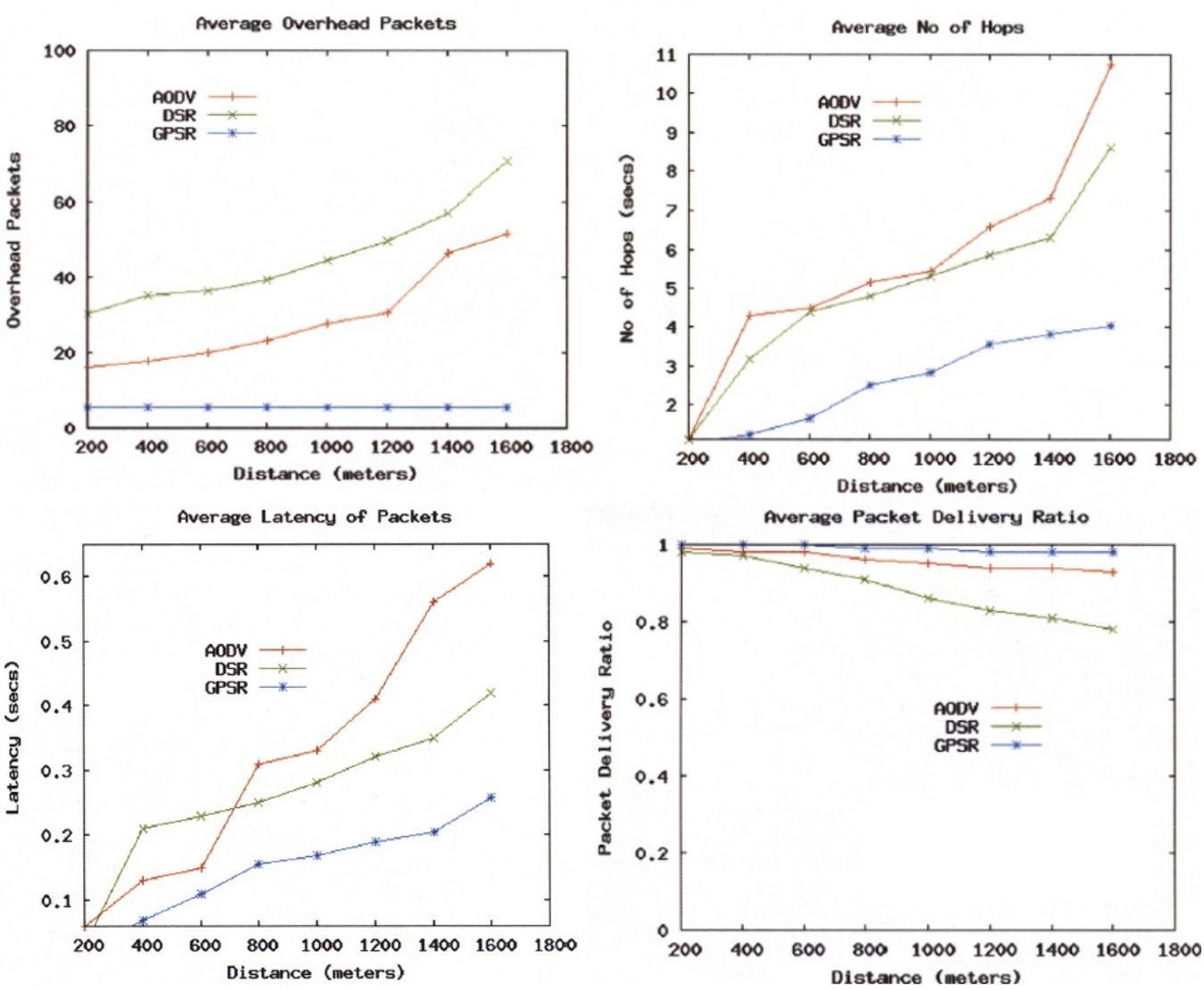

Figura 49 - Comparación de performance entre AODV, DSR y GPSR

4.5. Protocolos de ruteo orientados a entornos urbanos

Como mencionábamos anteriormente una importante cantidad de investigaciones se han centrado en optimizar los algoritmos de ruteo de VANETs ya existentes a entornos urbanos. Algunas utilizan ayuda de mapas asumiendo que el vehículo consta de un sistema de navegación mientras que otros por ejemplo aprovechan las rutas predefinidas de autobuses que son más estables. Entre los encontrados hacemos énfasis en GSR, GPCR y CAR.

4.5.1. Geographical Source Routing (GSR)

GSR, al igual que GPSR, utiliza un servicio de ubicación reactivo denominado RLS y además requiere contar con mapas de la ciudad. Si se quiere enviar paquetes de S a D, luego de haber obtenido la ubicación de D por medio del servicio mencionado, se calcula el camino más corto Dijkstra compuesto por una serie de intersecciones que el paquete debe recorrer. Luego se hace *greedy forwarding* entre esas intersecciones.
Este algoritmo intenta sobreponer el problema de GPSR en entornos urbanos. Al asegurarse que el paquete haga un salto en un vehículo ubicado en una intersección, la probabilidad de bloqueo por obstáculos es mucho menor que en GPSR.

La principal falla es que el algoritmo no tiene en cuenta si hay suficientes nodos entre dos intersecciones como para transmitir el paquete, algo que podría ser muy frecuente en redes de baja densidad de nodos. Además asumir que todos los vehículos cuentan con sistemas de navegación que incluyen mapas podría ser no del todo correcto, en particular en la etapa inicial del despliegue de una VANET.

4.5.2. Greedy Perimeter Coordinator Routing (GPCR) [30]

En GPCR se resuelve de manera similar pero se descarta el uso de mapas. También se requiere que el paquete sea encaminado a una intersección prioritariamente para evitar los obstáculos. Esta vez los nodos mandan periódicamente su posición y la de su tabla de vecinos. Se define un nodo como “coordinador” si este tiene 2 vecinos X e Y dentro de su rango de cobertura pero ellos no se listan como vecinos entre sí. Se calcula luego el coeficiente de correlación entre las posiciones de los vecinos. Valores de correlación cercanos a cero significan que no existe relación lineal entre las trayectorias de los vecinos. Entonces se puede asumir que el nodo se encuentra en una intersección y se define entonces a sí mismo como nodo coordinador. En este protocolo se utiliza *Restricted Greedy Forwarding* que se aplica de la siguiente manera: en caso de existir un nodo coordinador se encamina el paquete a éste, de lo contrario se utiliza *greedy forwarding* normal.

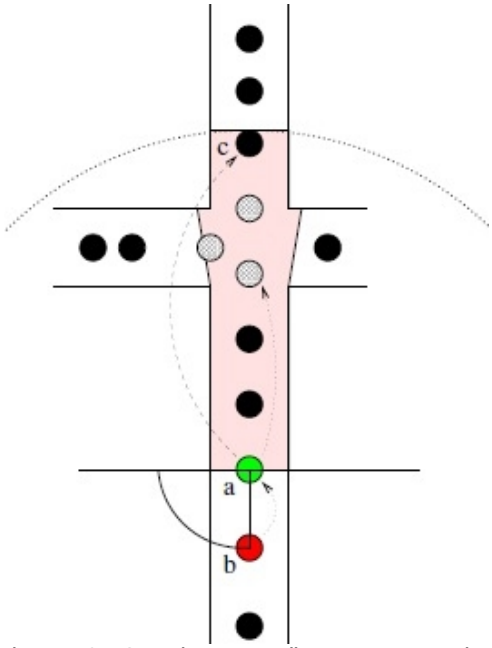

Figura 50 - Greedy Forwarding en una esquina

En la figura 50 se observa como *greedy forwarding* elegiría el nodo C, es decir el más cercano al destino dentro del alcance de radio (el destino no está incluido en la figura). En el caso de GPCR se prefiere que un nodo coordinador sea el siguiente salto ya que este decidirá por cuál de las dos intersecciones continuar.

El principal aporte de este protocolo es el poder discernir si un nodo se encuentra en una intersección sin contar con mapas. Este protocolo también presenta problemas como por ejemplo el hecho de que no se decide un camino óptimo en caso de contar con más de un coordinador (se elige aleatoriamente) lo que podría aumentar la cantidad de saltos y consecuentemente aumentar el retardo.

4.5.3. BUSNet y Anchor-based Street and Traffic Aware Routing [44]

BUSNet fue uno de los primeros proyectos en interpretar correctamente el problema que presentan los protocolos de ruteo al enfrentarse a entornos urbanos. En [39] se hace énfasis en que el problema radica en el uso de un modelo de movilidad incorrecto. Al comparar GPSR, AODV y DSR en la página anterior se están utilizando modelos de ubicación, rumbo y velocidad aleatorios, lo que sería correcto si los vehículos se movieran en cualquier dirección.

Pero la realidad es distinta, en particular en entornos urbanos. Los vehículos están limitados a moverse a lo largo de rutas predefinidas, limitados en la velocidad por los semáforos, etc. Este modelo hace muy difícil diseñar un algoritmo eficiente a la hora de enrutar un paquete. BUSNet propone entonces un nuevo modelo de movilidad denominado M-Grid demostrando que por ejemplo AODV, sufre una degradación de performance de un 20% al simular con misma cantidad de nodos utilizando el modelo M-Grid en comparación con el antes descripto, también conocido como *random waypoint.*

A partir de varias simulaciones, comparaciones de modelo y diferentes estudios realizados surge una nueva alternativa. Si bien los vehículos están restringidos a moverse dentro de las calles de una ciudad la aleatoriedad de sus orígenes y destinos hace difícil contar con una distribución suficiente de nodos a lo largo de todas las rutas como para poder enrutar paquetes correctamente. Esto es así al menos para la mayoría de los vehículos. Pero existe un grupo

especial que mantiene aproximadamente constante sus rutas a lo largo del tiempo: estos son los vehículos de transporte público como autobuses, tranvías, etc.

BUSNet propone crear algo similar a un backbone para la VANET basado en rutas fijas y confiables creadas por las rutas de autobuses. Sin embargo no propone detalles sobre algoritmos o protocolos y es aquí donde surge A-STAR (*Anchor-based Street and Traffic Aware Routing*). Similar a DSR y sumándole la idea de BUSNet, A-STAR crea el concepto de *"noción de calles"* para describir con mayor precisión el uso de mapas al calcular caminos a través de *"rutas ancladas"*. Se le asigna un peso a las rutas ancladas inversamente proporcionales a la cantidad de líneas de autobuses que circulan por las mismas. Se calcula entonces el camino óptimo aplicando Dijkstra a las rutas ancladas. Los caminos entre rutas ancladas se resuelven con *greedy forwarding*. También se agrega la funcionalidad de marcar una ruta como fuera de servicio por un tiempo determinado cuando ocurre el problema de máximo local.

A pesar de presentar una novedad muy interesante, debería estudiarse en mayor profundidad la posible congestión que se puede presentar en las rutas ancladas ya que la mayoría de los nodos elegirían estas rutas, incluso cuando rutas secundarias no ancladas formadas por nodos regulares pudieran tener muy buena conectividad.

4.5.4. Connectivity-Aware Routing (CAR) [34]

Por ser uno de los protocolos más recientes y más completos, CAR tendrá un mayor desarrollo que los anteriores. Basado en experiencias previas como AODV, CAR obtiene su motivación al intentar resolver varios dilemas planteados anteriormente en otros protocolos. La mayoría de los protocolos geográficos no tienen en cuenta si habrá conectividad a lo largo del camino elegido, por ejemplo el *greedy forwarding* solo mantiene información de las vecindades locales. Como un ejemplo se analiza el caso de la figura en que GPSR falla no una sino dos veces en encontrar el camino de S a D.

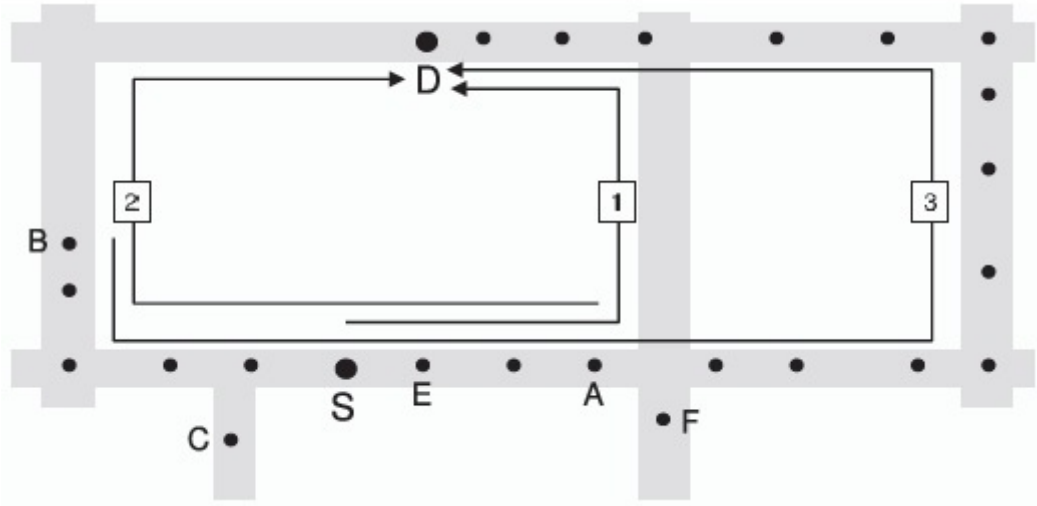

Figura 51 - Falla en la búsqueda de caminos

En primera instancia, luego de haber obtenido la ubicación por medio de algún servicio, el nodo S envía el paquete a E por *greedy forwarding*, pero E se encuentra con el problema de máximo local. Entonces se aplica la regla de la mano derecha y el paquete se dirige hacia B donde nuevamente se encuentra con un máximo local. Eventualmente el paquete encuentra su camino por la ruta 3 de la figura 51 pero los retardos evidentemente no son los deseados. Otros protocolos como A-STAR o GSR calculan la ruta por medio de Dijkstra pero nuevamente

sin conocer si la ruta elegida tiene suficientes nodos como para que el paquete llegue a destino como el camino 1 y 2.

El protocolo CAR consta de 4 pasos:

- Ubicación del nodo y descubrimiento del camino
- Enrutamiento del paquete a lo largo del camino elegido
- Mantenimiento del camino
- Recuperación de errores

Al ser un protocolo proactivo, CAR envía periódicamente mensajes beacon HELLO con la información de posición y velocidad. Al recibir un beacon se agrega en la tabla de vecinos la información recibida y se realiza una estimación del tiempo de vida del enlace punto a punto comparando los vectores velocidad y las posiciones actuales de cada nodo.

Se introduce un método novedoso para la estimación del tiempo de vida de un enlace entre dos nodos. Comparando los vectores velocidad y las posiciones recibidas con la suya propia, un nodo puede estimar cuánto tiempo durará el enlace. El enlace dejará de ser válido cuando se alcance la distancia R (*range*), parámetro configurable que se fija inicialmente en el 80% del alcance promedio de un nodo. Además el algoritmo tiene en cuenta la influencia que puede tener la cantidad de vecinos de un nodo en una comunicación que lo tenga como salto intermedio. En la figura 52 se hace referencia a esta situación.

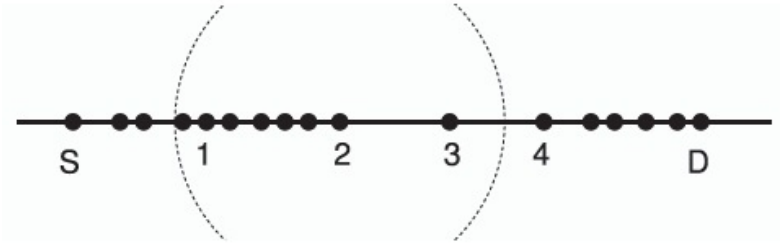

Figura 52 - Relevancia de la densidad de nodos

Es evidente que la información de la tabla de vecinos del nodo 3 en la figura 52 es mucho más importante que la de los primeros nodos ya que existe mayor probabilidad de desconexión en él. En cambio cualquier nodo de los próximos a 1 podría enrutar el paquete. A esto se refiere el protocolo con *"noción de conectividad"*. Para resolver este problema, se hace que la frecuencia con la que se envía un beacon se adapte de acuerdo a la cantidad de vecinos que éste tiene en la tabla. Cuantos menos vecinos tenga, más beacons son enviados. A su vez este mecanismo logra una baja en la carga de la red especialmente en donde existe una alta densidad de nodos y la cantidad de beacons enviados podría ser demasiado alta.

A diferencia de los protocoles anteriores, CAR incluye un sistema de descubrimiento de la ubicación del destino y en el mismo proceso crea la ruta hacia el mismo. Ésta es una adaptación de *Preferred Group Broadcast* (PGB [49]). Un nodo en busca de un destino inicia un PGB *path discovery*. Los nodos que reciben este paquete especial agregan los datos a una nueva tabla llamada *Received-Path-Discoveries-Table* las entradas que están en la tabla no se repiten, éstas entradas expiran a los 60 segundos. De manera similar a GPCR, cuando un nodo contiene vecinos con direcciones no paralelas (vectores velocidad con ángulo mayor al parámetro Θ) se considera que se encuentra en una intersección. Dicho nodo agrega un "ancla" al paquete si su dirección no es paralela a la del nodo anterior. El nodo destino tiene ahora entonces un camino completo hacia el originador guardado como una serie de puntos de anclaje y elige el mejor camino de los recibidos. Para enrutar la respuesta al pedido de ruta, el destino utiliza *Advanced Greedy Forwarding* (AGF) entre los puntos de anclajes registrados.

Como mencionamos previamente, CAR tiene la capacidad de realizar mantenimiento de los caminos. Este proceso se realiza por medio de la definición de zonas de guardia, las cuales pueden ser fijas o móviles. Por ejemplo si el nodo destino cambia de dirección al doblar en una esquina, define una zona de guardia fija en ese punto. Esta zona de guardia es mantenida por cualquier nodo que se encuentre en la misma (definida como coordenada del centro y radio) agregando la información relevante a su propio mensaje HELLO. Cuando un nodo en la zona de guardia recibe un paquete para el destino, agrega un nuevo punto de anclaje, actualiza la posición y vector velocidad del destino en el paquete extendiendo el camino existente sin necesidad de hacer un nuevo descubrimiento de camino.

Otro ejemplo es en el caso que el nodo destino se mueve originalmente hacia donde vienen los paquetes y por algún motivo cambia de dirección. En este caso se crea una guarda móvil que sigue la trayectoria que el nodo tenía y contiene información de la nueva trayectoria y ubicación. Entonces cualquier nodo *"guardián"* se encargará de corregir el camino y mantener viva la guarda mientras sea necesaria.

Se podrían crear huecos en el camino a pesar de los métodos anteriores, para corregir estos errores se implementan dos sistemas. En caso de que a mitad de la ruta un nodo detecta un hueco hacia delante (sentido de viaje del paquete) comienza a almacenar la información actuando como buffer o destino temporal. Inmediatamente envía un mensaje pidiendo un próximo salto, su ubicación y el siguiente punto de anclaje. En caso de existir un nodo la comunicación se re establece hacia el destino. De lo contrario el nodo sigue almacenando paquetes hasta encontrar un próximo salto o hasta que ocurra un timeout. Luego del timeout el nodo en cuestión realiza una búsqueda de camino propia hacia el destino y avisa al nodo originador de la situación. En caso de encontrar la nueva ruta se envía al nodo originador la concatenación de los caminos, de lo contrario puede descartar los paquetes almacenados o reenviarlos al originador. El originador al recibir la nueva posición del destino puede saber si éste está más cerca que antes y por lo tanto el camino se puede optimizar, entonces realiza un nuevo *route discovery*. Se denomina *Walk-around error recovery* a este método.

Se realiza el estudio de performance de este nuevo protocolo comparando GPSR, GPSR con AGF, CAR y CAR+WA en ns2 tanto para escenarios de autopista como ciudad variando la densidad de nodos por kilómetro de calle/autopista. Se elige la banda de 2.4 GHz a 2 Mbps utilizando el modelo *Shadowing* de propagación que tiene en cuenta efectos de propagación por caminos múltiples y calcula estadísticamente la potencia de recepción, no como una función determinística dependiente de la distancia. A continuación los resultados:

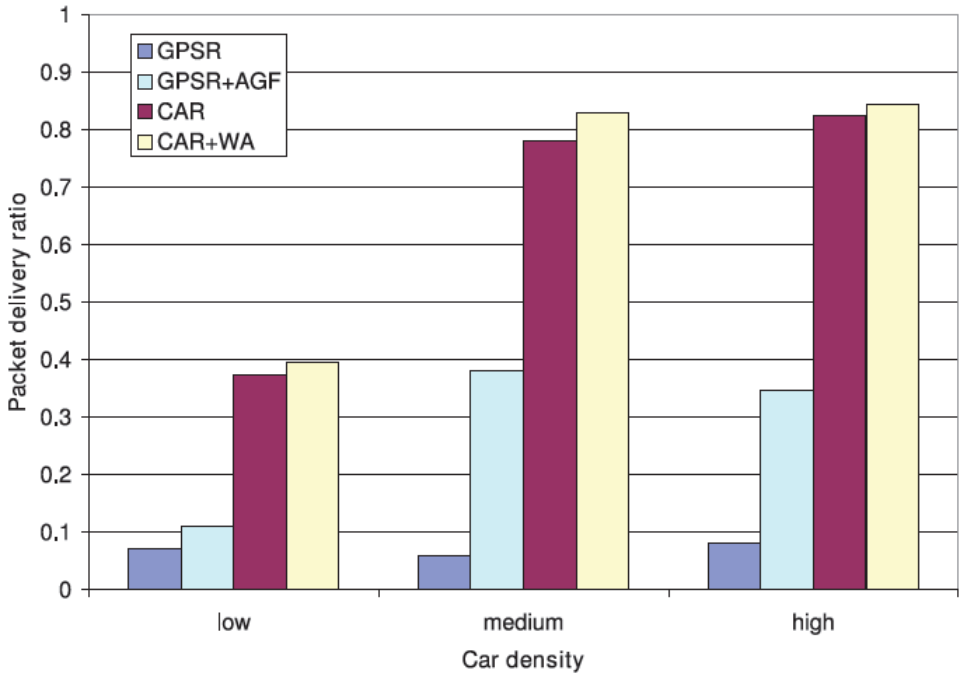

Figura 53 - GPSR vs. CAR - Entrega de paquetes

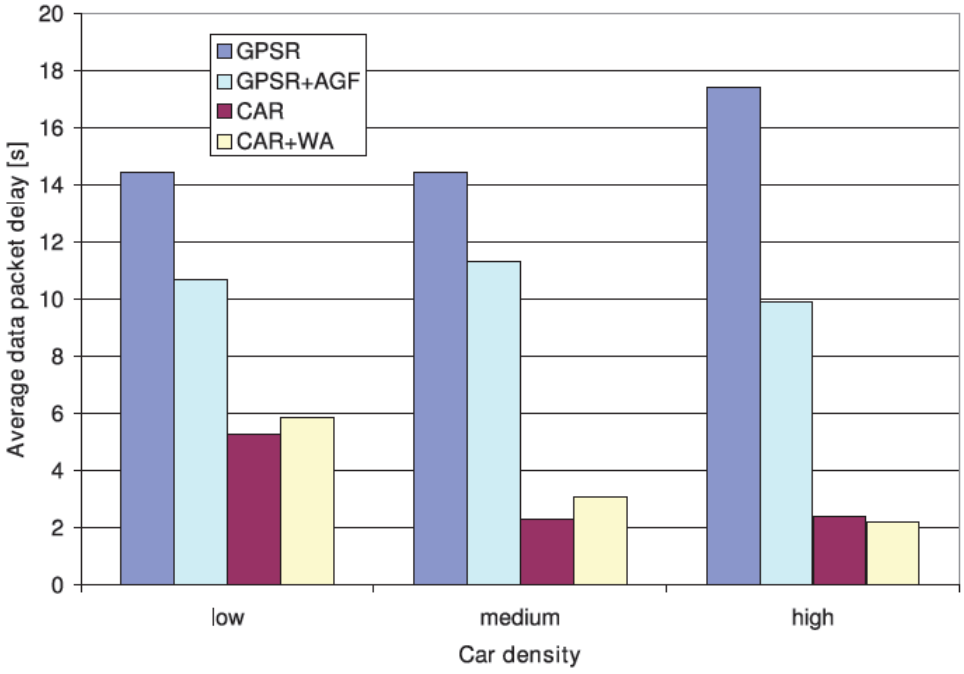

Figura 54 - GPSR vs. CAR - Retardo promedio

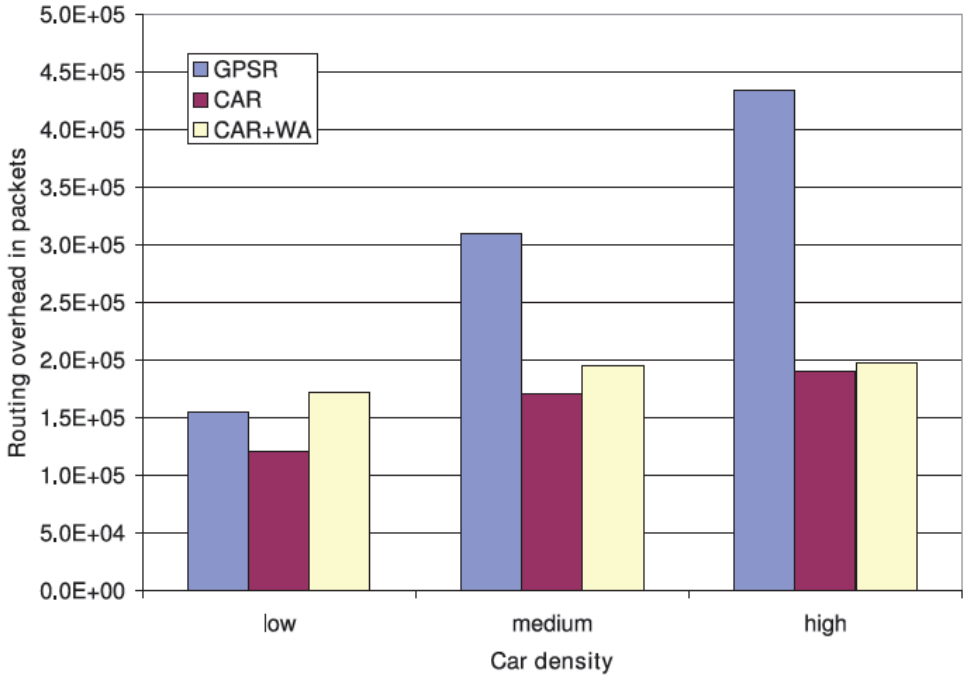

Figura 55 - GPSR vs. CAR - Overhead de ruteo

4.6. Conclusiones de la sección

De lo estudiado anteriormente se puede concluir que definitivamente los algoritmos propuestos para MANETs no son eficientes a tal punto que algunos ni siquiera funcionan en redes VANET. La razón es que dichos algoritmos no toman en cuenta la alta velocidad de los vehículos en comparación con dispositivos móviles y no resuelven correctamente la ruptura de rutas en este caso.

Además se observa que existen gran cantidad de protocolos de ruteo propuestos para VANETs pero ninguno aún ha probado ser efectivo en una red de gran tamaño o en diversas situaciones como entornos urbanos y rurales. Si bien algunos son mejores que otros resolviendo problemas puntuales no se ha llegado a la solución global y escalable como para implementar finalmente en el mercado.

Sí se observan patrones claros a los que tendría que apuntar la solución definitiva. Entre ellos la rápida recuperación ante situaciones de bloqueo de rutas, minimizar la congestión agregada a la red por el protocolo pero más especifico aún, el uso de posicionamiento como herramienta fundamental al encaminar un paquete y el aprovechar la capacidad de procesamiento y almacenamiento local de cada nodo al máximo.

En nuestro prototipo no planteamos revolucionar las redes VANET con un nuevo protocolo de ruteo sino implementar uno sencillo que sin tener en cuenta la congestión y optimización de caminos logre conectividad multi salto.

5. Experiencias prácticas

Ante el objetivo de construir un prototipo de sistema de comunicación entre vehículos, y luego de realizar un estudio teórico general sobre las distintas tecnologías inalámbricas existentes comenzamos a realizar pruebas prácticas. Del mencionado estudio se desprendió que si bien varias de las tecnologías tienen algunas fortalezas para el cometido de este proyecto, 802.11 es la más indicada. A la hora de comenzar con la práctica optamos por basar nuestras pruebas en sistemas 802.11 por varias razones:

- La popularidad de los sistemas WiFi de hoy en día hace que las placas 802.11 sean más accesibles que otros sistemas de radio.
- Contamos con estas placas en las notebooks, por lo que desde el comienzo ya contábamos con varios sistemas para probar.
- Por tratarse de sistemas que operan en bandas ISM de uso libre, no requerimos de tramitar permisos en URSEC u obtener la licencia de radioaficionado.
- La combinación frecuencias y potencias utilizadas determina un alcance de radio intermedio para las distancias y velocidad que están involucradas.
- Linux tiene incorporado los drivers de estas placas en el kernel, y con el paquete wireless-tools se puede –al menos intentar- tener un control total sobre las funciones de la placa.
- Car-2-Car así como las VANETs, parten su desarrollo de 802.11 por lo que es interesante trabajar en el mismo sentido que los grupos de discusión de la actualidad.

5.1. Objetivos de las pruebas

El objetivo de las pruebas era de estudiar la viabilidad de utilizar un sistema 802.11 comercial para construir una red entre vehículos, independiente de nodos centrales de control y resistente a los cambios rápidos de topología.

5.2. Primer camino – modo Ad-Hoc

Todas las placas 802.11 en la actualidad pueden operar en dos modos distintos: *managed* y *ad-hoc*.

El modo *managed* es el que la placa utiliza cuando accede a un *access point*, un nodo central que controla la comunicación entre los distintos nodos de la red, y donde toda comunicación entre dos nodos cualesquiera de la red pasa a través del mismo. Claramente, este modo de operación no resulta apropiado para nuestros objetivos.

El modo *ad-hoc*, en cambio, permite a dos placas 802.11 cualesquiera hablar entre sí sin necesidad de contar con un nodo central que actúe de puente entre las dos. Mientras toda la bibliografía, así como su definición hablan de este modo como una forma de comunicación entre 2 palcas, en principio nada indicaba que no fuera compatible para la comunicación entre más de 2 placas. Carentes de cualquier referencia sobre la viabilidad de esta propuesta, nuestras pruebas se enfocaron en estudiar esta posibilidad.

Utilizando el paquete *wireless-tools*, Linux permite configurar las placas para realizar estas pruebas. Con el programa *iwconfig* podemos configurar el modo de operación de la placa a ad-hoc, así como definir el canal, ESSID, identificador de celda, potencia de transmisión, bitrate,

etc. En principio todo indicaba que si varias placas se configuraban en modo ad-hoc con los mismos parámetros de canal y ESSID, entonces podrían comunicarse indistintamente entre ellas sin necesidad de un nodo central.

5.3. Problemas del modo ad-hoc

Ya desde el comienzo encontramos problemas para fijar estos parámetros. Encontramos que la respuesta de la placa a los comandos de *iwconfig* tenía una fuerte dependencia con la placa en sí misma (distintos fabricantes), drivers de la placa y hasta distribuciones de Linux. Hicimos pruebas con Xubuntu 9.4, Fedora 10, Fedora 11 y finalmente con Ubuntu 9.10. Sucedía que la placa no respondía al comando, o que poco después de fijado un parámetro, éste se cambiaba solo. También cualquier tipo de administrador de redes, como el *NetworkManager*, generaba problemas al intentar configurar las placas en forma automática para unirse a los access points de la zona.

El problema más importante lo tuvimos con el identificador de celda, un parámetro de 6 bytes que identifica la red inalámbrica. Dos placas que estuvieran configuradas en el mismo canal y con el mismo ESSID, pero cuyos identificadores de celda fueran diferentes, no iban a poder comunicarse entre sí. *Iwconfig* permite configurar este parámetro a través de la opción *ap*, pero salvo en Ubuntu 9.10, las placas no respondían a este comando, o rápidamente cambiaban solas de identificador de celda. Vimos que este parámetro se negocia automáticamente a nivel de hardware entre las placas al establecer la red ad-hoc, no habiendo ningún problema para el establecimiento de una red entre 2 nodos. Sin embargo, al aumentar la cantidad de nodos, era muy difícil que todas se auto-configuraran al mismo identificador de celda. Si esto sucedía, entonces lográbamos la comunicación entre todos los nodos sin ningún problema. Pero este parámetro se podía cambiar repentinamente en cualquiera de las placas, por ejemplo al entrar otro nodo a la red.

Por tratarse de un parámetro negociado a nivel de hardware, era poco lo que podíamos hacer para estabilizarlo. Sin embargo, al cambiar a Ubuntu 9.10, este parámetro quedaba fijo al configurarlo con *iwconfig ap*. Teníamos ahora una manera de configurar todas las placas en forma idéntica.

La distribución Ubuntu 9.10 nos permitió tener un control efectivo sobre los parámetros configurables de las placas. Un ejemplo de configuración podía ser ejecutar el siguiente comando sobre las placas: *iwconfig wlan0 mode ad-hoc channel 6 essid vadhoc ap 11:11:11:11:11:11 rate 1M*. Al ejecutar un comando de este tipo en todas las placas, estábamos configurando idénticamente todos los parámetros accesibles de las mismas en forma idéntica. Sin embargo, aún así obtuvimos un resultado muy inestable de la red ad-hoc. Uno de los casos resultados típicos era de tener 2 nodos conectados perfectamente, y al entrar un tercer nodo en la red, la comunicación entre los primeros dos se perdía, aun manteniendo todos la misma configuración.

Como mencionamos anteriormente, no existen referencias a redes ad-hoc de más de 2 nodos más que alguna breve referencia a su inestabilidad. Pasamos entonces a estudiar más en profundidad lo que estaba sucediendo y encontramos que las placas efectúan muchas funciones a nivel de hardware que no parecen estar estandarizadas para el modo ad-hoc. De hecho, en la norma 802.11 no se especifica 100% el modo ad-hoc.

Utilizando una placa en modo monitor, pudimos observar como los distintos fabricantes implementan el modo ad-hoc de diferentes formas. Las placas Intel, emiten *beacons* cada 100

ms tal como lo hace un access point, mientras que las placas Broadcom se descubren con *probe request*.

Otro fenómeno es que aún efectuando toda la configuración en forma manual, existen otros requerimientos que resuelve la placa en forma automática y a nivel de hardware que determinan el éxito de la comunicación. No basta con configurar todos los parámetros de la placa para que esta comience a comunicar los paquetes que se le envía. Observamos que las placas no transmiten los paquetes al aire hasta no sincronizar la ad-hoc por sí misma a nivel de hardware. Se podía observar en la PC como los paquetes eran enviados a la placa por parte del sistema operativo, pero con una placa independiente en modo monitor no se observaba que los mismos fueran transmitidos al aire. Típicamente, cuando esto sucedía, se podía observar un comportamiento extraño en los mensajes de *beacon* y *probe request*, dejando en evidencia que la placa aún estaba solicitando mayor información a la red.

Revisamos el código de los drivers de una de las placas (la Intel que tienen drivers open source), y pudimos observar que efectivamente la placa busca identificar en la red, si ella es el primer nodo encendido o si existe otro nodo buscando pareja. Al tener más de 2 nodos en el aire, estos algoritmos empiezan a fallar haciendo que la placa no encuentre un nodo con quien hacer pareja, o distorsionando la red que ya estaba formada entre otras dos placas.

Este último fenómeno es el que nos hizo abandonar el camino del modo ad-hoc, pues entendimos que para seguir intentando estabilizar la red, el paso siguiente era modificar los drivers de las placas sin que esto garantizara el éxito de la comunicación.

5.4. Pruebas de movilidad en modo ad-hoc

Cuando optamos por comenzar el estudio por el modo ad-hoc, pensamos que el overhead del mismo iba a simplificar y robustecer la creación de redes entre varios miembros. De la misma manera, comenzamos nuestras pruebas de transmisión de datos utilizando el protocolo UDP, con direcciones IP estáticas y transmitiendo todos los mensajes a través de la dirección de broadcast de la subred. Escribimos un simple programa que verificaba la comunicación entre dos nodos enviando un paquete a intervalos regulares a través de la dirección de broadcast. Con este programa realizamos nuestras primeras pruebas de movilidad estudiando el comportamiento de dos nodos en modo ad-hoc cuando uno de ellos salía y entraba de la zona de alcance de radio en forma reiterada.

El resultado de las pruebas fue sorprendentemente satisfactorio, donde la configuración de las placas se mantenía aún cuando las mismas estaban fuera de alcance y ni bien quedaban nuevamente en alcance de radio, los paquetes comenzaban a llegar sin problemas de un nodo a otro. Utilizando 15 dBm de potencia de transmisión obtuvimos unos 50 mts de alcance en un entorno urbano. Lamentablemente, al no poder extender la red a más de 2 nodos en forma estable, no pudimos estudiar el comportamiento de ésta cuando la movilidad y problemas de alcance de radio se daban entre varios nodos a la vez.

5.5. Pruebas realizadas por el grupo NOW

El grupo de trabajo alemán Network On Wheels, es parte del consorcio Car-2-Car que desarrollaremos más adelante en el documento. Este grupo tiene por cometido desarrollar un protocolo de comunicación entre vehículos basándose en el estándar 802.11.

Una parte interesante de la documentación liberada por este grupo son las pruebas realizadas sobre 802.11 en ambientes vehiculares. Para ello equiparon dos vehículos con placas inalámbricas 802.11a, 802.11b y 802.11g y realizaron pruebas de latencia y alcance en distintos escenarios, manteniendo entre los vehículos una conexión en modo ad-hoc.

Al medir el alcance en campo abierto, encontraron que las placas se comportaban de manera formidable, logrando más de 1 km de alcance.

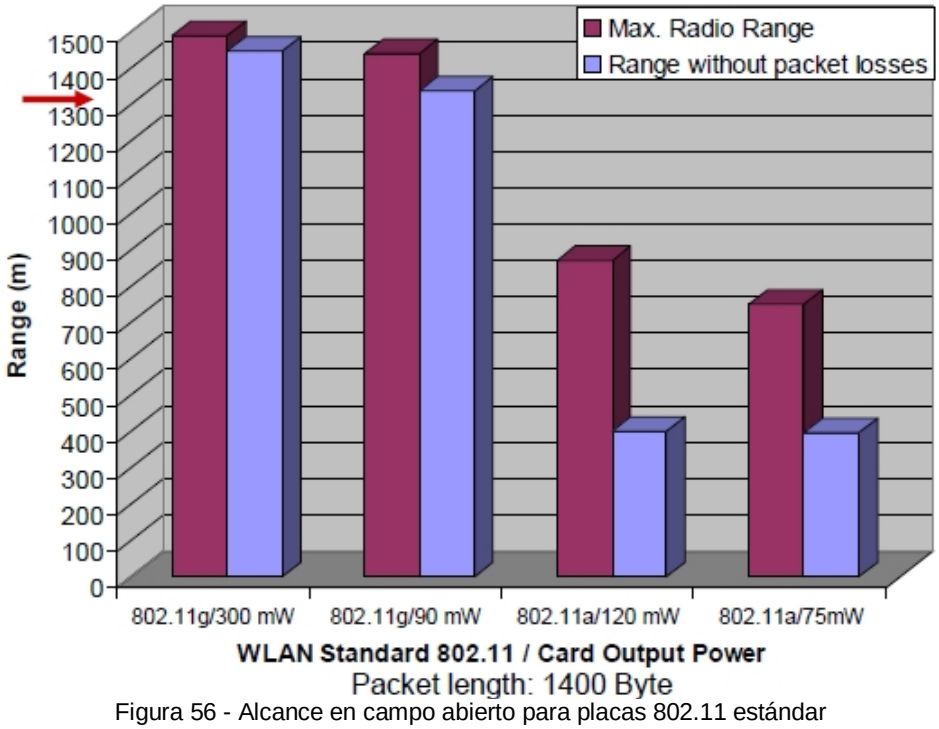

Figura 56 - Alcance en campo abierto para placas 802.11 estándar

Sin embargo, al realizar estas pruebas sobre una ruta transitada, encontraron que simplemente el obstáculo causado por 2 camiones entre los vehículos era suficiente para que se perdiera el enlace.

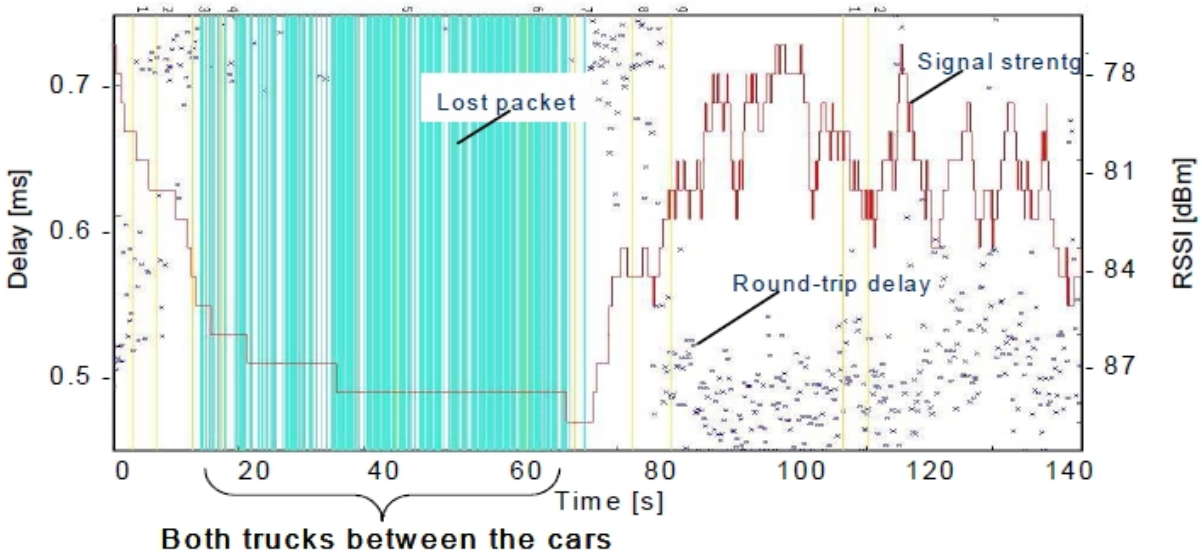

Figura 57 - Pérdida de enlace con dos camiones entre los nodos

Finalmente, uno de los resultados más interesantes es arrojado por las pruebas de conectividad con movimiento relativo entre los vehículos. Se realizaron varias pasadas a distintas velocidades, midiendo con el comando ping, el tiempo durante el cual los dos vehículos estaban efectivamente asociados.

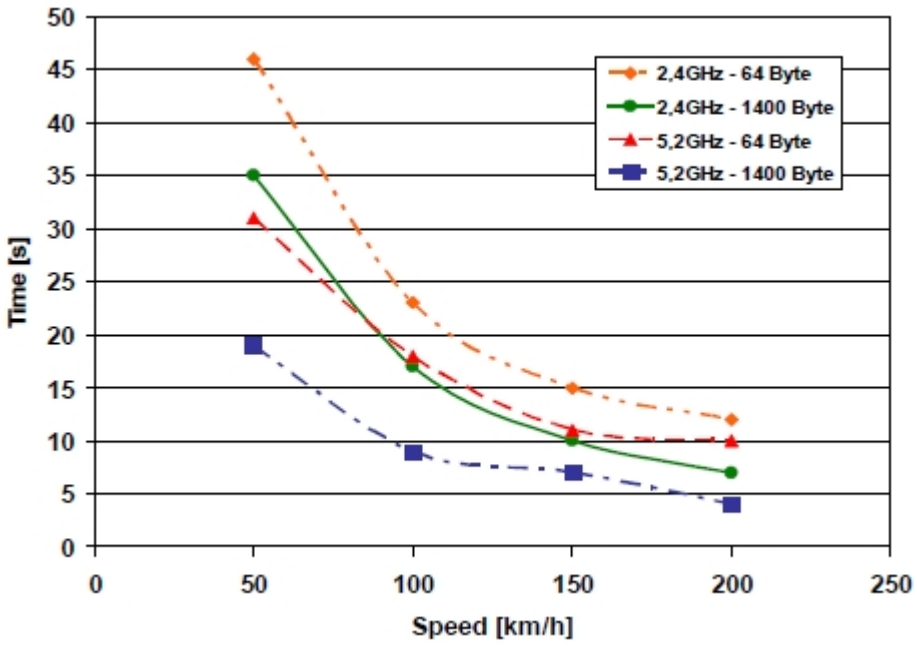

Figura 58 - Tiempo de conexión vs. velocidad relativa entre los nodos

Vemos en la figura 58 que a medida que se aumenta la velocidad relativa entre los vehículos, el tiempo durante el cual se mantiene la conectividad es menor. Sin embargo, la relación con la que baja este tiempo, condice con que al ir más rápido, toma menos tiempo cubrir la distancia de cobertura de las placas inalámbricas. Se puede decir entonces que la velocidad no influye sobre área de cobertura.

5.6. Segundo camino – Inyección de paquetes en modo Monitor

Anteriormente mencionamos al modo Monitor de las placas 802.11, como un modo de operación separado del *managed* y *ad-hoc*, que nos permite escuchar el tráfico de la red en forma pasiva y sin procesar. Este modo de operación permite ver el tráfico de señalización de hardware como lo son los *beacons* o los *probe requests*. En operación habitual, estas tramas son filtradas por la placa antes de pasar al sistema operativo y por lo tanto no son visibles aún utilizando analizadores de redes en modo promiscuo.

Por lo tanto, el modo monitor nos brinda la posibilidad de utilizar las placas 802.11 en una forma más cruda y sin intervención del hardware en el tráfico de la red. Es importante aclarar que no todas las placas de red, ni todos los sistemas operativos cuentan con el modo monitor, ya que no es un modo definido en la norma 802.11.

Como mencionamos, el modo monitor está pensado para realizar una escucha pasiva del tráfico de la red y por lo tanto no es posible asociarse a otros nodos cuando utilizamos este modo. Menos aún podemos enviar tramas, pues al estar inactivos los servicios de capa de red, no le son agregados los encabezados apropiados.

Sin embargo, existen algunas referencias donde se prueba la posibilidad de enviar tramas aún estando en modo monitor. En particular, la biblioteca *libpcap* permite inyectar paquetes crudos en cualquier interfaz de red. De la misma forma, permite leer los paquetes crudos que llegan a la interfaz. De aquí surge la idea de aprovechar el modo monitor de las placas, con sus servicios inactivos, para inyectar en la red paquetes arbitrarios creados por el usuario. Esto nos independiza de cualquier tipo de asociación con otros nodos ya que los paquetes son enviados al aire en forma arbitraria. De la misma forma, podemos escuchar los paquetes que están en el aire, en el mismo canal, independientemente de los parámetros de ESSID o identificador de celda.

En contraste con el modo ad-hoc, donde pensábamos utilizar el protocolo de asociación de la norma así como UDP a nuestro favor, en modo monitor el paquete debe ser formateado por nosotros en todas sus capas. En este modo, no se agregarán encabezados extras pues los servicios están todos inactivos. Si bien esto genera más trabajo a la hora de programar los servicios de comunicación, también nos da otra libertad para formatear los paquetes en forma más eficiente y apropiada para nuestra aplicación. Podemos quitar encabezados que no sean interesantes y así reducir el overhead del protocolo, así como liberar ancho de banda al tener inactivos todos los servicios de asociación como los *beacons* y *probe requests* que tantos problemas nos causaron.

Los programas escritos para comunicar los nodos a través de este sistema son a más bajo nivel que en el modo ad-hoc. Aquí debemos hablar directamente con la placa utilizando las herramientas disponibles en la biblioteca *libpcap*, haciendo previamente el formateo de toda la trama que sale al aire. En el modo ad-hoc, simplemente se debía hablar con los servicios de transporte como UDP y el resto era resuelto por el sistema operativo. De la misma forma, no podemos utilizar programas estándar de TCP/IP pues estos servicios no están disponibles con la placa en modo monitor.

Los programas que realicen comunicación a través de la inyección de paquetes deben manejar todo el stack de protocolos a ser utilizados al enviar las tramas. En la recepción, al estar la placa en modo monitor, esta escucha todo el tráfico del canal sin filtrar nada por direcciones MAC, ESSID, identificador de celda, etc, como habitualmente sucede en los modos de operación normal. Afortunadamente la biblioteca *libpcap* nos brinda herramientas para crear filtros de captura a las tramas que llegan a la placa y de esta forma librarnos del tráfico indeseado e incontrolable por tratarse de bandas ISM libres.

5.7. Biblioteca Libpcap

De la misma forma que el protocolo RadioTap cobró un gran protagonismo durante el desarrollo de este proyecto, la biblioteca *libpcap* fue una de las herramientas fundamentales tanto durante el estudio previo del comportamiento de los distintos modos de operación de los dispositivos 802.11, como en la implementación del prototipo.

Originalmente desarrollado por los mismos desarrolladores de tcpdump en el Network Research Group at Lawrence Berkeley Laboratory, Libpcap es una biblioteca open source escrita en C que provee una interfaz de alto nivel con la cual obtener tramas directamente del driver de la tarjeta de red. Mediante este mecanismo se pueden extraer todas las tramas en el medio compartido, aún aquellos que no tienen nuestra dirección mac como destino. Además, Libpcap es perfectamente portable entre un gran número de sistemas operativos.

Con Libpcap es posible escribir paquetes capturados a un archivo o leer paquetes desde un archivo conteniendo paquetes guardados. De esta manera se pueden desarrollar aplicaciones capaces, no sólo de capturar, sino analizar los paquetes capturados, así como también levantar paquetes guardados en un archivo y analizarlos utilizando el mismo código de análisis. Tal es el caso de tcpdump y wireshark, aplicaciones de análisis de tráfico muy difundidas en el mundo Linux.

Más allá de la complejidad del programa de captura que se desee implementar, todo programa debe seguir el siguiente esquema básico:

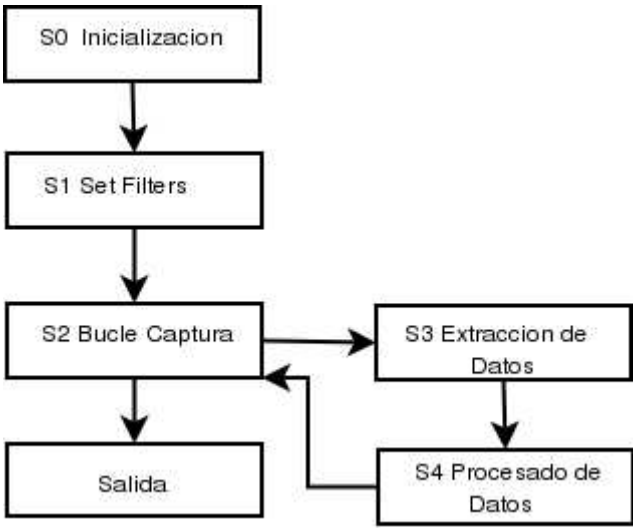

Figura 59 - Diagrama de bloques de la captura de tramas

Inicialización

En la etapa de inicialización se obtiene información relevante del sistema como ser interfaces de red instaladas, configuración de estas interfaces (máscara de red, dirección de red), etc. Para esto se realizan llamadas a diferentes funciones de la biblioteca que se encargan de obtener esta información proveniente de capas inferiores del sistema.

Algunas de estas funciones son:

- pcap_lookupdev: devuelve un puntero al primer dispositivo de red válido para ser abierto para capturar paquetes
- pcap_lookupnet: devuelve la dirección de red con su respectiva máscara para una interfaz de red válida conocida
- pcap_findalldevs: devuelve todas las interfaces de red que puedan ser abiertas para capturar datos
- pcap_datalink: devuelve el tipo de enlace de datos asociado a una interfaz de red

Filtros de captura

Luego de la inicialización viene una segunda etapa fundamental que es el proceso de filtrado de paquetes. Libpcap es una biblioteca de funciones y los procesos que ejecutan estas funciones lo hacen a nivel de usuario. Sin embargo la captura real de datos tiene lugar en las capas más bajas del sistema operativo, en el área denominada kernel del s.o.

Dado que es muy costoso, en términos de recursos del s.o., pasar de un nivel a otro por cada paquete capturado en la red, se utilizan filtros de captura, para poder pasar al nivel de usuario sólo aquellos paquetes que nos interesa analizar, además de hacerlo de una manera segura, ya que cualquier fallo en capas tan profundas degradará la performance de todo el sistema.

No existe un único sistema de filtrado, sino que cada s.o. reescribe su propia solución. Uno de los más extendidos, y en la cual está basada la implementación para LSF (Linux Socket Filter), es el BPF (Berkeley Packet Filter) originalmente utilizado en sistemas BSD.

A continuación se muestra el esquema general de funcionamiento de BPF dentro del sistema:

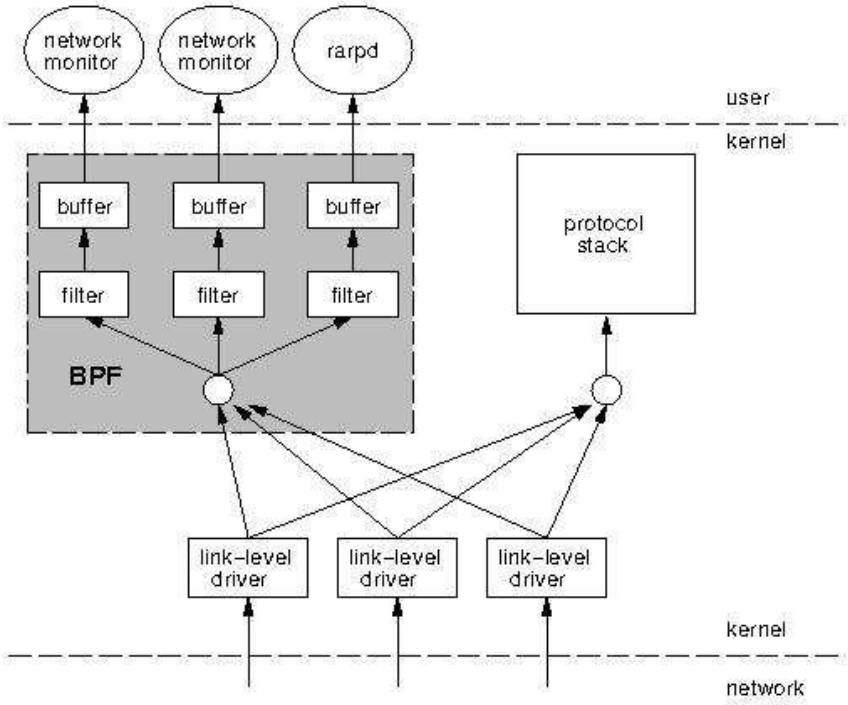

Figura 60 - Berkeley Packet Filter

En funcionamiento normal, cada vez que llega una trama la placa de red verifica que esté dirigido hacia la propia maquina, y de ser así lo pasa hacia arriba en la pila de protocolos para ser procesado por capas superiores. En caso contrario, lo descarta.

Estando el BPF activado, una copia de los paquetes entrantes es primero procesada por éste. BPF será el encargado de comparar el paquete con cada uno de los filtros establecidos, pasando una copia a cada uno de los buffers de las aplicaciones cuyo filtro se ajuste a los contenidos del paquete. Luego se devuelve el control al driver, que actuará normalmente con el paquete original.

Puede haber procesos interesados en consultar cada uno de los paquetes de la red, lo que implicaría un pésimo rendimiento al tener que pasar de a uno los paquetes hacia el nivel de usuario. Para evitar esto BPF agrupa varios paquetes para luego pasarlos todos de una vez, al tiempo que se añade una marca de tiempo, tamaño y offset a los paquetes del grupo para mantener la secuencia.

Libpcap implementa un lenguaje de programación de filtros de alto nivel, que debe luego ser compilado a BPF compatible antes de ser aplicado. Cada expresión de filtrado consiste en una o más primitivas. Cada primitiva usualmente consta de un identificador (nombre o número) precedido por uno o más calificadores.

Existen 3 clases de calificadores:

- tipo: puede ser host, net o port
- dir: especifican una dirección particular de transferencia, puede ser src, dst, y combinaciones de ellos con or y and
- proto: especifica el protocolo que queremos capturar, puede ser tcp, udp, ip, ether, arp, entre otros

Libpcap provee dos importantes funciones para esta etapa de la programación: pcap_compile que se utiliza para compilar un programa de filtrado escrito con las expresiones de filtrado antes mencionadas, y pcap_setfilter que se utiliza para aplicar el filtro obtenido con la función anterior.

Captura y almacenamiento de tramas

Existen varias funciones para capturar paquetes, las principales diferencias son el número de paquetes que queremos capturar, el modo de captura, normal o promiscuo, y la manera en que se definen sus funciones de llamada o Callbacks (la función invocada cada vez que se captura un paquete). A continuación se listan las principales.

- pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms, char *errbuf)
  Se utiliza antes de entrar en el bucle de captura para obtener un descriptor de la captura de tipo pcap_t, es una estructura invisible al usuario y cuyos datos serán utilizados por las distintas funciones de la biblioteca. Acepta como parámetros el nombre del dispositivo de red en el que queremos iniciar la captura, el número máximo de bytes a capturar, si capturar en modo promiscuo o no, y cuantos milisegundos queremos que el kernel agrupe paquetes para luego pasarlos todos de una vez aumentando el rendimiento. Si la función devuelve NULL se habrá producido un error y puede encontrarse una descripción del mismo en errbuf.

- int pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user)
  Se utiliza para capturar y procesar los paquetes. cnt indica el número máximo de paquetes a procesar antes de salir, cnt = -1 para capturar indefinidamente. callback es

un puntero a la función que será invocada para procesar el paquete. La función devuelve el número de paquetes procesados o –1 en caso de error, en cuyo caso pueden emplearse las funciones pcap_error() y pcap_geterr() para mostrar un mensaje más descriptivo.

- int pcap loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)
  Es bastante parecida a pcap dispatch, la diferencia es que no finaliza cuando se produce un error por timeout. En caso de error se devolverá un número negativo y 0 si el número de paquetes especificados por cnt se ha completado con éxito.

- u char *pcap next(pcap_t *p, struct_pcap_pkthdr *h)
  Lee un único paquete y devuelve un puntero a un u_char con su contenido. Sin necesidad de declarar ninguna función callback.

Libpcap también dispone de funciones para volcar los datos a un archivo para su posterior análisis. Algunas de esas funciones son:

- pcap_dump_open: abre un fichero de salida en el que ir guardando los datos

- pcap_open_offline: abre un fichero con paquetes ya guardados en formato tcpdump en modo lectura

- pcap_dump: una vez abierto el fichero de salida con pcap_dump_open, podemos comenzar a registrar los paquetes con esta función

- pcap_dump_close: cierra el fichero de salida abierto por pcap_dump_open

Procesamiento de tramas

Esta etapa consta del reconocimiento y extracción de la información contenida en los encabezados de cada protocolo y la carga útil para su posterior análisis. Esto implica conocer como están estructuradas las cabeceras de los distintos protocolos, las cuales están definidas en los estándares RFC por lo que su consulta es fundamental a la hora de realizar esta tarea.

Por otro lado, Linux dispone de una serie de funciones que permiten facilitar la tarea de descifrar los datos. Como ejemplo, en el fichero /usr/include/netinet/ether.h se dispone de una serie de funciones que transforman direcciones Ethernet de 48 bits en texto legible.

5.8. Algunas limitaciones del modo monitor

Si bien el modo monitor supone que todos los filtros y servicios de la placa están inactivos, en la práctica hemos encontrado que en algunos casos la placa sigue realizando algún tipo de procesamiento al recibir las tramas. En particular sucedió que las placas Broadcom descartaban los paquetes si en el encabezado 802.11 las banderas indicaban que se trataba de un paquete dirigido a un access point. Es un fenómeno que no debería ocurrir, pues en modo monitor la placa no debería leer el contenido de este encabezado pero no tenemos nada más que respetar estar restricciones al construir los paquetes.

Un comportamiento similar sucede al manipular otros campos del encabezado 802.11 como el tipo y subtipo de los datos contenidos bajo el encabezado. Observamos que al utilizar tipos y subtipos “reservados” por la norma, los paquetes son descartados al arribar al destinatario. Por

lo tanto, fue necesario mantenerse con los tipos y subtipos ya definidos, como lo son *data* y *beacon*.

Por otra parte, así como al configurar una placa en modo monitor perdemos la posibilidad de setear los parámetros de la misma (ESSID, AP, etc, pues carece de sentido realizarlo), perdemos la posibilidad de setear el bitrate utilizando *iwconfig*. Si bien el comando es ejecutado de forma satisfactoria, no observamos que el mismo tenga un efecto sobre todas las placas. Para algunos fabricantes el bitrate en el modo monitor está fijo en 1 Mbps aún cuando instruyamos a la placa a utilizar otros bitrates, ya sea con *iwconfig* o a través del encabezado radiotap de los paquetes. Estamos perdiendo entonces mucho ancho de banda que a priori está disponible en las placas, que siendo 802.11g pueden alcanzar bitrates de hasta 54 Mbps.

Si bien el espíritu del encabezado radiotap es instruir a la placa sobre estos parámetros, observamos que la mayoría de las directivas son ignoradas. Vemos que no son tenidas en cuenta las directivas de potencia de transmisión, frecuencia, etc. Solamente la placa Atheros respondió a la directiva de bitrate, aunque no a las demás.

Hemos mencionado también que no todas las marcas de placas 802.11, así como sistemas operativos admiten el modo monitor. Este modo fue introducido en Linux y de a poco otros sistemas operativos lo van incorporando. De la misma forma los chipset Broadcom, Intel y Atheros que estamos utilizando, admiten la operación en este modo pero hay otra cantidad de marcas que no lo soportan.

5.9. Posibilidades trabajando con modo monitor

Como hemos mencionado anteriormente, al inyectar paquetes en una placa configurada en modo monitor, debemos encargarnos de formatear todos los encabezados del mismo. De la misma forma, debemos realizar todo el procesamiento de los encabezados al recibir un paquete, pues no tenemos servicios activos que realicen esta tarea por nosotros.

Si bien esto nos agrega una gran carga de trabajo a la hora de programar, deja abierta también la puerta para introducir más fácilmente conceptos de otras tecnologías inalámbricas que estudiamos al comenzar el proyecto. Podríamos por ejemplo configurar *digipeaters* con alias genéricos como en APRS y así extender el alcance de nuestras redes aún con topologías desconocidas, o realizar ruteo basado en posición geográfica si equipáramos a nuestros dispositivos con GPS. También el descubrimiento de vecinos es una tarea a resolver y podemos aquí nuevamente utilizar conceptos de distintas tecnologías.

A su vez, con la flexibilidad que nos provee este modo, podemos acercarnos aún más a los objetivos de Car-2-Car y las VANETs. Es por esto que creemos que el camino por la inyección de paquetes en modo monitor es el más apropiado y el que presenta las mejores posibilidades para los cometidos del proyecto Vo!zila y fue el elegido para el desarrollo del prototipo que se detalla en los capítulos siguientes.

5.10. Pruebas de RF

Si bien con las pruebas descritas anteriormente logramos comprender muchos aspectos del funcionamiento de las placas 802.11, se mantenían muchas interrogantes. Algunas preguntas como si las placas efectivamente respondían a los comandos de control de potencia, o los efectos de la interferencia sobre el enlace sólo podían responderse relevando el espectro de la señal.

Utilizando un analizador de espectro realizamos pruebas sobre las placas configuradas en modo monitor y funcionando en modo 802.11b a 1 Mbps.

Para comprobar el control de potencia utilizamos las funciones de medición de potencia de canal 802.11b provista por el analizador de espectro y manteniendo una inyección de paquetes al límite de la capacidad fuimos variando la potencia de la placa con el comando *iwconfig wlanX txpower <potencia en dBm>*.

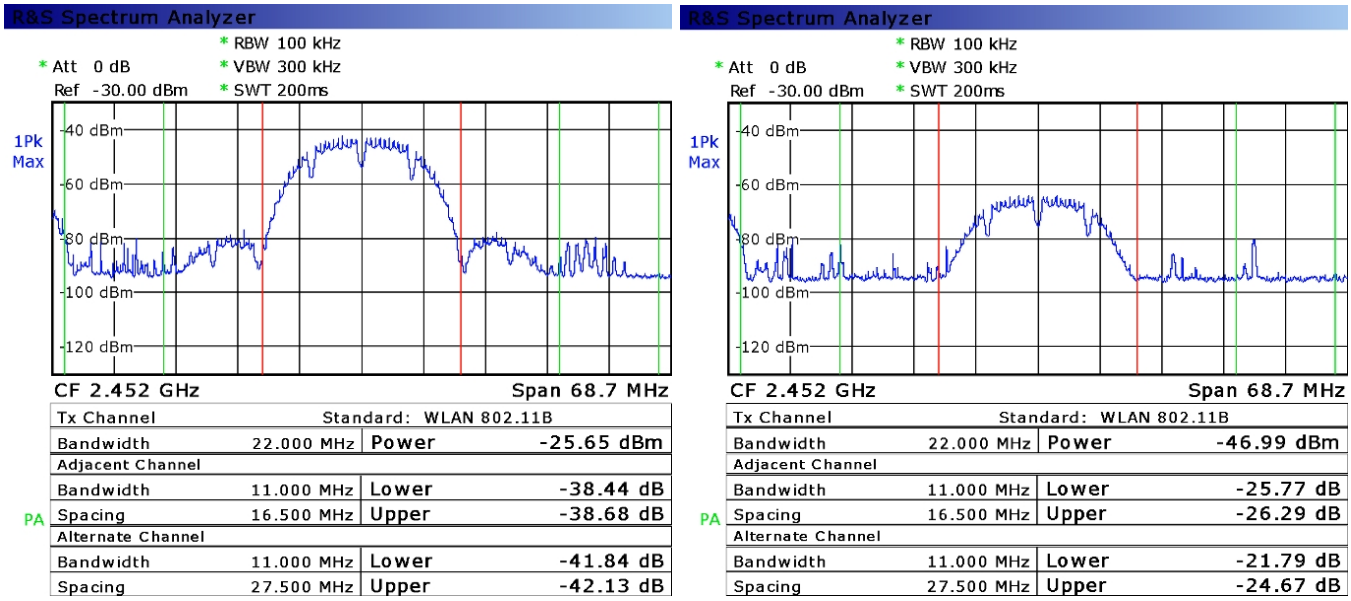

Figura 61 - Potencia medida en el canal para configuraciones de 20 dBm y 1 dBm

En las capturas vemos la placa con chipset Atheros configurada con una potencia de transmisión de 20 dBm en el primer caso, y 1 dBm en el segundo. Observamos una diferencia de casi 22 dB en la potencia medida con el instrumento. La misma prueba se realizó con una configuración de potencia de 15 dBm, 10 dBm y 5 dBm, y en todos estos casos las mediciones corroboraron un correcto funcionamiento del control de potencia. En estos casos, la diferencia de potencia entre las distintas configuraciones fue incluso mejor que en el caso citado arriba, donde tenemos casi 3 dB de diferencia con lo configurado.

Una de las primeras conclusiones que se desprenden de esta prueba es que el comando *iwconfig* si bien se ejecuta correctamente para cualquier valor de potencia que se pase como parámetro, sólo responde para valores positivos. Por lo tanto no es posible configurar las placas para tener una potencia de salida de 0 dBm o menos.

Uno de los comportamientos más extraños que detectamos en las pruebas fue que las placas se comunicaban entre sí aún cuando estaban configuradas en canales diferentes. Logramos mantener una comunicación aceptable entre ellas aún con una separación de 3 canales.

Confirmamos con el analizador de espectro que efectivamente las placas estaban transmitiendo en canales diferentes, por lo que concluimos que los receptores o bien están decodificando los lóbulos secundarios, o la mala filtración en recepción hace que la potencia recibida en los otros canales termine excitando al receptor.

Este mismo fenómeno causa también que las placas se interfieran cuando hablan en canales diferentes. Si bien está previsto que los canales de 802.11b se solapen, los canales 1, 6 y 11 están lo suficientemente separados entre sí como para no tener solape como se muestra en la figura 62. Teóricamente, 3 placas podrían transmitir en simultáneo si usan estos canales.

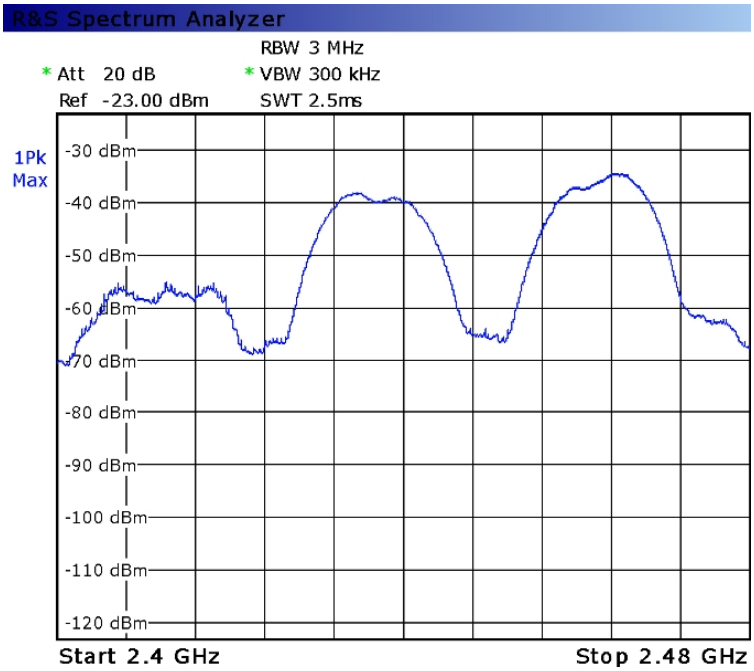

Figura 62 - Canales disjuntos

Sin embargo, observamos que las transmisiones en canales disjuntos generan interferencia. En particular, la placa con chipset Broadcom interfiere a la placa con chipset Intel, aún con una separación de 7 canales. Realizamos la siguiente prueba:

1. Placa Intel 3945 transmitiendo a la máxima capacidad en el canal 11.
    - Comprobamos el correcto funcionamiento de la placa, transmitiendo prácticamente de continuo.
2. Habilitamos luego la transmisión en la placa Broadcom 4311 en el canal 5.

El resultado es que al habilitar la placa Broadcom, la placa Intel instantáneamente detiene su transmisión en el canal 11, quedando la primera transmitiendo en el canal 5 sin problemas. Esta misma situación no ocurre a la inversa, es decir, la placa Broadcom siempre prevalece sobre la Intel. Entendemos que el mecanismo DCF en la placa Intel pasa a detectar el canal 11 como ocupado cuando la placa Broadcom inicia su transmisión. Por tratarse de canales disjuntos, la presencia de la portadora en el canal 5 no debería ser el problema para la placa Intel, sino un aumento en el piso de ruido en la frecuencia del canal 11.

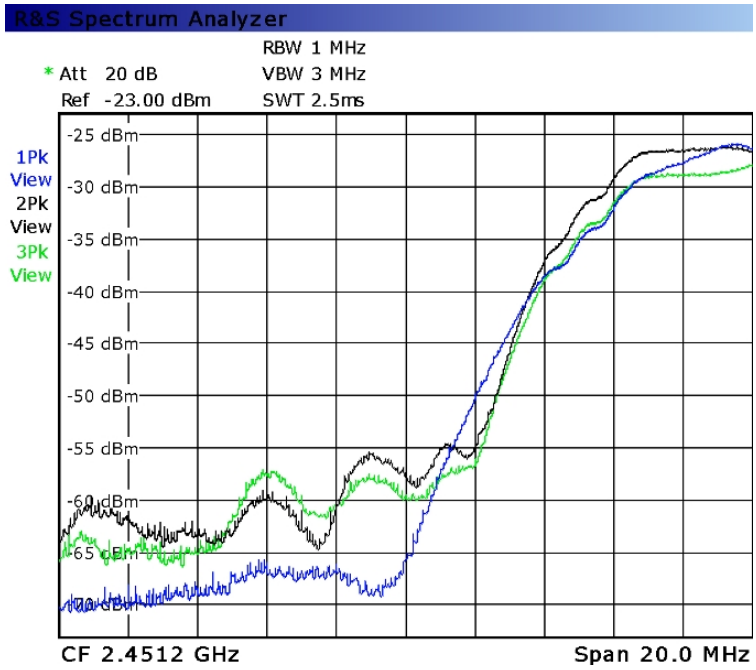

Figura 63 - Diferente comportamiento fuera de banda para distintos fabricantes

En la figura 63 mostramos el comportamiento de 3 placas inalámbricas diferentes fuera de la banda de 22 MHz asignada a cada canal. En azul se representa la placa con chipset Intel mientras que las otras dos corresponden a placas con chipset Broadcom. Se observa claramente la diferencia en el nivel de señal fuera de banda entre los dos fabricantes. De hecho, tenemos casi 15 dB de diferencia en algunos puntos. Evidentemente la calidad de filtrado de la señal transmitida es muy diferente según fabricantes. Creemos que esta puede ser la razón para la interferencia entre canales disjuntos, donde los lóbulos secundarios fuera de banda están siendo detectados como una portadora en otro canal o simplemente causando un nivel de ruido inaceptable.

Otra hipótesis es que el receptor de la placa Intel es más susceptible a la saturación, y que de hecho no sea el mecanismo DCF quien está previniendo la transmisión sino que el receptor de la placa se satura por el exceso de potencia presente en su entrada de RF.

Este último defecto descrito puede ser el responsable de las fallas en el modo ad-hoc que fueron detalladas en 5.3. Podríamos suponer que la asociación ad-hoc entre 3 placas no falla por razones de software sino que son las interfaces RF de las placas las que no soportan el tráfico generado por los intentos de asociación de las 3.

6. Prototipo Vo!zila

6.1. Introducción

La implementación del sistema de comunicación propuesto en este proyecto, basado en tarjetas inalámbricas con tecnología wifi como principal vía de comunicación, está ligada implícitamente a la elaboración de software que maneje el sistema. El mismo debe ser capaz de implementar los diferentes procesos y protocolos propuestos en forma automática y transparente al usuario, así como también disponer de una interfaz adecuada para permitir al usuario acceder a información proveniente de las aplicaciones de más alto nivel y poder incluso interactuar con ellas.

Este capítulo tiene como principal objetivo documentar todos los aspectos relacionados al diseño e implementación del software, que serán de utilidad no solo al usuario final, sino también, y principalmente, a futuros desarrolladores que utilicen este proyecto como plataforma para poder continuar incursionando en el mundo de las comunicaciones vehiculares.

En particular quedarán muchas áreas por desarrollar, que por motivos de tiempo no pudieron ser abarcadas, como ser seguridad en las comunicaciones y el ruteo eficiente a modo de ejemplo, y que son un campo fértil para el desarrollo posterior de un sistema más avanzado partiendo de esta base.

6.2. Detalles previos

Tras un estudio previo acerca de cuál podía ser el modo más confiable de utilizar las interfaces inalámbricas, se optó por la captura e inyección de paquetes en modo monitor. A su vez, para poder inyectar y capturar paquetes se hizo uso de la biblioteca de funciones *libpcap* que está disponible para el lenguaje C. Se utilizó además el encabezado Radiotap para realizar la interfaz con los aspectos físicos de la placa inalámbrica.

Si bien en un principio la idea era programar en un lenguaje de más alto nivel, como java por ejemplo, la necesidad de utilizar las bibliotecas antes mencionadas determinó inevitablemente la utilización de C como lenguaje de desarrollo. Sin embargo cabe mencionar como ventaja el gran control que se logra sobre los diferentes hilos que corren en paralelo en el programa, así como también la manera en que acceden ordenadamente a recursos compartidos de memoria, e incluso la utilización eficiente del procesador por parte de algunos procesos específicos. La misma razón de flexibilidad es la que nos llevó también a programar para el entorno de Linux utilizando sus diferentes bibliotecas.

El sistema debe manipular todos los encabezados relacionados con la comunicación mediante 802.11, pues en modo monitor carecemos de los servicios brindados por el sistema operativo para estos fines.

Sobre todo en lo referente al encabezado 802.11, fue necesario mantener muy acotadas las modificaciones a realizar sobre la norma. Como fue introducido en capítulos anteriores, a pesar de estar trabajando en modo monitor, las placas inalámbricas toman en cuenta la información contenida en este encabezado. Por esta razón hemos tenido que mantener los campos con un estilo similar al de la comunicación tradicional, respetando las banderas y tipos de datos. En un

comienzo intentamos utilizar los rangos reservados de tipos y subtipos pero esto generó que las placas descartaran los paquetes al recibirlos.

En lo referente al encabezado radiotap, si bien el espíritu del mismo en la trasmisión es controlar los parámetros físicos de la placa, la mayoría de los drivers ignoran sus directivas.

6.3. Funcionalidades principales del sistema

El sistema basa su comunicación en la inyección y captura de tramas 802.11, con la interfaz inalámbrica trabajando en modo monitor. Por lo tanto para utilizar el sistema, cada nodo en la red debe tener configurada su interfaz inalámbrica en modo monitor en un canal pre acordado con anterioridad. Si bien ahora podemos optar por cualquiera de los canales disponibles para WiFi, tal como lo propone Car-2-Car, sería importante contar con un canal propio, libre de tráfico interferente.

En esta etapa básica del desarrollo debemos proveer las funciones básicas para el descubrimiento de vecinos así como brindar un servicio básico de datagramas. La primera funcionalidad básica del sistema es emitir un beacon cada cierto intervalo de tiempo. Este beacon es el mensaje básico del sistema y permite al nodo ser descubierto por los que están en su alcance de radio. En este mensaje se envía información básica relacionada al entorno vehicular como ser la dirección física, nombre, alias, posición, velocidad, además de los datos ya contenidos por defecto en los encabezados de capas inferiores. Los datos de posicionamiento deben ser provistos por un GPS y, a pesar de que no es imprescindible para lograr la comunicación, es necesario entre otras cosas para realizar geocast, un algoritmo de ruteo que propone Car-2-Car basado justamente en datos geográficos de la red como lo es la posición de los nodos. En esta primera implementación estamos transmitiendo los datos de GPS mediante la sentencia NMEA GPRMC. Si bien es una forma muy ineficiente de transmitir los datos, pues se trata de una cadena de caracteres ASCII, permite su fácil lectura en los paquetes capturados por herramientas como tcpdump o wireshark. A su vez nos estamos apegando a un estándar bien establecido como lo es NMEA, sin incursionar ahora en la elaboración de un estándar nuevo para la comunicación de datos GPS.

Cada nodo dispone de una tabla donde almacena información que va recabando de los nodos vecinos. La información de cada nodo se almacena con la siguiente estructura:

```
typedef struct {
   __u8 address[6];           //!< Dirección mac.
   char name[MAX_NAME];       //!< Nombre.
   char alias[MAX_ALIAS];     //!< Alias.
   struct timeval timestamp;  //!< Timestamp de la última recepción.
   unsigned int beaconCount;  //!< Cantidad de paquetes de beacon recibidos.
   unsigned int beaconLoss;   //!< Cantidad de paquetes de beacon perdidos.
   unsigned int seqBeacon;    //!< Número de secuencia de la última recepción de Beacon.
   unsigned int seqUniTo;     //!< Número de secuencia del último paquete unicast.
   unsigned int seqUniFrom;   //!< Número de secuencia del último paquete unicast.
   unsigned int seqBroadFrom; //!< Número de secuencia de la última recepción de broadcast.
   unsigned int ttl;          //!< Time To Live.
   double lat;                //!< Latitud.
   double lon;                //!< Longitud.
   double hdg;                //!< Heading.
   double spd;                //!< Speed.
   __u8 flags;                //!< Flags.
} neighbour;
```

Al capturar un beacon, se verifica si el vecino que lo envió ya se encontraba en la tabla. De no encontrarse el vecino en la tabla, se crea un nuevo vecino con los datos provenientes de la

trama capturada. Si el nodo ya se encontraba en la tabla, se actualizan todos los datos con la nueva información.

A cada nodo en la tabla de vecinos se le asigna un campo denominado TTL (time to live). Este campo forma parte de un mecanismo de actualización de la tabla de vecinos, y es útil para no conservar en la tabla vecinos que ya no están a nuestro alcance. Al entrar un vecino a la tabla, su TTL se fija en cierto valor inicial y se va decrementando periódicamente, a no ser que se vuelva a recibir un beacon o algún otro tipo de mensaje del vecino, en cuyo caso, se vuelve a inicializar el TTL. Si el TTL llega a 0, se quita el vecino de la tabla.

Continuando con los servicios, hacemos disponibles funciones para el envío de paquetes unicast y broadcast, así como la recepción de los mismos. Estos servicios son sin garantía de entrega por lo que será tarea de las capas superiores implementar los mecanismos apropiados en caso de requerir flujos con garantía. De todas formas, en el caso de los paquetes de broadcast como en los beacons, estos se envían con un número de secuencia consecutivo para permitir en el lado receptor identificar la pérdida de paquetes. Aunque estas secuencias son independientes entre sí, hemos observado que algunas placas inalámbricas alteran la numeración introducida por el programa. Esto es parte de las limitaciones identificadas anteriormente al trabajar con el modo monitor haciendo uso del encabezado 802.11 a nuestro favor.

6.4. Estructura e inyección de tramas

Como mencionamos anteriormente, el sistema inyecta tramas al aire mediante las funciones que provee la biblioteca *libpcap* donde es necesario construir los encabezados acorde a los respectivos protocolos que se quieran utilizar. En particular, el encabezado 802.11 debe construirse siempre ya que es el encabezado de más bajo nivel al cual tenemos acceso (y que viaja con el paquete).

El encabezado Radiotap está más abajo aún, pero este no viaja en el aire sino que la tarjeta lo quita antes de enviar la trama, y del lado del receptor, lo agrega con la información disponible que provea la tarjeta. Si bien observamos que la mayoría de las directivas son ignoradas por las placas, debemos agregarlo de todas formas a los paquetes a enviar. Los desarrolladores de Radiotap están trabajando para que en un futuro las directivas de este encabezado sean interpretadas por las tarjetas al momento de inyectar una trama y puedan de esta manera setear diferentes parámetros relativos a la transmisión, como ser potencia o canal, según lo que indique este encabezado. Esto nos brindará un punto de desarrollo posterior.

Es importante remarcar que en ambos encabezados los campos son escritos en Little-Endian. Siendo esta estructura de encabezados la mínima imprescindible para la inyección de paquetes, y teniendo en cuenta el cometido de mantener al mínimo el overhead en el sistema es que el stack de encabezados quedó compuesto de la siguiente forma:

Figura 64 - Stack de encabezados a ser procesados por el sistema

El encabezado Radiotap que inyectamos tiene la siguiente estructura:

Versión	Pad	Largo del encabezado	Banderas presentes	Flags	Rate	Canal	Banderas del canal	Antena
1 byte								

Figura 65 - Encabezado Radiotap

Una plantilla en lenguaje C:

```
__u8 radioTapHeader[] = {

       0x00, 0x00,                         // versión & pad
       0x0f, 0x00,                         // header length
       0x0e, 0x08, 0x00, 0x00,             // bitmap
       0x08,                               // flags
       0x16,                               // rate
       0x85, 0x09, 0x80, 0x04,             // channel & flags
       0x00,                               // antenna
};
```

En este ejemplo estamos configurando el canal 6 con un bitrate de 11 Mbps. En todas las placas con las que hemos experimentado, el canal que efectivamente se usa es el configurado previamente mediante *iwconfig*, ignorando el contenido de este encabezado. En cuanto al rate, solamente el chipset Atheros es el que respeta esta directiva mientras que los demás fabricantes se mantienen en 1 Mbps.

El encabezado 802.11 que inyectamos tiene la siguiente estructura:

Control	Duración	Destinatario	Originador
Subtipo Vozila	BSSID Vozila	Número de secuencia	
1 byte			

Figura 66 - Encabezado 802.11 modificado

Una plantilla en lenguaje C:

```
__u8 ieeeHeader[] = {
       0x08, 0x00,                         // Frame control (ad-hoc y tipo data)
       0x00, 0x00,                         // Duration (0)
       0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,  // Destination
       0x66, 0x22, 0x33, 0x44, 0x55, 0x66,  // Source
       0x00,                               // Vozila Subtype
       0xFF, 0xFF, 0xFF, 0xFF, 0xFF,       // Vozila BSSID
       0x00, 0x00,                         // Sequence Number
};
```

Como mencionamos anteriormente, debemos ser cautelosos al modificar este encabezado pues puede generar que los paquetes sean descartados en la recepción. Esto amerita una descripción detallada de los campos.

- Control: se fijó a modo *ad-hoc* y con el tipo y subtipo *data*.
- Duración: fijada a cero.
- Destinatario: dirección mac del destinatario, FF:FF:FF:FF:FF:FF para broadcast.
- Originador: dirección mac del originador. Utilizamos la dirección mac original de la placa inalámbrica, cambiando el primer byte de 0x00 a 0x66 para facilitar el filtrado como veremos más adelante.

- Subtipo Vozila: Ante la necesidad de contar con un campo que diferencie los tipos de datos y la imposibilidad de utilizar los tipos reservados por la norma para el campo subtipo, hemos optado por tomar un byte del campo BSSID y utilizarlo como campo para indicar el tipo de datos contenidos en el paquete. Hasta el momento estamos diferenciando los tipos por *beacon, data, repeatable*, siendo este último un tipo dedicado a diferenciar los paquetes a ser propagados por la red.
- BSSID Vozila: 5 bytes para identificar el BSSID. Al momento no estamos diferenciando distintas BSSID pero es un campo muy útil para la ampliación del sistema.
- Número de secuencia: Si bien hemos detectado que algunas placas interfieren en este campo (llevando su propia numeración), la idea original es usarlo para detectar la pérdida de paquetes de beacon y de broadcast.

6.5. Filtrado de tramas

La captura de paquetes mediante *libpcap* se realiza en espacio de kernel. Estos paquetes deben pasar luego al espacio de usuario para tenerlos disponibles en nuestro programa. Este cambio de contexto es muy costoso para el sistema operativo por lo que es deseable no pasar al espacio de usuario paquetes que no corresponden a nuestro protocolo. Al estar utilizando los canales libres para WiFi, estamos sujetos a la contaminación del canal por tráfico de WLAN. Para evitar levantar paquetes que no corresponden al sistema Vo!zila, se realiza un filtrado en especio de kernel de estos paquetes y para esto es necesario identificarlos de alguna manera para saber qué filtrar.

La dirección MAC consta de 6 bytes, los primeros 3 indican el bloque de direcciones que se le da a cada fabricante, los últimos 3 identifican al dispositivo dentro de esa familia, por lo tanto el conjunto identifica unívocamente a cada dispositivo. Decidimos por tanto utilizar estas direcciones para el filtrado modificando el primer byte por 0x66, número que hoy en día es 0x00 para todas las placas comerciales. Configuramos un filtro de recepción que mira el primer byte de la dirección de origen en el encabezado 802.11. De esta manera, las tramas cuya MAC de origen comience con 0x66 serán reconocidas como provenientes del sistema Vo!zila y pasadas hacia el espacio de usuario mientras que las demás serán descartadas. El campo de dirección podemos modificarlo sin que las placas descaren los paquetes, como sí nos sucede si modificamos otros campos del encabezado 802.11, tal como comentamos en las pruebas realizadas con la inyección de paquetes en modo monitor.

6.6. Propagación de tramas

Si bien el ruteo es un aspecto que todavía no fue cubierto por el programa, sí hemos dejado disponible la propagación de las tramas haciendo uso de los conceptos de APRS. Creemos que el protocolo APRS, diseñado para comunicaciones de bajo ancho de banda introdujo una forma muy conveniente de controlar la propagación y que puede ser muy útil para incorporar a nuestro prototipo. Se trata del uso de Alias para identificar las propiedades de los nodos de la red.

Utilizando Alias estandarizados, se puede indicar si un nodo es fijo o móvil (RSU – roadside unit, OBU – onboard unit), si cuenta con características que favorezcan un alcance mayor de sus transmisiones, etc. Este es un tipo de propagación que se controla desde el origen del paquete, donde se establece quiénes van a repetir la trama en cuestión.

La propagación por Alias se realiza indicando qué Alias son los que deben repetir el paquete y durante cuántos saltos. Cuando un nodo recibe un paquete del subtipo *repeatable*, se fija si el

Alias que debe repetir el paquete coincide con el suyo. De ser así, pasa a leer la cantidad de saltos restantes para el paquete, y de ser mayor a cero, decrementa en uno este campo y re-inyecta el paquete.

Esta información es contenida en un encabezado especial diseñado para este fin. El encabezado de repetición se agrega al payload y queda debajo del encabezado 802.11.

Alias	Id	Hops restantes	Hops realizados	Originador	Repetidor 1	Repetidor 2	...

Figura 67 - Encabezado de repetición

Es muy probable que un mismo paquete a ser repetido llegue por dos caminos distintos. Para evitar procesar un paquete dos veces, los mismos contienen un número de identificación que es almacenado cada vez que un nodo repite el paquete. Cuando llega un paquete para ser repetido, lo primero que hace el nodo es verificar si ese número está en la lista de paquetes ya procesados, en cuyo caso descarta el paquete.

De la misma forma, una vez que un nodo repite un paquete, su dirección pasa a estar en el campo de originador del encabezado 802.11. Para mantener el registro del verdadero originador del paquete, cada vez que un nodo repite un paquete, previo a sustituir el campo de originador, copia la dirección contenida ahí al encabezado de repetición. De esta manera, al llegar el paquete a destino no sólo sabemos quién fue el verdadero originador sino que también tenemos un registro de todos los nodos que atravesó en el camino.

Para enviar este tipo de paquetes, contamos con funciones de envío de unicast y broadcast que se encargan de generar el encabezado de repetición. Estas se agregan a las funciones de envío de paquetes unicast y broadcast tradicionales ya descritas.

6.7. Estructura del programa

El programa en sí fue creado en lenguaje C utilizando programación concurrente para mantener varios hilos corriendo en paralelo. En particular tenemos 6 hilos:

- Hilo principal que maneja la entrada de instrucciones por la línea de comando.
- Hilo receptor de paquetes que se encarga de capturar los paquetes en la interfaz inalámbrica y procesarlos según su subtipo.
- Hilo de beacon que inyecta en la interfaz los paquetes de beacon en un intervalo de tiempo predefinido, actualizando en él la información de GPS y número de secuencia
- Hilo de GPS que mantiene la comunicación con el dispositivo GPS (y a modos de prueba también es capaz de simular una entrada de GPS).
- Hilo mantenimiento de la tabla de vecinos que se encarga de decrementar el TTL de los miembros de la tabla de vecinos en forma periódica y así liberar la misma de nodos que ya no estén en alcance.
- Hilo de interfaz gráfica que despliega la información de la tabla de vecinos así como un mapa con la posición relativa de los mismos respecto a la posición propia.

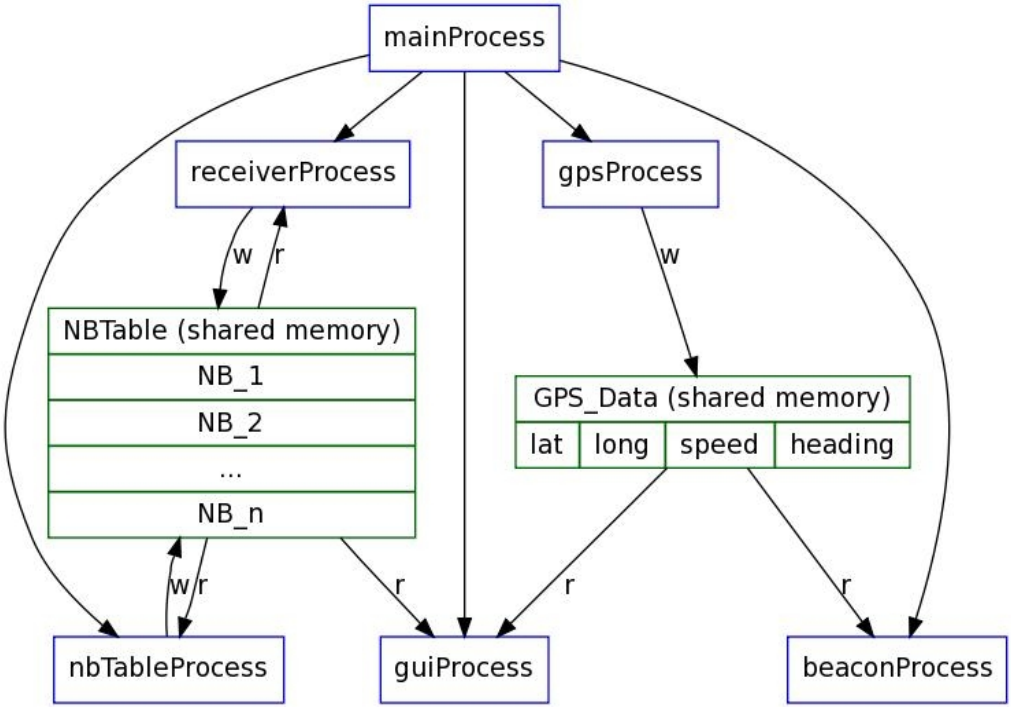

Figura 68 - Dependencia de procesos y memoria compartida

Manteniendo el mismo concepto modular, los códigos fuente fueron separados en distintas bibliotecas entre las que destacamos:

- Programa principal: se encarga de inicializar el programa y lanzar los distintos hilos.
- Biblioteca *sender*: Contiene las funciones de inyección de tramas, como lo son la inyección de un paquete de beacon, un paquete unicast, un paquete de broadcast o la retransmisión de un paquete marcado para su propagación en la red.
- Biblioteca *receiver*: Contiene las funciones de captura y procesamiento de paquetes. Es importante destacar aquí el uso del método *select* de Linux que permite esperar por una interrupción de la placa inalámbrica para llamar a las funciones de capturas de *libpcap*. Esto es necesario para bajar el uso del procesador por parte del programa, pues las funciones de *libpcap* que deberían aguardar por la llegada de paquetes naturalmente consumen muchos recursos en forma innecesaria.
- Biblioteca *neighbours*: Contiene las funciones necesarias para mantener la tabla de vecinos, así como para obtener información de la misma. Dado que esta tabla puede ser accedida desde varios hilos, es necesario alojarla en un segmento de memoria compartida controlando su acceso mediante el uso de semáforos.
- Biblioteca *GPS*: Contiene las funciones para manipular los datos de GPS. También tiene implementadas funciones de simulación para utilizar en las pruebas estáticas del sistema. Esta biblioteca también maneja un segmento de memoria compartida donde se aloja la posición actual del vehículo de forma que esté disponible para todos los hilos. Naturalmente, el acceso a la misma está coordinado por semáforos.
- Biblioteca *GUI*: Se encarga de la interfaz gráfica del programa.

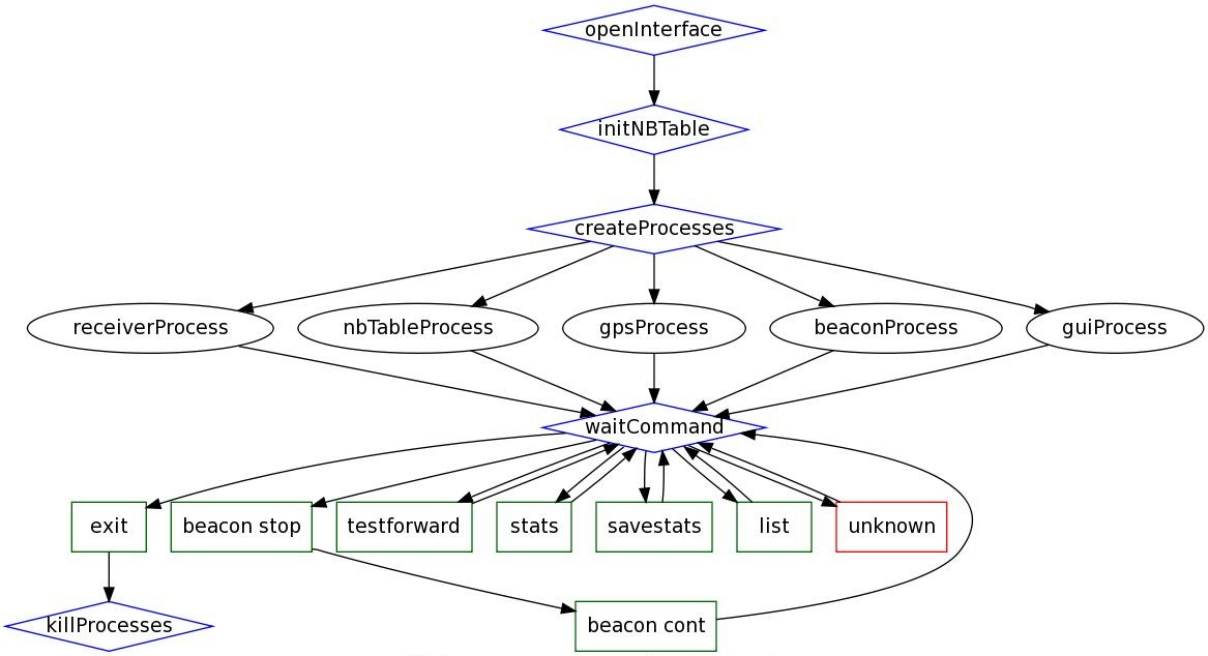

Figura 69 - Intérprete de comandos de línea

El programa principal puede ser invocado con el nombre de la interfaz inalámbrica a utilizar como argumento, o dejar que el programa busque la primera interfaz inalámbrica disponible. El primer modo de uso es útil cuando se cuenta con varias interfaces inalámbricas y queremos indicar cuál se debe utilizar para el programa. Una vez que el programa identifica la interfaz a utilizar se pasa a su apertura por parte de *libpcap* obteniendo el *handler* necesario para controlar tanto la captura como la inyección de paquetes en dicha interfaz.

Luego se da paso a la creación de los hilos independientes que a su vez se encargarán de inicializar sus recursos particulares, tales como segmentos de memoria compartida y semáforos. Para esto se utilizan los recursos IPC de Linux que proveen desde el sistema operativo segmentos de memoria compartida y semáforos.

Para finalizar mostramos la siguiente captura de la interfaz gráfica del programa. Cabe aclarar que la información desplegada por la misma puede ser adaptada a las necesidades de la aplicación en cuestión. La tasa de actualización de la información puede ser variada en función de las necesidades de la aplicación.

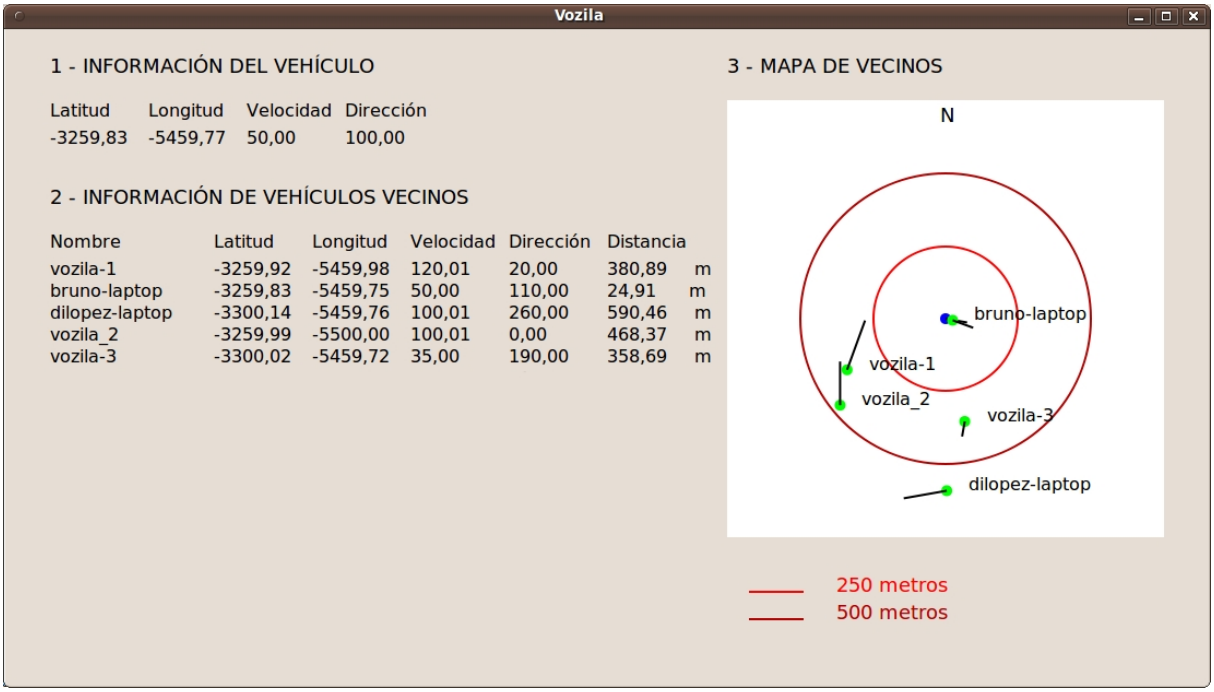

Figura 70 - Interfaz Gráfica

En el ejemplo de la figura 70 tenemos a la izquierda una lista de los vecinos en alcance de radio. Identificados por su nombre, se despliegan las coordenadas geográficas de su ubicación, velocidad y dirección de desplazamiento y la distancia que lo separa del vehículo propio. A la derecha tenemos una representación tipo radar donde se representa el vector posición de los demás vehículos respecto al vehículo propio, ubicado en el centro.

6.8. Pruebas

Se realizaron pruebas prácticas sobre el prototipo para estudiar su comportamiento frente a diferentes situaciones. Para ello se equiparon 4 nodos con el prototipo, 2 móviles y 2 fijos, y se realizaron pruebas en entornos urbanos.

Alcance de radio y robustez

Una de las pruebas estuvo orientada a estudiar el alcance de radio de la las placas 802.11 cuando se utilizan en modo monitor. De la misma forma, se estudió la robustez del sistema frente a cambios de topología. Al carecer de mecanismos de asociación, esperábamos encontrar una respuesta inmediata al entrar y salir un nodo del área de cobertura.

Para esta prueba equipamos un vehículo con una notebook corriendo el prototipo a modo de OBU. Un nodo fijo hacía las veces de RSU en un primer piso sobre la vereda. Se trata de un entorno urbano denso, con fuerte vegetación. Ambos nodos se anunciaban a la red mediante el beacon periódico provisto por el prototipo y cada uno se verificaba la conectividad tanto a través del software Vo!zila, como con wireshark. Se realizó un recorrido en torno a la manzana (recorrido 1) con el vehículo, el cual detallamos en la figura 71. En verde están marcados los segmentos donde la conectividad era positiva y en rojo donde no la hubo. Se realizó la experiencia en varias oportunidades y el resultado fue siempre muy similar.

Figura 71 - Recorrido 1

También se realizaron pruebas con otro recorrido (recorrido 2), donde se buscó determinar el máximo alcance en línea recta obteniendo más de 200 mts de alcance (figura 72).

Figura 72 - Recorrido 2

El resultado de estas pruebas fue sumamente favorable constatando las siguientes propiedades:

- La recuperación de la conectividad a nivel del prototipo es inmediata: ni bien aparece un paquete de beacon en wireshark, podemos ver cómo el vecino es incorporado a la tabla.
- El sistema es resistente frente a las innumerables fuentes de interferencia que se encontraron en el camino. Podemos decir que al menos la baja carga del canal que requiere el beacon es satisfecha por 802.11 aún con fuerte interferencia en 2.4 GHz. Quedó registrado en wireshark que a lo largo del camino se atravesaron infinidad de access points.

Propagación de paquetes

Para probar el método de propagación de paquetes propuesto en este prototipo, ubicamos los 4 nodos de forma que sólo hubiera alcance de radio entre nodos consecutivos, como se muestra en la figura 73. Se configuraron los nodos para responder todos al mismo alias y desde los extremos se enviaron paquetes de prueba para ser propagados en la red.

Figura 73 - Pruebas de propagación de paquetes

Nuevamente en este caso el resultado fue sumamente satisfactorio, constatando no sólo que los paquetes llegaban de un extremo (RSU1) al otro (OBU2) y viceversa, sino que el mecanismo para evitar loops funcionó correctamente. Esto último se verificó mediante wireshark, estudiando explícitamente el encabezado de repetición de los paquetes que circularon por la red. En OBU2 por tanto, recibíamos paquetes desde OBU1, que en su encabezado de repetición tenían como lista de nodos intermedios a RSU1 y RSU2.

Packet loss y throughput

Finalmente realizamos pruebas de packet loss y capacidad del sistema. Para ello creamos rutinas especiales que modificaban la cadencia de envío de beacons así como el tamaño de estos. La rutina se encargaba de calcular la cantidad de paquetes perdidos, usando para esto el número de secuencia provisto en los paquetes de beacons.

En este caso observamos que mientras se mantuviera la carga dentro de la capacidad del canal, no ocurría un valor significativo de pérdida de paquetes. De hecho, la cantidad de paquetes perdidos parece tener mayor relación con la calidad de la placa receptora que con un tema de colisiones. Pudimos ver que la placa Atheros tiene una mayor sensibilidad y por tanto logra decodificar un mayor número de paquetes.

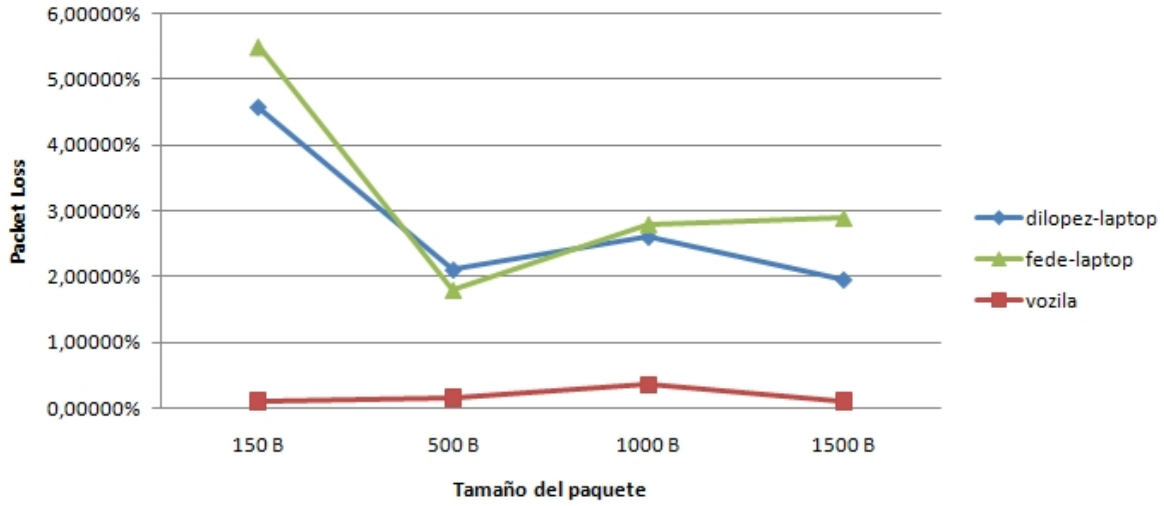

Figura 74 - Packet Loss vs. Tamaño del paquete envíado por un mismo nodo

En la figura 74 observamos los valores de packet loss para la comunicación proveniente desde un nodo, según se calcula en los otros 3 nodos en forma simultánea. El nodo Vozila es el equipado con una placa Atheros y vemos como los valores de packet loss con los que recibe la comunicación son mejores que las otras dos placas. El comportamiento de las otras dos placas se asemeja bastante en este caso y en todos los estudiados. Esto probablemente se deba al hecho que ambas placas utilizan el mismo chipset (Broadcom) y como mencionamos anteriormente, la pérdida de paquetes parece estar relacionada más que nada a la calidad de decodificación de la placa inalámbrica.

Para probar el throughput, estresamos las pruebas de forma que la cadencia de envíos de paquetes y el tamaño de los mismos llegara a saturar la capacidad del canal. En estas pruebas las placas estaban funcionando a una velocidad de 1 Mbps.

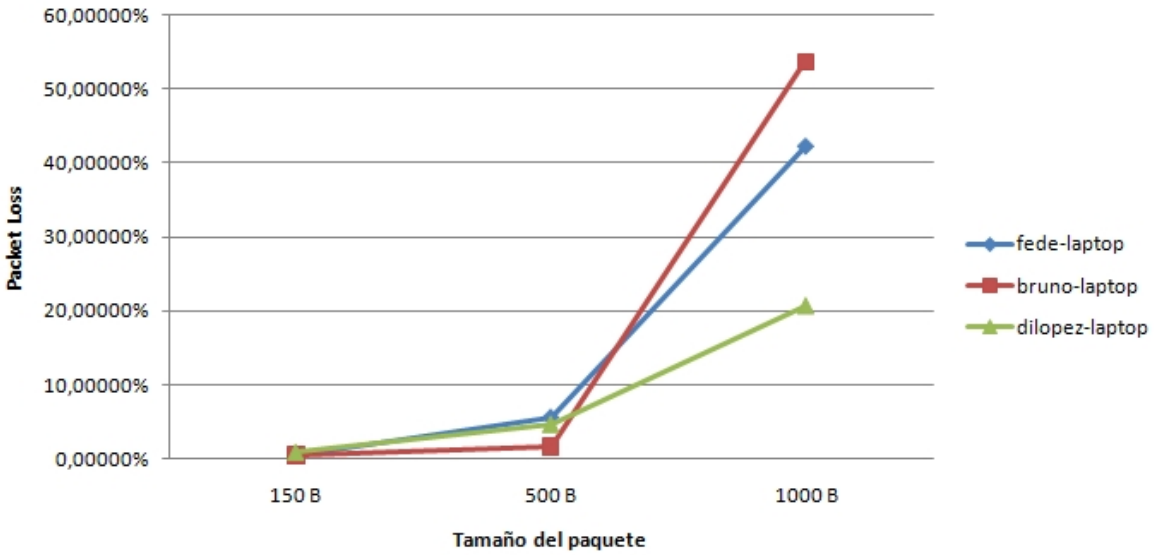

Figura 75 - Packet Loss vs. Tamaño del paquete envíado por distintos nodos

En la figura 75 vemos el packet loss calculado en el nodo equipado con placa Atheros, para la comunicación proveniente desde los otros 3 nodos. Se está utilizando una cadencia de paquetes en cada nodo de 20 ms, y vemos como los 3 comienzan a saturar en el mismo punto. Teniendo en cuenta que el nodo que está realizando el cálculo también está participando en el envío de paquetes, tenemos 4 nodos enviando paquetes de 1 KB cada 20 ms. Para cada nodo estaríamos requiriendo una capacidad de 50 KB/seg, por lo que un total de 200 KB/seg serían necesarios para comunicar a los 4 nodos en un límite teórico. Sin embargo el canal de 1 Mbps nos está brindando una capacidad máxima teórica de tan sólo 125 KB/seg. Eso explica el aumento abrupto en el packet loss para paquetes de 1 KB.

Sin embargo, sorprende el buen comportamiento para paquetes de 500 B, pues aquí la capacidad mínima requerida es de 100 KB/seg y si tomamos en cuenta que 125 KB/seg es el máximo teórico del canal, los valores de packet loss obtenidos son muy buenos.

7. Conclusiones del proyecto

Comenzamos el proyecto Vo!zila con motivación de crear un sistema de comunicación inalámbrica entre vehículos, que fuera capaz de funcionar en forma autónoma y sin infraestructura. Este último punto es de vital importancia sobre todo para la introducción de una tecnología de este tipo en países de bajos recursos como Uruguay ya que la instalación de infraestructura para soportar este sistema puede resultar muy costosa. Sin lugar a dudas que la única forma en que se puede popularizar un sistema de este tipo es si cada usuario es independiente de utilizarlo.

Otro de los fundamentos del proyecto era centrarse en las bases del sistema, sin focalizarnos en una aplicación determinada. Este punto también es vital para el éxito del sistema, pues será el mercado quien determine las aplicaciones que son “interesantes” para correr sobre el sistema. Simplemente debemos dejar una base firme y versátil.

Con estos cometidos en mente nos avocamos a estudiar cuál sería el mejor camino para lograr estos objetivos. Comenzamos entonces por investigar los distintos protocolos y sistemas de comunicación que se usan para sistemas móviles, tanto para comunicaciones de larga distancia (sistemas en HF como APRS) como para muy corta distancia como lo es bluetooth. Encontramos que todas estas tecnologías tienen sus fuertes y debilidades, y que el mejor resultado se obtendría combinando todas.

Vimos como el método del *digipeating* por alias genérico de APRS es una muy buena solución al encaminamiento de paquetes en una red de la que no conocemos la topología. De hecho, implementamos una versión muy simplificada de este algoritmo en nuestro prototipo. Por el contrario, los tiempos circulación de la información en APRS son muy largos, haciendo que este sistema por sí mismo no sea apropiado para aplicaciones donde el delay es crítico como puede ser un entorno urbano con información de seguridad.

Luego en STDMA vimos como se puede administrar en forma muy eficiente el acceso a un canal compartido de bajo ancho de banda. Si bien en el ejemplo estudiado (AIS), el límite de vehículos dados por la capacidad del canal es de 250, aumentando el ancho de banda del mismo se podrían lograr resultados mejores y fácilmente adaptables a aplicaciones masivas, como las vinculadas al mercado automotor privado.

Con bluetooth, dimos el puntapié al estudio de los sistemas de corto alcance. Bluetooth con sus piconets resuelve la falta de infraestructura organizándose entre nodos vecinos y asignando uno como nodo maestro, coordinador de las comunicaciones. Es posible pensar una implementación a gran escala donde las piconets se conectan entre sí (hecho soportado por la norma), creando una gran malla interconectada. Sin embargo, dado que bluetooth fue diseñado con determinadas aplicaciones en mente, tiene desde el comienzo demasiado overhead de protocolo.

Como parte este estudio preliminar, nos adentramos en las redes vehiculares modernas. Vimos con sorpresa la cantidad de esfuerzos internacionales en torno a las VANETs, como los son la variedad de consorcios compuestos tanto por universidades como por fabricantes de automóviles. La industria automotriz está muy avocada al desarrollo de estos sistemas, no sólo por sus aplicaciones en seguridad sino por la posibilidad de extender las posibilidades de *infotainment* al automóvil mismo. No es en vano que la Unión Europea está prestando especial importancia al tema, financiando a estos consorcios desde el 6º programa marco.

La mayoría de estos grupos de trabajo, como el Car-2-Car Consortium, Network On Wheels, etc, proponen como punto de partida el estándar de comunicación WLAN 802.11. Si bien existe el estándar borrador 802.11p diseñado específicamente para VANETs y todo indica que éste será el que se implemente definitivamente, resultó interesante investigar la viabilidad del 802.11 comercial para este tipo de redes.

Es por esto que prestamos una atención especial a este protocolo, realizando no sólo un estudio teórico sino que también implementamos sistemas sencillos para realizar pruebas prácticas que nos ayudaran a comprender a cabalidad el funcionamiento del mismo. Comparamos el modo ad-hoc con la inyección de paquetes en modo monitor, y también estudiamos el comportamiento de las placas inalámbricas comerciales en RF utilizando un analizador de espectro.

De aquí se desprendieron resultados sumamente interesantes y que requirieron muchas horas de trabajo e investigación. Vimos que el comportamiento de las placas depende fuertemente del fabricante y de su implementación de los drivers. En el caso del modo ad-hoc, encontramos grandes problemas para el establecimiento de redes de más de 2 nodos a pesar que nada indica que esto sea inviable. De hecho, es un problema que en Windows no se presenta.

Afortunadamente existió una alternativa al modo ad-hoc para crear redes carentes de infraestructura (recordemos que en 802.11, el modo normal de operación es a través de un access point que coordina a todas las estaciones). Un modo de operación diseñado originalmente para la auditoría de redes 802.11 llamado modo *Monitor*, permite no sólo la recepción de las tramas “crudas” que están en el aire en modo promiscuo, sino que con especial cuidado en el manejo de todos los encabezados se pueden inyectar paquetes al aire. La gran ventaja de este camino es precisamente que en este modo la placa no necesita estar asociada a ningún otro dispositivo de la red para recibir o transmitir tramas. Pasamos entonces a enfocar los esfuerzos de nuestro proyecto en este camino, el cual nos permitiría emular hasta cierto nivel las funciones de una VANET.

Partiendo de esta base es que diseñamos el prototipo presentado en este documento, cumpliendo con uno de los objetivos fundamentales del proyecto que era implementar un sistema tangible y no sólo quedarnos en el estudio teórico de las posibles soluciones. El prototipo Vo!zila fue implementado a bajo nivel, prestando servicios de capa de red básicos para correr sobre distintas aplicaciones. Equipamos al prototipo con funciones básica de propagación de paquetes en la red y a su vez creamos una aplicación a modo de ejemplo, que mediante una representación gráfica muestra la ubicación geográfica de todos los vecinos.

Las pruebas realizadas sobre el prototipo arrojaron resultados sumamente alentadores. El sistema cumplió a cabalidad con los objetivos propuestos en forma muy eficiente si tomamos en cuenta la cantidad de obstáculos que se presentan en las comunicaciones a través de los canales libres de 802.11. Queda demostrado que es posible implementar una solución económica y sencilla utilizando equipamiento de consumo masivo. Esta conclusión no es nada trivial, pues es importante para un país cuyo mayor potencial se encuentra en la capacidad profesional más que en el acceso a desarrollos tecnológicos de hardware, encontrar caminos alternativos que nos acerquen a las tendencias mundiales.

En este sentido, los cambios propuestos por 802.11p dan mayores garantías a la comunicación en el ambiente móvil y operando en canales reservados, libres del tráfico ordinario que nos causó tantos problemas de interferencia. A su vez, al operar con canales de control y de tráfico en paralelo, la capacidad del sistema se incrementa.

7.1. Desarrollo futuro

El programa fue implementado de forma abierta y con suficiente documentación para que sirva como base para desarrollos más avanzados en el futuro. Como mencionábamos al comienzo del capítulo, en esta primera etapa se trataba de dejar disponibles servicios hasta capa 3. Creemos que la implementación de las capas más altas va a depender fuertemente de la aplicación que se le vaya a dar al sistema, siendo esto muy variable.

Si bien hemos dejado disponible la propagación de paquetes mediante el método de los Alias, sería muy conveniente para completar una base robusta implementar algún método de ruteo eficiente para redes móviles. Este no es un problema trivial, como estudiamos en los capítulos anteriores y su eficiente implementación podría ser tema de un proyecto de fin de carrera completo. Esta ampliación le agregaría una muy importante funcionalidad al sistema, ya sea para “dirigir” mensajes hacia determinada región de interés (vehículos que vienen atrás, etc) o para establecer comunicaciones punto a punto y realizar transferencia de datos de interés (por ejemplo entre un ómnibus y su centro de control).

Pensando ya en un despliegue piloto, sería interesante trabajar con dispositivos embebidos que puedan ser fácilmente ubicados dentro de un vehículo. El programa no tiene exigencias fuertes de memoria por lo que la unidad de abordo puede mantenerse bastante sencilla. A esta se le necesita conectar un receptor de GPS y conectar ambos a sus respectivas antenas en el exterior del vehículo. Ya en un despliegue comercial, es posible integrar en un mismo dispositivo tanto la unidad de procesamiento como la placa inalámbrica y el receptor de GPS.

8. Bibliografía

1. **Labiod, Huda, Afifi, Hossam y Santis, Costantino.** *Wi-Fi, Bluetooth, Zigbee and WiMax.* s.l. : Springer, 2007. ISBN-10 1402053967.

2. **TechnoCom.** TheWAVE Communications Stack: 802.11p, 1609.4, 1609.3. *IEEE Vehicular Technology Society.* [En línea] 2007. [Citado el: 19 de junio de 2009.] http://www.ieeevtc.org/plenaries/vtc2007fall/34.pdf.

3. **Ni, Sze-Yao, y otros.** *The Broadcast Storm Problem in Mobile Ad Hoc Network.* Taiwan : Department of Computer Science and Information Engineering.

4. **Lee, Alex.** Survey of Position Based Routing for Inter Vehicle Communication System. *Malaysia University Science Technology.* [En línea] [Citado el: 10 de enero de 2010.] http://mit.edu/its/mitsimlab.html.

5. Radiotap Headers. *Linux Kernel Documentation.* [En línea] [Citado el: 8 de octubre de 2009.] http://www.mjmwired.net/kernel/Documentation/networking/radiotap-headers.txt.

6. Radiotap Hao. [En línea] [Citado el: 8 de octubre de 2009.] http://my.opera.com/midshipman/blog/index.dml/tag/Radiotap.

7. Radiotap - Linux Wireless. *Linux Wireless.* [En línea] [Citado el: 9 de octubre de 2009.] http://linuxwireless.org/en/developers/Documentation/radiotap.

8. Radiotap. *Radiotap.* [En línea] [Citado el: 8 de octubre de 2009.] http://www.radiotap.org.

9. **González, Javier, Bauzá, Ramón y Sepulcre, Miguel.** Protocolos para comunicaciones móviles vehiculares multihop. *Universidad Miguel Hernández de Elche.* [En línea] [Citado el: 19 de julio de 2009.] http://www.uwicore.umh.es/files/paper/2008_national/uwicore_IST08_Protocolos%20para%20Comunicaciones%20M%C3%B3viles%20Vehiculares%20Multihop.pdf.

10. *Problems When Realizing Ad Hoc Networks: How a Hierarchical Architecture Can Help.* **Bouckaert, Stefan, y otros.** s.l. : PIPS, 2007. International Multiconference on Computer Science and Information Technology. págs. 995-1004. ISSN 1896-7094.

11. **Mohmad, N. R., Abdullah, A. y Amri, A. F.** Performance Evaluation of AODV, DSDV & DSR. [En línea] [Citado el: 10 de enero de 2010.] http://www.icir.org/bkarp/gpsr/gpsr.html.

12. **Organización de la Aviación Civil Internacional.** Manual on detailed technical specificatinos for the VDL Mode 4 digital link. *Report of the seventh meeting of the Aeronautical Mobile Communications Panel.* [En línea] 30 de marzo de 2000. [Citado el: 7 de junio de 2009.] http://www.icao.int/anb/panels/ACP/Meetings/amcp7/rep2appB.PDF.

13. *Locationbased routing for vehicular ad-hoc networks.* **Fübler, H., y otros.** 1, enero de 2003, ACM SIGMOBILE Mobile Computing and Communications Review (MC2R), Vol. 7, págs. 47-49.

14. **Ernst, Thierry.** ITS Communication Architectures: The Road Towards IPv6. *GeoNet Documents.* [En línea] 2009. [Citado el: 3 de marzo de 2010.] http://www.geonet-project.eu/?download=20091029-ITS-IPv6-VNC-TErnst.pdf.

15. *ITS Communication Architectures: The Road towards IPv6.* **Ernst, Thierry.** Tokio : s.n., 2009. 1st IEEE Vehicular Networking Conference.

16. **Bruninga, Bob.** Introduction to APRS. *Automatic Packet Reporting System.* [En línea] febrero de 2009. [Citado el: 2009 de mayo de 2.] http://aprs.org/APRS-by-Bob-j.ppt.

17. **Internationl Maritime Information Systems.** *Indroduction to the maritime Automatic Identification System – a Systems approach.* s.l. : Internationl Maritime Information Systems, 2004.

18. **IEEE Transportation Technology Council.** *IEEE Trial-Use Standard P1609.1.* s.l. : IEEE, 2006.

19. —. *IEEE Trial-Use Standard 1609.4.* s.l. : IEEE, 2006.

20. —. *IEEE Trial-Use Standard 1609.3.* s.l. : IEEE, 2007.

21. **IEEE Intelligent Transportation Systems Society.** *IEEE Trial-Use Standard 1609.2.* s.l. : IEEE, 2006.

22. **IEEE Computer Society.** *IEEE Standard 802.15.1.* s.l. : IEEE, 2005. ISBN 0-7381-4708-7.

23. —. *IEEE Standard 802.11a.* s.l. : IEEE, 1999. ISBN 0-7381-1810-9.

24. —. *IEEE Standard 802.11.* s.l. : IEEE, 1999. ISBN 0-7381-5656-6.

25. —. *IEEE Draft Standard P802.11p.* [Documento versión D5.0] s.l. : IEEE, IEEE, 2008.

26. **IALA.** *IALA Technical Clarifications on Recommendation ITU-R M.1371-1",.* s.l. : IALA.

27. **Bernsen, James y Manivannan, D.** *Greedy Routing Protocols for Vehicular Ad Hoc Neworks.* s.l. : University of Kentucky.

28. **Karp, B. y Kung, H. T.** *GPSR: Greedy Perimeter Stateless Routing for Wireless Networks.* s.l. : MobiCom 200.

29. **Zhang, Wenhui.** GeoNet Design Goals and Requirements. *GeoNet Documents.* [En línea] 2009. [Citado el: 3 de marzo de 2010.] http://www.geonet-project.eu/?download=GeoNet_Design_Goals_and_Requirements.pdf.

30. *Geographic routing in city scenarios.* **Lochert, C., y otros.** 2005, ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 9, págs. 69-72.

31. **Miller, Michael.** *Discovering Bluetooth.* s.l. : Sybex Inc, 2001. ISBN-10 0782129722.

32. **GeoNet.** *D2.2 Final GeoNet Specification.* [Documento GeoNet-D.2.2-v1.1] [ed.] Andras Kovacs. s.l. : BROADBIT, 2010.

33. —. *D1.2 Final GeoNet Architecture Design.* [Documento GeoNet-D.1.2-v1.1] [ed.] Thierry Ernst. s.l. : INRIA, 2010.

34. **Naumov, Valery y Gross, Thomas.** Connectivity-Aware Routing (CAR) in Vehicular AdHoc Networks. *Laboratory for Software Technology.* [En línea] [Citado el: 3 de marzo de 2010.] http://www.lst.inf.ethz.ch/research/publications/INFOCOM_2007/INFOCOM_2007.pdf.

35. **COMeSafety.** COMeSafety Presentation. *COMeSafety.* [En línea] [Citado el: 3 de marzo de 2010.] http://www.comesafety.org/uploads/media/COMeSafety_Project_Presentation_01.pdf.

36. **Zhang, Wenhui.** Combining C2C-NET GeoNetworking and IPv6. *GeoNet Documents.* [En línea] 2009. [Citado el: 3 de marzo de 2010.] http://www.geonet-project.eu/?download=GeoNet-2009-11-03-C2C-CC_Forum.pdf.

37. Car-to-X Measurements. *NoW Final Workshop.* [En línea] 2008. [Citado el: 3 de marzo de 2010.] http://www.network-on-wheels.de/downloads/final-workshop/7-NoW_Measurements_Luebke.pdf.

38. **Car-2-Car Consortium.** *Car-2-Car Consortium Manifiesto.* [Documento versión 1.1] s.l. : Car-2-Car Consortium, 2007.

39. *BUSNet: Model and Usage of Regular Traffic Patterns in Mobile Ad Hoc Networks for Inter-Vehicular Communications.* **Wong, K., y otros.** 2003. ICT 2003.

40. Bluetooth Technology. *The Official Bluetooth Technology Info Site.* [En línea] [Citado el: 3 de mayo de 2009.] http://spanish.bluetooth.com/Bluetooth/Technology/Works/.

41. **Muller, Nathan J.** *Bluetooth Demystified.* s.l. : McGraw-Hill Professional Publishing, 2000. ISBN-10 0071363238.

42. **Kammer, David, McNutt, Gordon y Senese, Brian.** *Bluethooth Application Developer's Guide: The Short Range Interconnect Solution.* s.l. : Syngress, 2001. ISBN-10 1928994423.

43. **Beech, William A., Nielsen, Douglas E. y Taylor, Jack.** *AX.25 Link Access Protocol for Amateur Packet Radio.* [Documento versión 2.2] s.l. : Tucson Amateur Packet Radio Corp, 1998.

44. *A-STAR: A Mobile Ad Hoc Routing Strategy for Metropolis Vehicular Communications.* **Seet, B., y otros.** 2004. NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications. págs. 989-999.

45. **The APRS Working Group.** *APRS Protocol Reference.* [Document v 1.0.1] s.l. : Tucson Amateur Packet Radio Corp, 2000. ISBN 0-9644707-6-4.

46. **Crosswell, Alan.** APRS from de Bottom Up. *APRS maps & stuff.* [En línea] 30 de 12 de 2002. [Citado el: 9 de agosto de 2009.] http://www.users.cloud9.net/%7Ealan/ham/aprs/aprs.pdf.

47. **López Monge, Alejandro.** Aprendiendo a programar con Libpcap. *e-ghost.* [En línea] 2005. [Citado el: 11 de noviembre de 2009.] http://www.e-ghost.deusto.es/docs/2005/conferencias/pcap.pdf.

48. **National Institute of Standards and Technology.** *AODV Guide.* s.l. : U.S. Department of Comerce.

49. *An evaluation of inter-vehicle ad hoc networks based on realistic vehicular traces.* **Naumov, Valery, Baumann, Rainer y Gross, Thomas.** Florencia : ACM, 2006. 7th ACM international symposium on Mobile ad hoc networking and computing. págs. 108-119. ISBN 1-59593-368-9 .

50. *Ad-hoc on demand distance vector routing.* **Perkins, C. E. y Royer, E. M.** 1999. 2nd IEEE Workshop on Mobile Computing Systems and Applications. págs. 90-100.

51. *A study on the feasibility of mobile gateways for vehicular ad-hoc networks.* **Namboodiri, V., Agarwal, M. y Gao, L.** 2004. First International Workshop on Vehicular Ad Hoc Networks. págs. 66-75.

52. *A routing strategy for metropolis vehicular communications.* **Liu, G., y otros.** 2004. International Conference on Information Networking (ICOIN). págs. 134-143.

53. *A practical routing protocol for vehicle-formed mobile ad hoc networks on the roads.* **Wang, S. Y., y otros.** 2005. 8th IEEE International Conference on Intelligent Transportation Systems,. págs. 161-165.

54. **Pejman, Roshan y Leary, Jonathan.** *802.11 Wireless LAN Fundamentals.* 2003 : Cisco Press. ISBN 1-58705-077-3.

55. *The BSD Packet Filter: A New Architecture for user-level Packet Capture.* **McCanne, Steven y Jacobson, Van.** s.l. : Lawrance Berkeley Laboratory, 1992.

56. *IEEE 1609.4 DSRC Multi-Channel Operations and Its Implications on Vehicle Safety Communications.* **Chan, Qi, Jiang, Daniel y Delgrossi, Luca.** s.l. : Mercedez-Benz Research & Development North America, 2009.

57. **Green, Andy.** Packetspammer. *Penumbra WiFi Network.* [En línea] [Citado el: 14 de agosto de 2009.] http://penumbra.warmcat.com/.

ANEXO A. Informe detallado de tecnologías

A.1. APRS

El sistema APRS es un protocolo de comunicación de paquetes diseñado por Bob Bruninga quien lo introdujo en la conferencia de comunicación digital TAPR/ARRL en el año 1992. A pesar que en ese momento ya existía el packet radio para la transmisión y hasta repetición de mensajes digitales a través de enlaces de radio, APRS le introdujo una serie de aplicaciones que lo volvieron muy popular entre los radioaficionados, así como para su uso oficial por parte de agencias gubernamentales.

Una de las principales características que lo hacen destacar es la operación conjunta con el sistema de posicionamiento satelital GPS permitiendo así la transmisión de la posición del transceptor. Esta aplicación hizo que en un principio la sigla APRS fuera atribuida a Automatic Position Reporting System.

Sin embargo, APRS es un sistema mucho más amplio que tan sólo la transmisión de la posición. En APRS se logra una red global, donde todos los transceptores conectados comparten información de índole variada, teniendo cada uno de ellos la información completa de toda la red. El protocolo comunica los datos en la forma uno a muchos, razón por la cual todos los participantes logran obtener la información de toda la red.

Dentro de las aplicaciones de APRS encontramos:

- Ubicación de transceptores y diferentes objetos sobre mapas.
- Reportes de estaciones meteorológicas
- Mensajes punto a punto
- Transmisión de boletines y anuncios
- Acceso a Internet
- Direction finding

APRS hace uso de repetidoras digitales llamadas *digipeaters*, conocidas del tradicional packet radio. La gran diferencia está en la introducción del concepto de repetición genérica. Esto significa que las estaciones repetidoras se clasifican en función de sus características generales (una estación en lo alto de una montaña, o en el centro de una ciudad, o funcionando de Gateway con Internet, etc). Cuando un nodo APRS quiere enviar un paquete, no necesita conocer el camino específico de las repetidoras por donde debe pasar (digamos, la dirección absoluta). En cambio, sólo necesita saber qué tipo de repetidoras desea usar y el protocolo se encargará del resto. A modo de ejemplo, un nodo compuesto por una radio portátil generalmente necesitará usar una repetidora cercana en la ciudad para lograr que su mensaje sea retransmitido más allá de su pobre alcance. Sin embargo, un nodo fijo con una buena antena puede obviar la repetidora urbana, ya que por sí solo puede lograr el alcance necesario. En el caso del equipo portátil, sus mensajes estarán dirigidos a las repetidoras de tipo urbano, mientras que en el segundo los mismos estarán dirigidos a repetidoras más lejanas.

Para evitar que los paquetes se repitan indefinidamente en la red, el protocolo de repetición genérica incorpora contadores de saltos. Cada vez que una repetidora entiende que debe repetir un paquete, primero modifica este atributo y luego transmite al aire. Esto nos permite construir caminos de repetición de lo más variados, del estilo “repita n veces a través de repetidoras del tipo A y luego m veces en repetidoras del tipo B”.

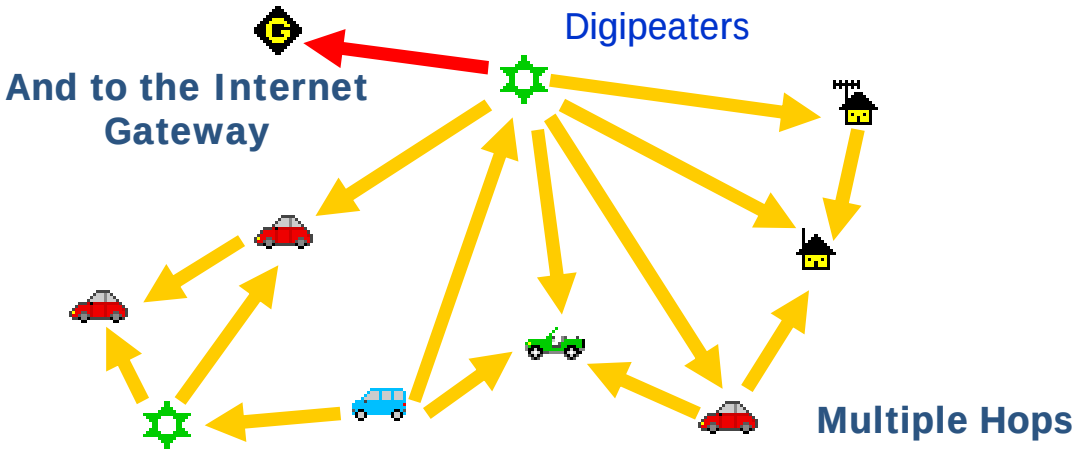

Figura A. 1 - Esquema de funcionamiento de saltos en APRS

El siguiente diagrama muestra cómo la combinación de las distintas redes que conviven en APRS logra que la información se transmita en forma global.

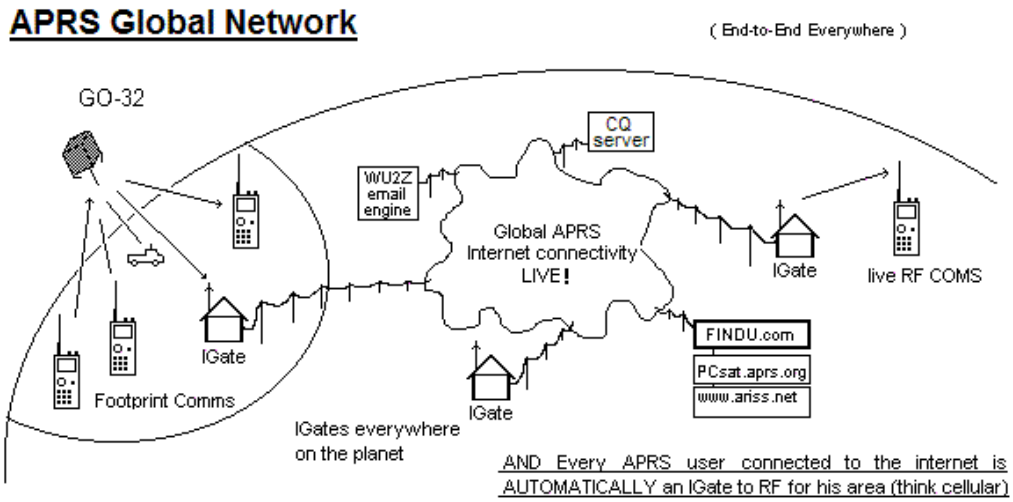

Figura A. 2 - Esquema de la red APRS

APRS maneja el concepto de tiempo de ciclado de la red para hacer referencia al tiempo que un nodo debe estar a la escucha hasta recibir al menos una vez a todos los nodos en alcance de radio. El objetivo es que dentro del alcance local, el tiempo de ciclado sea de 10 minutos. Por esta razón, las estaciones transmiten su posición al menos una vez cada 10 minutos, de forma también de mostrar que están activas en la red. Para evitar la congestión del canal, los mensajes que son enviados para ser repetidos en la red, tienen un tiempo de ciclado mayor llegando a los 30 minutos para los mensajes con más de 3 saltos. Esto significa, que en el peor de los casos nos tomará 30 minutos desde que se enciende el receptor para tener una imagen completa de todas las estaciones de la red.

A.1.1 Protocolo de capa de enlace

Como ya mencionamos, APRS parte del packet radio, donde el protocolo de capa de enlace utilizado es el AX.25 (descendiente amateur del X.25, protocolo sí estandarizado por la ITU). APRS lo conserva, utilizando únicamente las tramas UI y obteniendo de él un servicio no orientado a conexión y sin garantía de recepción, aunque intentando que sea libre de errores.

	AX.25 UI-FRAME FORMAT								
	Flag	Destination Address	Source Address	Digipeater Addresses (0-8)	Control Field (UI)	Protocol ID	INFORMATION FIELD	FCS	Flag
Bytes:	1	7	7	0–56	1	1	1–256	2	1

Figura A. 3 - Formato de la trama AX.25

• Flag — Esta bandera en cada extremo separa las tramas con la secuencia 0x7E
• Destination Address — Este campo puede contener el distintivo de llamada APRS del destinatario o simplemente datos. En el caso de datos, se le da el formato apropiado para que sea compatible con el estándar de distintivos de llamadas en AX.25 (ej: 6 caracteres alfanuméricos + SSID). Si el SSID es distinto de cero, entonces se está especificando un camino genérico de digipeaters APRS.
• Source Address — Este campo contiene el distintivo de llamada APRS y SSID del originador del mensaje. Si el SSID es distinto de cero, entonces se puede estar especificando un símbolo APRS.
• Digipeater Addresses — En este campo se pueden incluir desde cero hasta 8 distintivos de llamada de digipeaters. Estas direcciones son sobrescritas por los caminos genéricos de digipeaters que se puedan especificar en la zona de SSID del campo de origen.
• Control Field — Este campo se fija en 0x03 (trama UI)
• Protocol ID — Este campo se fija en 0xF0 (no hay protocolo de capa 3).
• Information Field — Este campo contiene más datos APRS. El primer caracter se usa para el Identificador de Tipo de Datos APRS, que especifica la naturaleza de los datos que van a continuación.
• Frame Check Sequence — Se incluye aquí una secuencia de 16 bits para chequear la integridad de la trama recibida.

APRS incorpora direcciones de destino genéricas, las cuales permiten que el mensaje sea copiado por todas las estaciones. De esta forma, lo que se indica en el campo de destino es el tipo de información que se está difundiendo, como ser posición, información de corrección diferencial, etc. De esta forma APRS brinda la posibilidad de tanto enviar mensajes punto a punto, como realizar la difusión a todas las estaciones en forma muy sencilla y eficiente.

El protocolo también define el formato de los datos a incluir en el Information Field de la trama AX.25. Como ya mencionamos, el primer caracter indica el tipo de datos. Dependiendo de este tipo de datos, APRS especifica la forma en la que se deben transmitir y los campos que lo componen. Esta especificación escapa al estudio de este informe.

A.1.2 Propagación de paquetes

Los nombres genéricos para las repetidoras, nos brindan mucha más flexibilidad a la hora de propagar los mensajes. En AX.25 puro, se debe especificar claramente los distintivos de llamada de las repetidoras por las que pasará el mensaje, conformando entonces un camino único y conocido desde el momento que se origina el mensaje. Con la introducción de los nombres genéricos, ya no es necesario conocer qué repetidora específica debe tomar el mensaje, sino qué tipo de repetidora. Esto nos da la posibilidad de propagar los paquetes sin conocer el distintivo de llamada específico de las repetidoras en cuestión.

Los nombres genéricos ("Alias") más comunes son RELAY y WIDE. Un nodo se identifica a sí mismo como RELAY si es capaz de repetir los paquetes que recibe. La mayoría de los nodos APRS responde a este alias. Al alias WIDE responden aquellas estaciones que tienen largo alcance y que están ubicadas en forma estratégica (en una montaña, etc).

Como describíamos en la sección anterior, la trama de APRS incluye además del destinatario (o destinatario genérico) el camino de repetidoras (en orden) por las que debe pasar. A modo de ejemplo, si un nodo que responde al alias RELAY recibe un paquete cuyo próximo salto es el alias RELAY, retransmitirá el paquete. Antes de retransmitir, cada repetidora modifica el encabezado del paquete y marca el alias al que respondió. De esta manera, los paquetes no se retransmiten infinitamente por la red y a su vez el camino se va recorriendo en el orden que figura en el encabezado.

Siguiendo con los ejemplos, supongamos un paquete cuyo camino de repetición es *RELAY,WIDE* originado por la estación CX-BGP. En el área de cobertura de CX-BGP tenemos las estaciones CX-DMLC y CX-FA84, que responden a los alias RELAY y WIDE respectivamente. Cuando CX-BGP envía inicialmente el paquete, tanto CX-DMLC como CX-FA84 lo reciben, pero dado que el camino se inicia con RELAY, sólo CX-DMLC lo repite modificando el encabezado para que el camino ahora se lea: *RELAY*,WIDE*. Con el *, se marca ese salto como efectuado. CX-FA84 vuelve a recibir el paquete y ahora sí reconoce que el próximo salto en el camino es WIDE, alias al cual él responde. Por lo tanto, lo repite modificando el camino para que se lea *RELAY*,WIDE*.* Si en este segundo salto (correspondiente a WIDE) un nodo RELAY recibe el paquete, no hará nada, pues el RELAY* indica que el salto ya se efectuó y no corresponde que los nodos RELAY sigan propagando.

Para indicar varios saltos del mismo tipo, se puede usar el alias WIDEN-n. En este alias se indica que el paquete debe ser retransmitido N veces por estaciones de tipo WIDE. En "n", se va decrementando un contador que comienza en N a medida que se efectúan los saltos hasta llegar a cero. Por ejemplo, un camino *WIDE2-2*, se convertirá en *WIDE2-1* en el primer salto, y en *WIDE2** en el segundo.

Las repetidoras mantienen un caché de los paquetes que han propagado de forma de no volver a repetirlos si vuelven a recibirlos en el futuro. A modo de ejemplo, en un paquete de *WIDE3-3*, es muy probable que la primer repetidora reciba el paquete *WIDE3-1* que retransmite el nodo siguiente, pero no retransmitirá, pues sabe que es un paquete que ya pasó por él.

A.1.3 Capa física

A pesar que AX.25 no define específicamente el protocolo de capa física a usar, se utiliza comúnmente AFSK para modular la información binaria de los paquetes. Este tipo de modulación en tonos de audio tiene la ventaja de poder ser usada con transceptores comunes, donde el filtro de audio (pensado para voz humana) deja pasar estas frecuencias.

En VHF se usan los tonos de 1200 Hz y 2200 Hz para modular los “unos” y “ceros” logrando una tasa de transmisión de 1200 baudios.

Las estaciones escuchan el canal de radio y transmiten sólo si lo identifican como libre e incorporan timers de retransmisión para el caso de que existan colisiones.

A.1.4 Conclusiones

APRS es un muy buen ejemplo de redes Ad-Hoc de largo alcance, en donde es posible propagar información en forma periódica así como realizar comunicaciones punto-punto. Su forma de resolver el camino de propagación de mensajes a partir de los alias es una excelente solución para el enrutamiento de paquetes en redes dinámicas, sin la necesidad de la gestión externa para definir los caminos.

En su forma actual de uso, APRS podría ser utilizado para la comunicación entre vehículos de mensajes con información, posición, y todo tipo de contenido de interés general, con una gran área de cobertura. Su bajo ancho de banda hace que esta solución sea un tanto limitada para aplicaciones de tiempo real, sobre todo cuando se tienen muchos vehículos participando en alcance de radio, pudiendo llegar rápidamente a la saturación del servicio. Mientras tanto, su método de propagación por alias puede ser incorporado a otras tecnologías.

A.2. STDMA

El sistema “Self-Organising Time Division Multiple Access” (STDMA) es un sistema de acceso al medio compartido inventado por el sueco Hakan Lans y patentado (bajo el nombre “Position Indicating System”) por la empresa GP & C Systems International AB de la cual es dueño.

Este sistema plantea una forma de acceso al medio compartido usando división en el tiempo y sin necesidad de un nodo central de coordinación. De ahí su nombre “Self-Organising” que hace referencia a la capacidad de los nodos de auto gestionar el acceso al medio, coordinando con el resto de los nodos de la red.

La clave de este sistema está en el uso de una referencia de tiempo muy precisa y común a todos los nodos, con la cual divide el tiempo en ranuras sincronizadas. Como base de tiempo se utiliza el sistema de posicionamiento global GPS, que provee una fuente de reloj muy precisa y disponible globalmente a través de sus satélites.

Con todos los nodos sincronizados a la misma referencia de tiempo, es posible coordinar la transmisión de cada uno en ranuras de tiempo distintas utilizando para ello un protocolo de reserva de uso del canal. De esta forma, cada nodo anuncia con anticipación sus intenciones de uso del canal, reservando ranuras para el futuro las cuales en la medida de lo posible serán respetadas por los demás nodos de la red de forma que en cada ranura sólo esté transmitiendo un nodo a la vez, evitando entonces las colisiones.

La misión principal de esta invención es la de transmitir la posición de cada nodo en forma de difusión y así lograr que todos los nodos de la red conozcan la posición de los demás. No obstante, el sistema también prevé la comunicación de mensajes arbitrarios punto a punto.

Otra aplicación que tiene el sistema es la de proveer a los móviles con una fuente secundaria de posición en caso de falla del receptor GNSS. Tomando los datos de posición de otros móviles, junto con la marca de la hora de transmisión de dicho mensaje, el móvil puede determinar su propia posición.

El STDMA está actualmente estandarizado por la Organización Marítima Internacional (IMO) y la Organización de Aeronáutica Civil Internacional (OACI) bajo los nombres AIS y VDL modo 4 respectivamente. En ambos casos éste sistema ha sido adoptado para diseñar sistemas de distribución de posición e información entre móviles, y entre móviles y estaciones fijas. Ambos estándares están actualmente en uso, siendo el AIS el más desarrollado y difundido debido a su obligatoriedad en determinada clase de embarcaciones. Esto motivó incluso una estandarización del AIS por parte de ITU.

A.2.1 Descripción general

En STDMA el tiempo se divide en tramas, las cuales a su vez se dividen en ranuras dentro de las cuales viaja la información que envían los nodos. Cada trama dura 60 segundos y su comienzo está sincronizado con la señal de PPS (pulso por segundo) del sistema GPS. Cada trama se divide en un múltiplo de 60 ranuras. Esto último es para que la señal de PPS siempre se de en el límite entre dos ranuras, y nunca en el medio de una ranura. La razón es que de esta forma se puede referenciar las tramas en base a un contador inicializado con la señal de PPS.

En caso de falla en la recepción del reloj del GPS (se bloquea la antena, por ejemplo), el sistema prevé que se tome la referencia de reloj de las transmisiones de los otros nodos. Esta fuente secundaria de sincronismo es suficiente para seguir participando del STDMA aunque puede generar una degradación considerable.

Con los nodos sincronizados bajo un mismo entramado, se coordina entre ellos el uso de cada ranura. La manera para lograr esto es mediante un sistema de reservas a futuro. Cada vez que un nodo envía un mensaje, finaliza el mismo con información sobre la ranura que utilizará en el futuro. De esta forma, todos los nodos en alcance de radio sabrán que no deben usar esa ranura.

Cada nodo mantiene una tabla en memoria con la información de las reservas hechas por los demás nodos. Con esta información determina qué ranuras están disponibles para usar y en consiguiente realiza sus propias reservas. Esto implica que cada vez que un nodo ingresa a la red, debe permanecer a la escucha por algún tiempo hasta recibir suficiente información que le permita decidir en qué ranura transmitir.

Esto no es suficiente para que no existan colisiones. En particular, puede ocurrir que dos móviles que en un comienzo no estaban en alcance de radio y estén usando las mismas ranuras, se acerquen lo suficiente como para que sus paquetes colisionen. Para evitar esto, el sistema prevé que los nodos cambien las ranuras seleccionadas en forma periódica.

También existe el modo de transmisión aleatoria, que transmite en las ranuras disponibles usando un algoritmo no adaptativo p-persistente. Esto permite que los nodos transmitan aún sin tener una reserva (por ejemplo al ingresar por primera vez a la red). Este tipo de transmisiones también se pueden hacer en forma retrasada, en donde el nodo escucha el comienzo de la ranura para determinar si está ocupada o no, y si está libre entonces comienza la transmisión unos milisegundos después.

El sistema también prevé que puedan existir estaciones bases con privilegios sobre los demás nodos. Estas estaciones pueden tomar el control de la asignación de ranuras, y lograr una asignación más eficiente. Se menciona que para que el sistema funcione correctamente, debe haber una carga no mayor al 75 %, para permitir el ingreso de nuevos nodos a la red.

A.2.2 Método de reservas

Cada nodo debe mantener una tabla con todas las reservas realizadas por los demás nodos para las próximas 4 tramas + 128 ranuras. Una vez recibida información sobre una nueva reserva, el nodo debe actualizar su tabla antes de que finalice la ranura siguiente a ese mensaje. En la tabla se debe identificar el nodo que realizó la reserva y qué tipo de reserva es. Esta información será utilizada para definir qué ranuras están disponibles para reservar. No sólo las ranuras no reservadas son las candidatas, sino que por ejemplo, una ranura reservada para una comunicación punto a punto entre B y C, estando B y C a una distancia suficientemente grande del nodo A, puede ser reservada por A para transmitir puesto que es poco probable que su transmisión afecte a la comunicación entre B y C (interferencia co-canal).

Con la información contenida en la tabla de reservas mencionada en el párrafo anterior, cada nodo crea su propia tabla de ranuras disponibles. Se intenta que esta tabla tenga una gran cantidad de ranuras disponibles para elegir, por lo que no sólo se utilizan las ranuras libres, sino que también se agregan ranuras reservadas siempre que se cumplan determinadas condiciones de protección contra la interferencia co-canal. Se define el parámetro Q4 que corresponde a la cantidad deseable de ranuras en la lista. Si la cantidad de ranuras no

reservadas es menor a Q4, entonces es este escenario donde se deben agregar a la lista ranuras reservadas. Si la cantidad de ranuras no reservadas es mayor a Q4, entonces no será necesario tomar esas medidas.

Para transmitir se seleccionará una ranura de la tabla de ranuras disponibles descrita en el párrafo anterior de forma que la probabilidad de seleccionar una ranura determinada sea igual a la de seleccionar cualquier otra ranura de la lista.

Si el mensaje que se tiene para transmitir tiene un largo mayor al de una ranura, se debe construir la tabla de ranuras disponibles tomando en consideración sólo aquellas ranuras que tiene contiguas otras ranuras disponibles, formando bloques del largo necesario para transmitir el mensaje.

A.2.3 Especificación de RF

Más allá que cualquier tipo de modulación podría ser usada en este sistema, se recomienda utilizar GMSK para modular la señal de RF en FM. A su vez, un bitrate de 9600 bps es mencionado como el más apropiado.

La modulación GMSK tiene la ventaja de permitir una mejor reutilización del canal dado que es menos susceptible a la interferencia por parte de señales GMSK de menor potencia. Esto es crucial para poder transmitir en ranuras reservadas por otros nodos que se encuentran a suficiente distancia.

La transmisión del paquete de información en sí, es en forma asíncrona dentro de la ranura y los bits son codificados utilizando NRZI con relleno de bits para secuencias de más de 5 “ceros” seguidos.

A.2.4 Ejemplo de aplicación: Automatic Identification System – AIS

Introducción al AIS:

El Sistema de Identificación Automática, AIS, es un sistema de comunicación diseñado para el ambiente marino a través del cual se cursan mensajes en formato digital a través de uno o más canales de VHF. Dichos mensajes podrán ser Buque-Buque, Buque-Tierra o Tierra-Buque.
La principal función del AIS es mejorar la seguridad de la navegación, cursando datos de posición, desplazamiento y características de las embarcaciones, así como información de ayudas a la navegación y facilidades en tierra.
El sistema está diseñado para tener un alto nivel de disponibilidad y para ser independiente de estaciones centrales de control.

Descripción:

El AIS utiliza “Self Organising Time Division Multiple Access” – STDMA a través de hasta dos canales de VHF en banda marina (típicamente 161.975 MHz y 162.025 MHz – canales 87B y 88B respectivamente).
El sistema está definido en la recomendación de la Unión Internacional de Telecomunicaciones ITU-R M.1371-1 y con sus frecuencias asignadas en ITU-R M.1084-3.
Un equipo AIS consta básicamente de:

- Dos receptores de VHF que monitorean ambas frecuencias en forma simultánea.
- Un transmisor general (por lo tanto se transmite en un canal a la vez).

- Un receptor de GPS.
- Interfaz con otros sensores del barco (de los cuales extrae la información para armar los mensajes en forma automática):
    - GPS de navegación
    - Indicador de rumbo
    - Indicador de velocidad

STDMA en AIS:

Cada canal de RF se divide en el tiempo en tramas de 1 minuto, donde cada trama contiene 2250 ranuras que funcionando a la velocidad de 9600 bps proveen 256 bits para cada mensaje.

El entramado está sincronizado con la base de tiempos del sistema de posicionamiento global GPS, por lo que es un requisito que las terminales de AIS cuenten con receptor de GPS para la sincronización. De esta forma, todas las estaciones usarán la misma base de tiempos para calcular las ranuras de cada canal. Sin perjuicio de lo anterior, el sistema define la contingencia para el caso de pérdida de señal GPS, donde se pasa a extraer la sincronización de la propia trama recibida.

Cada estación decide en qué ranura transmitir a partir de la información que extrae del propio canal, donde escucha constantemente determinando las ranuras que ya están en uso. Con esta información genera un grupo de ranuras candidatas para usar y de las cuales termina optando por una en forma aleatoria y comunicando su decisión al resto de las estaciones. Esto implica que cada receptor clasifica a las ranuras en:

- Libre – no está siendo usada por ninguna estación.
- Disponible – está siendo usada por otra estación, pero es candidata para ser reutilizada.
- Asignada por el propio equipo – es de las ranuras que el equipo usa.
- Asignada por otro equipo – y *no podría* ser reutilizada.

La cantidad de ranuras que un equipo usa en cada trama dependerá del intervalo entre mensajes. Estos pueden ir desde los 3 minutos hasta los 2 segundos. Dicho intervalo se varía dinámicamente en función del estado de navegación del barco (anclado o navegando, velocidad de navegación y si está cambiando de rumbo). De esta manera se optimiza el uso del canal, evitando ocupar ranuras con mensajes sin información.

Otra forma en la que se optimiza el uso del canal es separando los mensajes en categorías básicas que se transmiten con distintos intervalos:

- Información de navegación (posición, velocidad, rumbo, velocidad de cambio de rumbo, etc)
- Información estática (nombre del barco, identificación, tipo y dimensiones del barco, etc)
- Información dinámica (tipo de carga, ETA, etc)
- Etc

A modo de ejemplo, para obtener todos los parámetros estáticos de una embarcación, es necesario monitorear los canales AIS durante varios minutos ya que dicha información se va transmitiendo de a partes y con un intervalo bastante holgado.

Es evidente que el sistema tiene un tope en los mensajes que se pueden cursar, que aunque es suficientemente alto, puede ser alcanzado en zonas de gran afluencia marina. Para estos casos, existe un algoritmo de resolución de congestión a partir del cual los equipos de AIS toman prioridad sobre otros, así como optan por reutilizar ranuras de buques lejanos.

Utilizando ambos canales de VHF, tenemos un total de 4500 ranuras disponibles por minuto. Dado que los buques en movimiento reportan en promedio cada 3.33 segundos, esto nos da una capacidad local del sistema cercana a los 250 equipos AIS.

Especificación de RF en AIS:

Como ya mencionamos, AIS utiliza dos canales de VHF en forma simultánea. Los canales reservados para este motivo son el 87B y 88B de la banda marina, correspondiente a 161.975 MHz y 162.025 MHz respectivamente.

Con el uso de dos canales no sólo se duplica la capacidad del sistema, sino que se lo protege contra el efecto de interferencia en una de las frecuencias. Los equipos transmiten en forma alternada en los dos canales y monitorean ambos en forma simultánea.
Los bits se codifican primero utilizando NRZI y luego se convierten a GMSK para modular la señal de RF en FM con un bitrate de 9600 bps.

La norma establece que la máxima potencia de transmisión es de 12.5 W, con lo que se logra un alcance en alta mar de hasta 40 Millas Náuticas (75 Km), el cual es por supuesto variable en función de la altura de la antena VHF entre otros.

A.2.5 Conclusiones

A pesar de la gran funcionalidad de este sistema, no parece muy adecuado para su uso masivo en vehículos terrestres. 250 vehículos puede resultar un número bastante limitado tomando en cuenta la gran cobertura del VHF.

Por otra parte, a primera instancia parecería que el sistema no cuenta con protección para la recepción por múltiples caminos típica de los entornos urbanos. Esto podría devenir en una excesiva interferencia intersimbólica.

Otra limitación en la ciudad se tiene en la recepción de las señales GPS, que pueden llegar a bloquearse totalmente por períodos muy largos (edificios altos, túneles, estacionamientos, etc).

De todas formas, AIS puede ser una base muy útil para el diseño de un sistema urbano, aumentando la frecuencia de RF, incorporando intervalos de guarda, etc.

A.3. Bluetooth

Bluetooth es un protocolo de comunicación para redes WPAN (Wireless Personal Area Network), especialmente diseñado para equipos personales, de bajo consumo, con cobertura baja y basados en transceptores de bajo costo.

Tiene como principales objetivos posibilitar la transmisión de voz y datos entre diferentes dispositivos móviles o fijos prescindiendo de cables, mediante un enlace por radiofrecuencia globalmente libre (2,4 GHz), ofreciendo además la posibilidad de crear pequeñas redes inalámbricas de tipo ad hoc, en la cual los dispositivos pueden establecer por si solos la red, sin infraestructura extra (access points por ejemplo).

A.3.1 Descripción General

Bluetooth surge como una alternativa económica, confiable y fácil de usar, a la utilización de cables como principal forma de transmitir datos entre periféricos de todo tipo. Es el principal competidor de los protocolos de comunicación vía puerto infrarrojo, y en gran medida lo ha ido sustituyendo gracias a su capacidad de operar a distancias mayores y sin línea de vista. Esto, a pesar de tener mayor tiempo de conexión y un bit rate bastante menor por el momento (1 a 3 Mbps contra 16Mbps el IR en su nueva versión de alta velocidad).

El hardware que compone el dispositivo Bluetooth está compuesto por dos partes:

- un dispositivo de radio, encargado de modular y transmitir la señal
- un controlador digital, compuesto por una CPU, por un procesador de señales digitales (DSP - Digital Signal Processor) llamado Link Controller (o controlador de Enlace) y de las interfaces con el dispositivo anfitrión.

La capa física de radio (RF) Bluetooth opera en la banda de 2.4 GHz libre para ISM (banda de frecuencia industrial, científica y médica). El sistema emplea un mecanismo de ensanchamiento de espectro por salto de frecuencia (frequency hopping spread spectrum), en el cual se establece un orden pseudo aleatorio de cambio en la frecuencia portadora, el cual es conocido sólo por los integrantes en la subred. Esto ayuda a contrarrestar las interferencias y la pérdida de intensidad, además de ser un buen punto de partida para asegurar una transmisión segura. Para minimizar la complejidad del transmisor, se utiliza una modulación de frecuencia binaria. La tasa de transferencia de símbolos es de 1 MS/s (megasímbolos por segundo), que admite una velocidad de transmisión de 1 Megabit por segundo (Mbps) en el modo de transferencia básica y una velocidad de transmisión total de 2 a 3 Mbps en un modo de transferencia de datos mejorada.

En condiciones típicas de operación, el canal de comunicación es compartido por un grupo de dispositivos sincronizados a un reloj común y un patrón de saltos entre portadoras. Un dispositivo denominado maestro provee la referencia de sincronización al resto de los dispositivos denominados esclavos. La red compuesta por maestro y esclavos se le denomina piconet y constituye la forma fundamental de comunicación de la tecnología Bluetooth.

Los dispositivos en una piconet usan un patrón de saltos de frecuencia específico, que es determinado por campos en la dirección del dispositivo y reloj del maestro. El patrón de saltos básico es un ordenamiento pseudoaleatorio de las 79 frecuencias disponibles en la banda ISM, aunque pueden ser excluidas aquellas portadoras cuya porción del espectro presente interferencia.

Se realiza un ranurado del tiempo en slots de 625 µs de duración. Los datos son transmitidos entre dispositivos en paquetes que son colocados en estas ranuras. Cuando las circunstancias lo permiten, un paquete puede tomar varios slots consecutivos. Los saltos de frecuencia ocurren entre la transmisión o recepción de paquetes. El estándar provee el efecto de una transmisión full dúplex a través del uso de TDD (Time-Division Duplex). También se provee de dos mecanismos distintos de conexión conocidos como Circuit Switching y Packet switching. En el primero, se establece un camino fijo durante la transmisión, similar a lo que sucede con una llamada telefónica, y especialmente diseñada para la transmisión de voz en tiempo real. Una vez terminada la transmisión, se liberan recursos para que otra conexión pueda tomar su lugar. En el segundo mecanismo, más conocido en el mundo de la Internet, permite utilizar los recursos en forma más eficiente pero sacrificando la garantía de disponibilidad de los mismos. Este último es el más propicio a la hora de intercambiar datos.

La arquitectura bluetooth sigue un modelo de capas del tipo OSI. Esto permite separar funcionalidades y promover la interoperabilidad entre productos de diferentes proveedores. Se distinguen 3 capas principales dentro de lo que se denomina el controlador bluetooth: la capa de radiofrecuencia, la de banda base, y la capa de gestión de enlace. Existen dos capas más por encima de éstas que conforman lo que se denomina el host bluetooth: la capa de control de enlace lógico L2CAP, y una última capa donde se encuentran las aplicaciones de más alto nivel. Diferentes stacks de protocolos son utilizados para diferentes aplicaciones. Además puede ocurrir que ciertos mensajes que se transmiten de una capa superior a una inferior no respeten con rigurosidad el modelo de capas y hablen directamente salteándose alguna capa intermedia. Por ejemplo, los datos de usuario de la capa L2CAP muchas veces son mandados directamente hacia la capa de banda base sin pasar por la capa de enlace, lo que no ocurre con los mensajes de control. Sin embargo, independientemente de la aplicación que se trate, el stack utilizado siempre contiene los protocolos comunes de manejo de enlace físico y enlace lógico de las capas de banda base y gestión de enlace.

Cada funcionalidad del estándar bluetooth requiere determinado stack de protocolos, lo que se encuentra detalladamente especificado en los “perfiles bluetooth”. Un perfil bluetooth es una descripción técnica de cómo hacer que una determinada aplicación realmente funcione. Los perfiles son extremadamente técnicos y se extienden lo suficiente como para describir todos los protocolos y procedimientos necesarios para ejecutar las acciones especificas necesarias para implementar una determinada actividad.

Por último, la transmisión de datos en bluetooth se realiza de una forma muy segura. Además de su limitado rango de alcance y el empleo de FHSS, que hace que la intercepción de la comunicación sea extremadamente difícil en primer lugar, la especificación bluetooth emplea funciones de autenticación y encriptado, que aseguran la privacidad y la integridad en las comunicaciones.

A.3.2 Radio Bluetooth

Los dispositivos Bluetooth funcionan en la banda de 2,4 GHz, una de las bandas de radio ISM (industrial, científica y médica) que no requieren licencia. Se utiliza FHSS para contrarrestar las interferencias y la pérdida de intensidad.

Se definen dos modos de modulación: Un modo obligatorio, llamado modo de transferencia básica, que usa una modulación de frecuencia binaria para reducir al mínimo la complejidad del transmisor/receptor. Existe un modo opcional, llamado de transferencia de datos mejorada, que usa modulación PSK y cuenta con dos variantes: π/4-DQPSK y 8DPSK. La tasa de transferencia de símbolos de todas las secuencias de modulación es de 1 Ms/s. La velocidad de transmisión aérea total es de 1 Mbps con el modo de transferencia básica; 2 Mbps con la transferencia de datos mejorada y π/4-DQPSK; y 3 Mbps con la transferencia de datos mejorada y 8DPSK. La cabecera de los paquetes hace que el bitrate de información caiga 1/3 del bitrate neto, en el mejor de los casos.

Para la transmisión bidireccional se emplea una técnica de dúplex por división de tiempo (TDD) en ambos modos.

Los 79 canales RF se organizan por números, de 0 a 78, con un espacio de 1 MHz entre ellos, empezando por 2402 GHz.

Alcance reglamentario	Canales RF
2,400-2,4835 GHz	f = 2402 + k MHz, k=0,…,78

Para satisfacer la reglamentación relativa a las transmisiones fuera de banda de cada país, se utiliza una banda de guarda en los extremos inferior y superior de la gama de frecuencias.

Banda de guarda inferior	Banda de guarda superior
2 MHz	3,5 MHz

Los dispositivos se clasifican en 3 clases según la potencia de transmisión:

Potencia Clase	Potencia de salida máxima (Pmax)	Potencia de salida nominal	Potencia de salida mínima*	Control de potencia
1	100 mW (20 dBm)	No corresponde	1 mW (0 dBm)	Pmin<+4 dBm a Pmax Opcional: Pmin** a Pmax
2	2,5 mW (4 dBm)	1 mW (0 dBm)	0,25 mW (-6 dBm)	Opcional: Pmin** a Pmax
3	1 mW (0 dBm)	No corresponde	No corresponde	Opcional: Pmin2** a Pmax

* Potencia de salida mínima con el máximo ajuste de potencia.
** El límite inferior de potencia Pmin<-30dBm es el recomendado, pero no es obligatorio, y puede elegirse según las necesidades de la aplicación.

A pesar de que los dispositivos de clase 1 son los que tienen mayor alcance, son los de clase 3 los que se han extendido masivamente.

Los dispositivos de clase 1 disponen de control de potencia. El control de potencia se usa para limitar la potencia transmitida por encima de +4 dBm. Por debajo de +4 dBm, la capacidad para controlar la potencia es opcional, y puede emplearse para controlar el consumo energético y el nivel global de interferencias.

La precisión de la frecuencia central debe ser de ±75 KHz con respecto a la frecuencia central. Durante la transmisión del código de acceso y la cabecera del paquete, se usa la secuencia de modulación GFSK de transferencia básica. Durante la transmisión de la secuencia de sincronización, la carga útil y la secuencia de cola se usa un tipo de modulación PSK con una velocidad de transmisión de 2 Mbps, u, opcionalmente, 3 Mbps.

Características del receptor
El nivel de sensibilidad real se define como el nivel de entrada para el cual se satisface un porcentaje de error de bit (BER) del 0,1%. Para cualquier transmisor Bluetooth, la sensibilidad del receptor será de –70 dBm o inferior.

La interferencia co-canal y las interferencias adyacentes en los canales de 1 y 2 MHz se miden con la señal útil 10dB sobre el nivel de sensibilidad de referencia En el resto de frecuencias, la señal útil debe estar 3 dB sobre el nivel de sensibilidad de referencia.

A.3.3 Bloques fundamentales de la tecnología

El estándar bluetooth define cuatro bloques funcionales principales que determinan la arquitectura del dispositivo bluetooth: en el controlador bluetooth propiamente dicho se encuentran el radiotransmisor bluetooth (RF), el controlador de enlace (LC), y el gestor de enlace (LM), y formando parte del host se encuentran las capas superiores (L2CAP) y de aplicación. Existe además un bloque funcional denominado controlador host (HC) que comunica al host con el controlador bluetooth a través de una interfaz estándar (HCI).

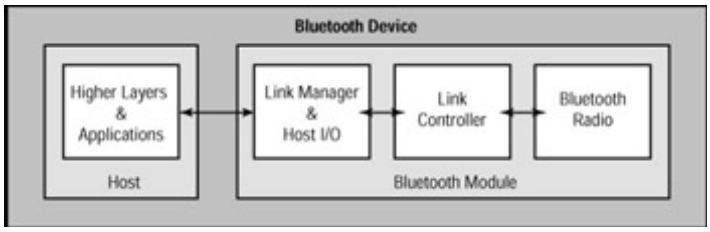

Figura A. 4 - Diagrama de bloques del dispositivo Bluetooth

Radio bluetooth

Como se explicó anteriormente, el radiotransmisor bluetooth transmite en la banda de 2.4GHz, utilizando FHSS y TDD, transmite una potencia de 1 mW con un alcance aproximado de 10 metros (para dispositivos de clase 3). El bloque RF es responsable de transmitir y recibir paquetes de información del canal físico.

Gestor de enlace y controlador de enlace

La conexión entre dos dispositivos bluetooth es llevada a cabo por el gestor de enlace (LM) y el controlador de enlace (LC). El LM esta implementado en software y lleva a cabo el establecimiento del enlace, autenticación, configuración, entre otras tareas necesarias para establecer la conexión. En esencia, LM descubre otros dispositivos corriendo el mismo software LM, y luego se comunica con ellos a través del protocolo LMP (Link Manager Protocol). Para realizar estas tareas, el software LM utiliza los servicios provistos por el controlador de enlace (LC). Este, facilita el envío y recibimiento de datos, establecimiento de la conexión, y otras tareas relacionadas.

Una vez establecido el enlace entre dos dispositivos bluetooth, el LM de cada unidad se comunica con la otra a través del protocolo LMP. Los mensajes intercambiados a este nivel se denominan PDUs (Protocol Data Units). Estos mensajes son clave para mantener el enlace por lo que tienen prioridad sobre los datos de usuario o de control proveniente de capas superiores.

Existen 55 PDUs diferentes definidas en la especificación bluetooth. Estas PDUs son instrucciones de alto nivel, que luego serán ejecutadas por el LC, por ejemplo, LMP_encryption _mode_req, utilizada para exigir modo encriptado en las transmisiones. Los siguientes son algunos de las posibles funciones de los mensajes de control:

Control de la conexión

- Establecimiento de la conexión
- Desconexión
- Control de energía
- Salto adaptable de frecuencia
- Cambio de velocidad de transmisión dictado por la calidad del canal (CQDDR)
- Calidad de servicio (QoS)
- Parámetros de programa de paginación
- Control de paquetes multi-ranura
- Transferencia de datos mejorada (EDR)
- PDU LMP encapsuladas

Seguridad

- Autenticación
- Emparejamiento
- Cambio de clave de enlace
- Cambio de tipo de clave de enlace actual
- Cifrado
- Solicitud de tamaño de clave de cifrado compatible
- Emparejamiento simple seguro

Solicitudes de información

- Precisión de la sincronización
- Compensación (offset) de reloj
- Versión LMP
- Funciones compatibles
- Solicitud de nombre

L2CAP

El estándar bluetooth define una capa por encima de LM, llamada L2CAP (Logical Link Control and Adaptation Protocol), que entra en juego una vez establecido el enlace vía LMP entre dispositivos. L2CAP lleva a cabo una variedad de funciones de alto nivel incluyendo multiplexado de protocolos, segmentación y re ensamblado de paquetes, y funciones de calidad de servicio (QoS). Además, L2CAP es capaz de hacer de interface con otros protocolos de comunicación, incluyendo SDP, TCS-BIN, y RFCOM, y a través de este último, protocolo que emula una conexión serie RS-232, es capaz de establecer enlaces PPP, y comunicarse a través de los protocolos TCP/IP, y obtener por ejemplo servicios WAP y WEP, entre otros.

Mientras el protocolo LMP del gestor de enlace, utiliza PDUs, la capa L2CAP se comunica a través un tipo de mensaje propio llamado evento, y también representan instrucciones de alto nivel, que serán interpretadas por LM, y luego por LC. El protocolo L2CAP es capaz de comunicarse en paralelo al protocolo LMP, con el controlador de enlace, para transferir datos de usuario.

L2CAP utiliza únicamente el enlace ACL de la capa de banda base. Las aplicaciones de audio y telefonía se comunican directamente con la capa de banda base sin pasar por el gestor de enlace, y utilizan enlaces SCO, aunque existe una ACL que es capaz de manejar información de audio en forma de paquetes (packetized audio data) como telefonía IP.

A.3.4 Capas y arquitectura de transporte de datos

Como se dijo anteriormente, el estándar bluetooth define un modelo de capas jerárquico, donde se reconocen las capas de RF, banda base, gestor de enlace, y gestor de recursos L2CAP, además de existir una capa de más alto nivel donde se encuentran las aplicaciones específicas asociadas a cada perfil bluetooth. Para cada capa se define un protocolo de comunicación: protocolo de RF, protocolo LC, LMP, y L2CAP. La estructura de capas, así como también los principales bloques funcionales de la arquitectura y los protocolos de comunicación de cada capa se muestran en la figura A. 5.

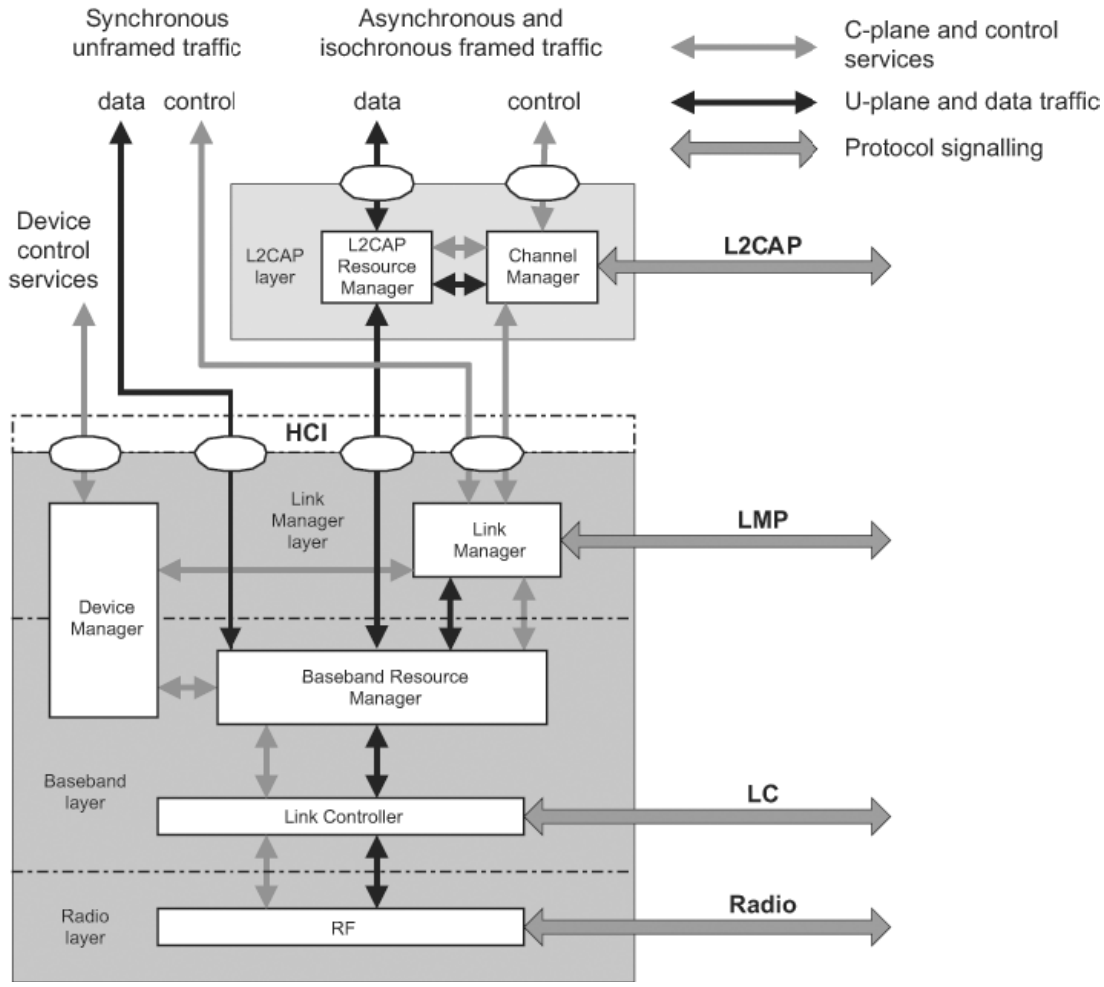

Figura A. 5 - Diagrama de capas de Bluetooth

Los niveles inferiores de la pila de protocolos constituyen el controlador Bluetooth, que contiene los bloques fundamentales de la tecnología, sobre los cuales se apoyan los niveles superiores y los protocolos de aplicación. Este componente está estandarizado y puede interactuar con otros sistemas Bluetooth de más alto nivel, aunque la separación entre ambas entidades no es obligatoria.

A través de cada capa se utiliza una variedad de enlaces y canales, que sirve de abstracción de las capas inferiores, y que siguen la siguiente jerarquía: canal físico, enlace físico, transporte lógico (también llamado comunicación lógica), enlace lógico, y canal L2CAP.

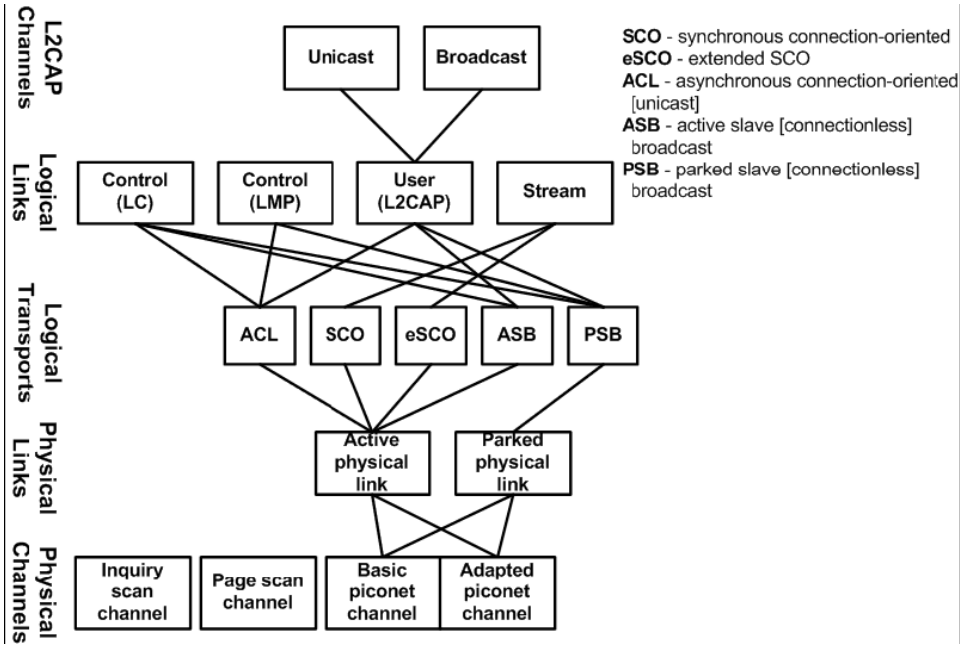

Figura A. 6 - Entidades de transporte en Bluetooth

A nivel de RF, se establecen los denominados canales físicos. Estos quedan determinados mediante una secuencia de saltos pseudo aleatorios en la portadora, la sincronización de los paquetes (ranuras) y un código de acceso al inicio del paquete indicando el tipo de canal físico. Existen 4 tipos de canal: canal de búsqueda (Inquiry), canal de paginación (Page), canal piconet básico, canal piconet adaptado, cuyas características se detallan más adelante.

Sobre el canal físico, se establece lo que se conoce como enlace físico. Éste representa una conexión de banda base punto a punto entre dos dispositivos y está siempre presente cuando un esclavo está sincronizado en la piconet. Un enlace físico esta siempre asociado a un canal físico. La identificación de los enlaces físicos no se realiza con parámetros propios de los enlaces, sino por su correspondencia con la identificación de la comunicación lógica utilizada, por lo que no existe un campo en el paquete que indique el enlace físico. El enlace puede estar activo o en modo de espera (park). Un enlace físico está activo cuando existe una comunicación lógica particular denominada comunicación lógica ACL predeterminada entre los dispositivos, y pasa a modo de espera cuando se está sincronizado a la piconet pero no se tiene una comunicación lógica ACL predeterminada.

Sobre el enlace físico se encuentra la comunicación lógica (o transporte lógico), y sobre ésta, el enlace lógico. Existe una variedad de enlaces lógicos que soportan diferentes requerimientos de transporte de los datos proveniente de la capa de aplicación. Cada enlace lógico está asociado con una comunicación lógica (o transporte lógico), que tiene un número de características determinadas que incluyen control de flujo, mecanismos de reconocimiento y repetición de paquetes, y números de secuencia, entre otras. El enlace físico se utiliza como medio de comunicación para uno o más enlaces lógicos que admiten tráfico síncrono, asíncrono e isócrono de unidifusión, y tráfico de difusión.

Pueden establecerse diferentes tipos de comunicaciones lógicas entre el dispositivo maestro y los dispositivos esclavos. Se han definido cinco comunicaciones lógicas:

- Comunicación lógica por conexión síncrona (SCO)
- Comunicación lógica por conexión síncrona ampliada (eSCO)
- Comunicación lógica por conexión asíncrona (ACL)
- Comunicación lógica por difusión del dispositivo esclavo activo (ASB)
- Comunicación lógica por difusión del dispositivo esclavo en espera (PSB)

Se definen 5 enlaces lógicos que son utilizados para control e información de usuario que corren sobre las comunicaciones lógicas:

- Link Control (LC)
- Link Manager (LM)
- User Asynchronous (UA)
- User Isochronous (UI)
- User Synchronous (US)

Los canales LC y LM son canales usados en las capas de control de enlace (L2CAP) y gestión de enlace respectivamente. UA, UI, y US son usados para transmitir información de usuario mediante el timming correspondiente. El canal LC se transporta en el encabezado de los paquetes, mientras que el resto de los canales viaja en la carga útil. El canal US viaja en enlace SCO, mientras que UA y UI viajan normalmente en ACL.

A través de los enlaces lógicos, además de los datos del usuario se transporta el protocolo de control de la banda base y las capas físicas, denominado protocolo de gestión de enlace (LMP). Los dispositivos que están activos dentro de la piconet tienen una comunicación lógica asíncrona predeterminada para el transporte de la señalización del protocolo LMP. Es lo que se conoce como comunicación lógica ACL. Esta comunicación es la que se establece cuando un dispositivo se une a una piconet. Se pueden crear comunicaciones lógicas adicionales si resulta necesario para transportar el flujo de datos síncronos.

El gestor de enlaces se sirve del protocolo LMP para controlar el funcionamiento de los dispositivos en la piconet y proporcionar servicios de gestión en las capas inferiores de la arquitectura: capa de radio y de banda base. El protocolo LMP sólo se transporta a través de la comunicación lógica ACL y la comunicación lógica de difusión predeterminadas.

Sobre el enlace lógico, se establecen los canales L2CAP que hace de interface entre los protocolos de aplicaciones y servicios, y la capa L2CAP. Estos canales se identifican mediante el identificador de canal (CID) que es asignado por la capa L2CAP.

A.3.5 Topología Piconet

Cuando dos dispositivos Bluetooth establecen una conexión, se crea un tipo de PAN (Personal Area Network) llamado Piconet. Un dispositivo en la piconet cumple el rol de maestro, mientras que el resto se denominan esclavos. En una piconet, el maestro controla el acceso al canal y provee la referencia de sincronización al resto de los dispositivos, que incluye no solo la referencia temporal sino también el patrón de saltos de la portadora. El reloj que se utiliza en el conjunto de la piconet es idéntico al reloj bluetooth del dispositivo maestro. Los esclavos calculan el offset necesario para agregar a su reloj interno y así, sincronizarse. En cuanto a la secuencia de salto, ésta se deriva del reloj y de la dirección del dispositivo maestro.

Pueden estar activos hasta 8 dispositivos incluyendo al maestro en una misma piconet, aunque más esclavos pueden estar conectados a la red pero en estado PARK, en el cual no están

activos para usar el canal pero permanecen sincronizados al maestro el cual los puede activar (desactivando otro esclavo activo), ahorrando de esta manera recursos y tiempo.

En una piconet se establecen enlaces de capa física entre los dispositivos, pero existen restricciones sobre estos enlaces. Los esclavos no se comunican directamente entre sí, sino que crean enlaces sólo con el maestro con el cual intercambian datos. Por otro lado, el maestro puede intercambiar el rol de maestro con cualquier esclavo de la piconet de ser necesario.

Piconets que tienen dispositivos en común conforman redes más grandes denominadas scatternet. Aunque las diferentes piconet en una scatternet no están sincronizadas, un dispositivo puede participar en diferentes piconet mediante TDM (Time Division Multiplexing). De esta manera la unidad participa secuencialmente en diferentes piconets estando activa en una sola a la vez. Cada piconet puede tener sólo un maestro asociado a la misma, pero puede haber esclavos pertenecientes a dos o más piconet distintas. Además, un esclavo en una piconet puede ser maestro en otra piconet.

Las conexiones en piconets pueden ser point-to-point (2 dispositivos), point-to-multipoint (hasta 8 dispositivos), además de poder conformar scatternets, como se muestra en la figura:

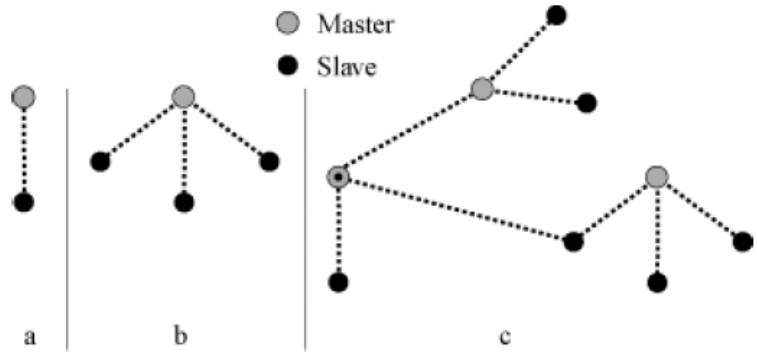

Figura A. 7 - Piconets

A.3.6 Banda base y detalles de conexión

La capa de banda base posibilita el enlace físico de RF entre los distintos dispositivos de una piconet. Dado que el sistema RF utiliza FHSS, en donde los paquetes se trasmiten en slots definidos de tiempo y a través de una sucesión pseudo-aleatoria de portadoras definidas, esta capa usa procedimientos de búsqueda y paginado para sincronizar las frecuencias de transmisión y los relojes de los diferentes dispositivos Bluetooth.

La comunicación entre los dispositivos se realiza, a nivel de banda base, a través de comunicaciones lógicas que pueden ser orientados a conexión o no, según los requerimientos de la aplicación, y se denominan enlace SCO (Synchronous Connection-Oriented) y enlace ACL (Asynchronous Connectionless) respectivamente, además de existir otras que surgieron luego: eSCO, ASB, y PSB. En cada comunicación, se establece por defecto el denominado enlace ACL predeterminado, el cual es utilizado por capas superiores para administrar la comunicación. Luego podrán establecerse otros enlaces físicos. Cada enlace físico está asociado a un canal físico.

La capa de banda base controla las operaciones sobre bits y paquetes, realiza detección y corrección de errores, broadcast automático y cifrado como sus labores principales. También emite confirmaciones y peticiones de repetición de las transmisiones recibidas.

La siguiente figura muestra el formato de un paquete estándar, donde se puede ver los encabezados correspondientes a los protocolos de las diferentes capas, así como la identificación de los canales y enlaces utilizados:

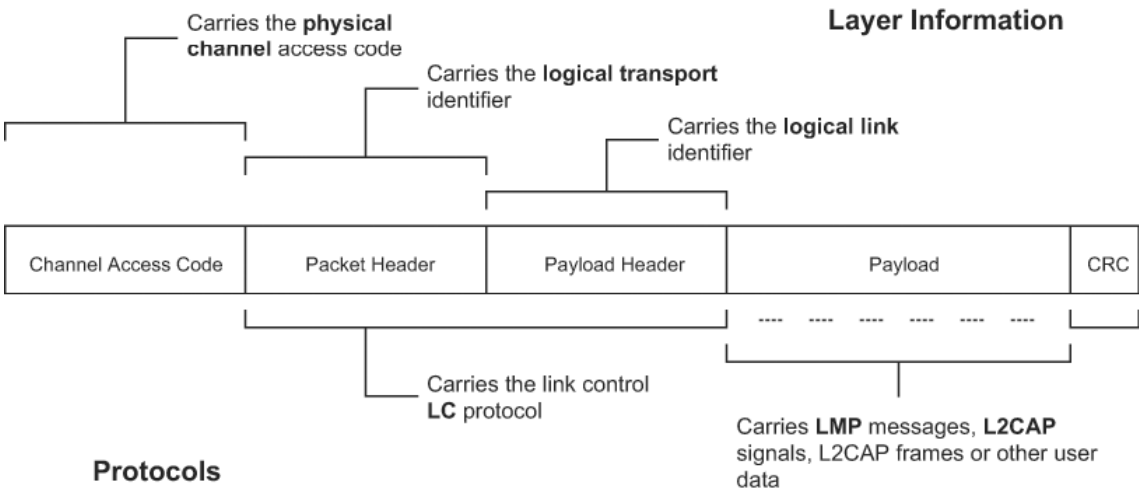

Figura A. 8 - Encabezados Bluetooth

Normalmente los paquetes incluyen sólo los campos necesarios para representar las capas utilizadas en la transacción. Por ejemplo, una simple búsqueda de dispositivos (Inquiry) no crea ni requiere un enlace lógico o alguna otra entidad superior de transporte, por lo que el paquete consistirá únicamente en el código de acceso del canal físico. Generalmente las comunicaciones normales en una piconet hacen uso de todos los campos.

Cada paquete transmitido comienza con un condigo de acceso, que consta de 74 bits, y que determina el tipo de canal físico utilizado. Existen tres tipos de código:

- código de acceso del canal (CAC) – Identifica una piconet. Cada paquete intercambiado en una misma piconet comienza con este código.

- código de acceso del dispositivo (DAC) – Es utilizado en el procedimiento de paginado y en la respuesta recibida al mismo.

- código de acceso de búsqueda (IAC) – Es utilizado en el procedimiento de búsqueda (Inquiry) de los dispositivos que están en el rango de alcance.

Luego del código de acceso viene el packet header que consta de 54 bits, el cual contiene información de identificación de la comunicación lógica y el enlace lógico utilizado. No hay ningún campo que contenga información relativa al enlace físico. Esta información va implícita en un campo llamado LT_ADDR en el packet header. Este campo es utilizado por el dispositivo receptor para determinar si el paquete es direccionado al dispositivo. El packet header, además de contener el LT_ADDR, contiene parte de los mensajes de controlador de enlace. El payload header contiene el identificador de enlace lógico (LLID) utilizado para enrutar el payload, además de contener la longitud del payload y, en ciertas ocasiones, un CRC usado para chequeo de errores.

Por último, el packet payload consta de 0 a 2745 bits, donde viajan los datos de usuario.

Todos los dispositivos Bluetooth cuentan con un reloj nativo que debe derivarse de un reloj del sistema de libre funcionamiento. Para la sincronización con otros dispositivos se utilizan compensaciones (offsets) que, cuando se suman al reloj nativo, proporcionan relojes Bluetooth temporales sincronizados entre sí. El maestro siempre es la referencia a sincronizarse.

A cada dispositivo Bluetooth se le asigna una dirección de dispositivo Bluetooth única de 48-bit (BD_ADDR) proporcionada por la autoridad reguladora de IEEE, y es utilizada a la hora de establecer la conexión, para identificar a los diferentes dispositivos.

Canales físicos

Los canales físicos se definen mediante una secuencia de saltos pseudo aleatorios en los canales RF, la sincronización de los paquetes (ranuras) y un código de acceso. La secuencia de saltos se determina a partir de la dirección del dispositivo Bluetooth y de la secuencia de saltos seleccionada. La fase de la secuencia de saltos se determina mediante el reloj Bluetooth. Todos los canales físicos se subdividen en ranuras de tiempo cuya longitud depende del canal físico. Existen 4 tipos de canales físicos diferentes, a través de los cuales se realizan diferentes funciones.

Canal físico básico de la piconet

El canal básico de la piconet permite la comunicación entre los dispositivos conectados en circunstancias de funcionamiento normales. Se distingue por una secuencia de saltos pseudo aleatorios en los canales RF. La secuencia de salto es exclusiva de la piconet y está condicionada por la dirección del dispositivo *Bluetooth* maestro, que determinará la fase de esta secuencia de acuerdo con su reloj. Todos los dispositivos *Bluetooth* conectados a la piconet sincronizan sus frecuencias de salto y relojes con el canal.

En el canal básico de la piconet, el maestro controla el acceso. Para ello, empieza a transmitir datos sólo en las ranuras de tiempo pares. Los paquetes transmitidos por el maestro se sitúan en el principio de la ranura y establecen la sincronización de la piconet. Por otra parte, estos paquetes podrán ocupar hasta cinco ranuras dependiendo del tipo al que pertenezcan.

Las transmisiones de los dispositivos maestros consisten en paquetes de información enviados a través de comunicaciones lógicas. A modos de respuesta, los dispositivos esclavos podrán transmitir datos en el canal físico. La forma en que esto se haga dependerá de la comunicación lógica a la que vaya dirigida la respuesta. Por ejemplo, en la comunicación lógica destinada a la conexión asíncrona, el dispositivo esclavo receptor responde enviando un paquete para la misma comunicación lógica, que en principio se encuentra al inicio de la siguiente ranura (con número impar).

Una característica especial del canal físico básico de la piconet es el uso de ranuras reservadas para transmitir balizas o beacons. Estos paquetes de control se utilizan si hay esclavos conectados al canal físico de la piconet en modo park o de espera. En este contexto, el maestro transmite un paquete por medio de las ranuras reservadas para las balizas y el esclavo utiliza la información transmitida para volverse a sincronizar con el canal físico de la piconet.

Un canal básico de piconet puede estar compartido por tantos dispositivos *Bluetooth* como permitan los recursos disponibles del dispositivo maestro. En la piconet sólo habrá un maestro,

el resto de dispositivos serán los llamados esclavos. Todas las comunicaciones se realizan entre el maestro y los dispositivos esclavos, ya que los esclavos no pueden comunicarse entre sí directamente en el canal de la piconet.

Canal físico adaptado de la piconet

El canal adaptado de la piconet se diferencia del básico en dos aspectos. En primer lugar, los esclavos transmiten en la misma frecuencia que el maestro que les precede. Es decir, la frecuencia para el envío posterior de paquetes esclavos no se vuelve a calcular. En segundo lugar, el canal adaptado no tiene por qué utilizar las 79 frecuencias disponibles. Puede excluir algunas del patrón de salto marcándolas como “sin uso” y utilizar tan sólo el resto de las frecuencias. La opción de utilizar un canal adaptado o uno básico para la comunicación, tiene en cuenta, entre otros factores, la gestión de recursos y la calidad de servicio.

Canal físico de búsqueda (Inquiry)

La detección de los dispositivos se realiza a través de un canal de búsqueda. Un dispositivo susceptible de ser detectado recibe una solicitud de búsqueda en su canal correspondiente y envía las respuestas apropiadas. Cuando un dispositivo está en modo de detección, realiza una serie de iteraciones o saltos pseudo aleatorios en todas las frecuencias del canal de búsqueda. Así, envía solicitudes de búsqueda en cada frecuencia y permanece atento a las posibles respuestas.

Los canales de búsqueda siguen un patrón de salto más ralentizado, en un número reducido de frecuencias de salto, para facilitar el proceso, y emplean un código de acceso para distinguir radiofrecuencias ocupadas momentáneamente por dos dispositivos próximos y que utilizan canales físicos diferentes.

Canal de paginación (Page)

Un dispositivo susceptible de ser conectado está preparado para aceptar conexiones y, para ello, se adentra en un canal de paginación. Este dispositivo recibe las solicitudes de conexión en su canal correspondiente, tras lo cual inicia una secuencia de intercambio de datos con el dispositivo emisor. Las iteraciones o saltos producidos en las frecuencias del canal de paginación, a la hora de conectar los dispositivos, se realizan de forma pseudo aleatoria enviando solicitudes de conexión a las distintas frecuencias, para, a continuación, quedar a la espera de recibir respuesta.

Procedimiento básico de conexión

Todos los dispositivos Bluetooth comienzan inicialmente en estado Standby. Cuando una unidad censa otro dispositivo en el área, se inicia el procedimiento de conexión. En este momento, el primer dispositivo (aquel que encuentra a la otra unidad) asume el rol de maestro de lo que pronto será una piconet. Se utilizan dos procedimientos para establecer la conexión. El primero es el procedimiento de búsqueda o detección (inquiry), utilizado cuando la dirección del otro dispositivo es desconocida. Una vez que se conoce la misma, se utiliza un segundo procedimiento llamado paginación (page). Este último sirve para “despertar“ a la otra unidad y establecer la conexión completa entre los dispositivos.

En la tecnología inalámbrica Bluetooth, los equipos se comunican directamente (comunicación ad-hoc) y existen, además, procedimientos de operación que facilitan la creación de piconets para que se establezcan comunicaciones adicionales. Los procedimientos y modos se aplican en capas diferentes de la arquitectura, por lo que un dispositivo podrá participar

simultáneamente en varios de estos procedimientos y modos. El procedimiento se muestra en la siguiente figura:

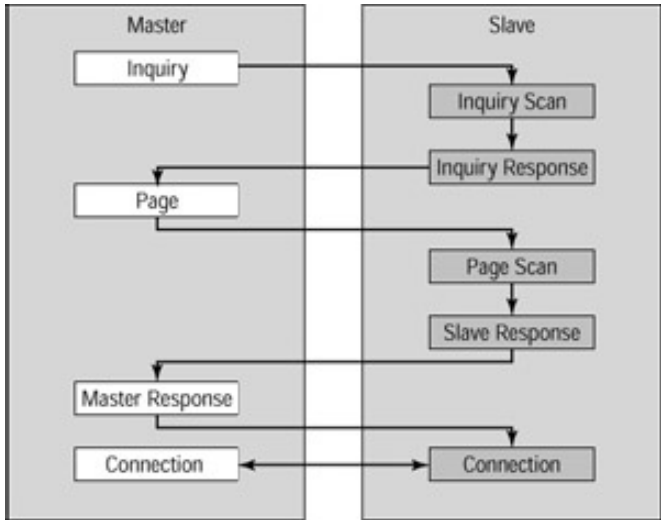

Figura A. 9 - Esquema de establecimiento de conexión

Tiempos de conexión

Tratándose de un enlace de radio, los tiempos de conexión pueden extenderse tanto como lo que les lleve al transmisor y receptor sincronizarse antes de que comience la comunicación. Esta limitación puede tener serias consecuencias si la transmisión fuera de naturaleza crítica.

Existen dos retardos a la hora de establecer un enlace Bluetooth. Primero, toma tiempo descubrir dispositivos en alcance de radio. En particular, un dispositivo manda un paquete de búsqueda y recibe respuestas de dispositivos en el área, que son reportadas al usuario. Puede lleva hasta 10 segundos encontrar todos los dispositivos en alcance de radio. Un segundo retardo ocurre cuando se establece la conexión propiamente dicha (en las aplicaciones orientadas a conexión), y puede tomar hasta otros 10 segundos.

Operation	**Minimum**	**Average**	**Maximum**
	Time (sec)	Time (sec)	Time (sec)
Inquiry	0.00125	3 – 5	10.24 – 30.72
Paging	0.0025	1.28	2.56
Total	0.00375	4.28 – 6.28	12.8 – 33.28

Estados de conexión

La figura A. 10 muestra un diagrama de estado que contiene los distintos estados que se usan en el controlador de enlaces. Hay tres estados principales:

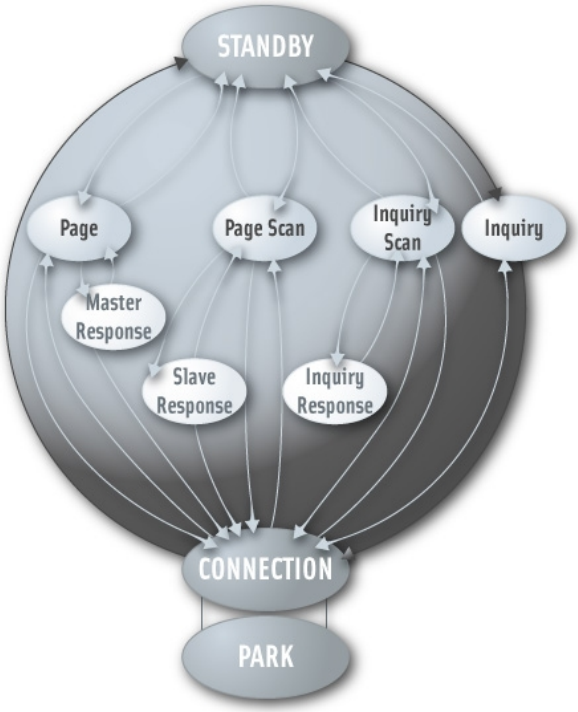

Figura A. 10 - Diagrama de estados del dispositivo Bluetooth

El estado de espera STANDBY es utilizado por los equipos cuando no hay necesidad de enviar información por un largo período de tiempo, por lo que transceptor es apagado para ahorrar energía. Además, este estado se utiliza si un dispositivo quiere ser descubierto por otros.

Un segundo estado de espera PARK es utilizado por un esclavo cuando no desea participar en el canal, pero no quiere perder la sincronización. Luego, a pedido suyo o del maestro, el esclavo puede volver a ser miembro activo de la piconet. El maestro puede forzar a un esclavo al estado park, en caso de ser necesario o porque así lo requiere determinada aplicación.

En el estado CONNECTION, el dispositivo es miembro activo en la piconet, ya sea como esclavo o maestro.

Además, hay siete sub-estados: paginación, detección de paginación, búsqueda, detección de búsqueda, respuesta del maestro, respuesta del esclavo y respuesta a la búsqueda. Los sub-estados son estados transitorios que se usan para establecer conexiones y permitir el descubrimiento de dispositivos. Para pasar de un estado o sub-estado a otro, se usan comandos del gestor de enlaces o bien señales internas del controlador de enlaces.

Comunicaciones lógicas ACL y SCO

Sobre los canales físicos, pueden establecerse dos tipos de enlaces entre el maestro y uno o varios esclavos: síncrono orientado a conexión (SCO) y asíncrono no orientado a conexión (ACL). En el estándar bluetooth del 2002, ACL y SCO son considerados como enlaces físicos, pero a partir del 2005, con la adición del SCO extendido (eSCO) y para futuras expansiones, es

mejor considerarlos como comunicaciones lógicas, describiendo con mayor precisión su propósito, y considerar al enlace físico como el concepto de enlace punto a punto entre el maestro y un esclavo que corre sobre el canal físico, y que está asociado biunívocamente a una comunicación lógica.
En la comunicación lógica SCO, el maestro mantiene la comunicación usando slots de tiempo reservados a intervalos regulares. El maestro puede soportar hasta 3 comunicaciones lógicas SCO al mismo esclavo, o a diferentes esclavos dentro de la piconet.
La comunicación lógica SCO es típicamente utilizado para mandar voz ya que es capaz de imponer un cierto retardo máximo en el envío de paquetes, funcionalidad fundamental a la hora de transmitir voz en tiempo real.
En los slots no reservados por comunicaciones lógicas SCO, el maestro puede intercambiar información con cualquier esclavo slot por slot, estableciendo comunicaciones ACL. Esta comunicación lógica provee una conexión del tipo packet-switched entre el maestro y todos los esclavos activos de la piconet, a diferencia de la comunicación lógica SCO cuya conexión es del tipo circuit-switched. ACL soporta servicios síncronos y asíncronos, pero entre el maestro y un esclavo, sólo puede estar activo una comunicación ACL.

A.3.7 Conclusiones

La tecnología bluetooth está especialmente diseñada para equipos portables lo que implica dispositivos pequeños, de bajo consumo y precio, aspectos que pueden resultar críticos según la aplicación.

La capacidad de los dispositivos Bluetooth de establecer redes de tipo ad-hoc es imprescindible a la hora de pensar en redes entre dispositivos móviles, donde tal vez, no se disponga de infraestructura fija como puede ser un access point, para gestionar la red. Resulta interesante la manera en que bluetooth resuelve este problema asignando un maestro que organiza la red, oficiando de alguna manera de access point, y cuyo rol puede ser tomado por cualquier unidad en la piconet. Puede asignarse este rol al dispositivo que tenga más recursos por ejemplo, o al que tenga más información que desee difundir. Pensando en comunicación vehicular, esto puede ser útil por ejemplo, en la aplicación concreta de un vehículo que va a la cabeza de una fila, y que al tener él la información del estado del tráfico primero que nadie, la difunda hacia atrás.

En cuanto al alcance, puede llegar a ser de 100 metros en el mejor de los casos para dispositivos de clase 1, aunque son los de clase 3, con un alcance máximo de 10 metros, los que se han extendido masivamente, dado que el espíritu de la tecnología y para lo cual fue diseñada era en un principio sustituir el cableado entre periféricos, lo que implica enlaces inalámbricos de corto alcance. Tratándose de redes vehiculares, el alcance es bastante importante dado que 100 metros de alcance en el mejor de los casos, según el desplazamiento relativo de los vehículos, puede implicar tiempos transmisión muy bajos. Analizando los tiempos de conexión, los cuales pueden tener un promedio de 5 a 7 segundos aproximadamente, pero pueden llegar a un máximo de 30, está claro que bluetooth no es un buen candidato para establecer redes en que la topología cambie rápidamente, como es el caso de los vehículos en movimiento. Esto puede afectar el alcance de la aplicación final imponiendo grandes limitaciones. Por otro lado, debe tenerse presente la capacidad de las redes bluetooth de conformar scatternets como una forma de extender el alcance de la red si se dispone de una alta densidad de nodos, como es el caso por ejemplo de zonas céntricas.

ANEXO B. Documentación del código Vozila

B.1. Referencia del Archivo vozila.c

Programa principal para comunicación en modo Monitor, contiene a la funcion main.

Inicializa las diferentes variables, recursos y procesos del sistema.

Lanza 6 procesos concurrentes que se encargan de las siguientes tareas:
· Proceso principal: inicializa y libera los recursos, lanza los demás procesos y recibe comandos de línea para el envío de paquetes y despliegue de la tabla de vecinos.
· Capturar y procesar las tramas inalámbricas.
· Interfaz con el GPS.
· Enviar las tramas de beacon en intervalos regulares.
· Depurar la tabla de vecinos.
· Interfaz gráfica.

Mantiene 2 segmentos de memoria compartida coordinando su acceso mediante semáforos:
· Tabla de vecinos.
· Última actualización del GPS.

Definiciones

#define	**OFFSET_FLAGS 0x08** **Offset del campo de banderas en el encabezado Radiotap.**
#define	**OFFSET_RATE 0x09** **Offset del campo rate en el encabezado Radiotap.**
#define	**OFFSET_CHANNEL 0x0a** **Offset del campo del canal de radio en el encabezado Radiotap.**
#define	**OFFSET_FRAMECONTROL 0x00** **Offset del campo de control en el encabezado 802.11.**
#define	**OFFSET_DURATION 0x02** **Offset del campo de duración en el encabezado 802.11.**
#define	**OFFSET_DESTINATION 0x04** **Offset del campo de dirección de destino en el encabezado 802.11.**
#define	**OFFSET_SOURCE 0x0A** **Offset del campo de dirección de origen en el encabezado 802.11.**
#define	**OFFSET_VOZILA_SUBTYPE 0x10**

	Offset del campo del subtipo de datos Vozila en el encabezado 802.11.
#define	OFFSET_VOZILA_BSSID 0x11 Offset del campo de BSSID Vozila en el encabezado 802.11.
#define	OFFSET_SEQUENCE 0x16 Offset del campo del número de secuencia en el encabezado 802.11.
#define	VOZILA_BEACON 0x01 Bandera que identifica al paquete como Beacon.
#define	VOZILA_DATA 0x02 Bandera que identifica al paquete como Datos.
#define	VOZILA_REPEATABLE 0x04 Bandera que identifica al paquete para ser repetido.
#define	RTAP_HEADER_LENGTH 15 Largo del encabezado Radiotap.
#define	n80211_HEADER_LENGTH 24 Largo del encabezado 802.11.
#define	BEACON_INTERVAL 300000 Intervalo de envío de Beacons, en microsegundos.
#define	GPS_SIM 1 Controla la simulación del GPS.
#define	le16_to_cpu(x) (x) Macro para convertir campos de 2 bytes a little endian.
#define	le32_to_cpu(x) (x) Macro para convertir campos de 4 bytes a little endian.
#define	unlikely(x) (x)

Documentación de las funciones

```
int main ( int    argc,
           char * argv[]
         )
```

Programa principal.

Inicializa variables y pone en marcha los diferentes procesos del sistema.

Comandos de línea:
· *list*: despliega la tabla de vecinos.
· *broadcast < largo >*: envía por broadcast un paquete con carga 0x01 y largo "largo".
· *repbroad < largo >*: envía por broadcast un paquete para ser repetido con carga 0x01 y largo "largo".
· *testforward < nombre >*: envía por broadcast un paquete para ser repetido identificado con "nombre".
· *stats*: Imprime en pantalla las estadísticas de packet loss calculado a partir de los beacon de los vecinos.
· *savestats*: Corre una rutina de cálculo de packet loss a partir de los beacon de los vecinos y guarda los resultados disco
· *beacon stop*: suspende el envío de paquetes de beacon.
· *beacon cont*: reanuda el envío de paquetes de beacon.
· *exit*: cierra el programa.

Devuelve:
0 si se ejecutó correctamente
1 si ocurrió algún error.

Documentación de las variables

__u8 ieeeHeader[]

Valor inicial:

```
 {
        0x08, 0x00,
        0x00, 0x00,
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0x66, 0x22, 0x33, 0x44, 0x55, 0x66,
        0x00,
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0x00, 0x00,
}
```

Encabezado 802.11. Contiene las direcciones de origen, destino y red, así como un número de secuencia.

Esta variable se usa como plantilla para los paquetes salientes, donde en tiempo de ejecución se modifica la dirección de destino, subtipo de datos y número de secuencia según corresponda. A su vez, ni bien se inicializa la interfaz inalámbrica, se copia a esta plantilla los últimos 5 bytes de la dirección MAC de la placa.

char myAlias[] = "R_OBU"

Alias genérico de este nodo. Útil en funciones de repetidor.

Han sido implementados hasta el momento dos opciones:
R_OBU = Repeating On-Board Unit
NR_OBU = Non Repeating On-Board Unit

Nota:

Este parametro queda abierto a futuras opciones como R_RSU y NR_RSU por ejemplo.

struct sembuf p = {0,-1,0}

Estructura para liberar un recurso compartido mediante semáforos del sistema.

__u8 radioTapHeader[]

Valor inicial:

```
 {

        0x00, 0x00,
        0x0f, 0x00,
        0x0e, 0x08, 0x00, 0x00,
        0x08,
        0x16,
        0x85, 0x09, 0x80, 0x04,
        0x00,
}
```

Encabezado Radiotap para adjuntar a los paquetes salientes. Contiene instrucciones para el driver de la placa de red inalámbrica.

struct sembuf v = {0,1,0}

Estructura para solicitar el acceso a un recurso compartido mediante semáforos del sistema.

B.2. Referencia del Archivo sender.c

Biblioteca con funciones para inyectar paquetes en la interfaz inalámbrica.

Contiene funciones que permiten construir e inyectar los diferentes tipos de paquetes como son los paquetes beacon o los de datos unicast y broadcast, así como también retransmitir paquetes entrantes.

Documentación de las funciones

```
void addToList ( int  id  )
```

Agrega el identificador de paquete VOZILA_REPEATABLE a la lista de identificadores.

Parámetros:

id **Identificador a agregar.**

```
int idInList ( int  id  )
```

Busca un identificador de paquete VOZILA_REPEATABLE a la lista de identificadores.

Parámetros:

id **Identificador a agregar.**

Devuelve:

1 Si el identificador ya está en la lista.
0 Si el identificador no está en la lista.

```
void initBeacon ( void    )
```

Inicializa las variables y construye los encabezados necesarios para armar los paquetes de beacon.

```
int repeatPacket ( pcap_t *          pcapDev,
                   __u8 *            n80211,
                   int               payloadLength
                                     ,
                   struct timeval    ts
                 )
```

Función que inyecta un paquete que indica especificamente que debe ser repetido.

Chequea si el paquete debe ser repetido por el host que lo recibió y en caso afirmativo lo inyecta. Previo a la inyección, cambia la dirección de origen por la propia y agrega la dirección de origen anterior a la lista de repetidores por las que pasó el paquete. Razones para no repetir: Estoy en la lista de quienes ya lo repitieron o es una copia que llegó por otro camino.

Parámetros:

pcapDev	Puntero al descriptor del dispositivo a inyectar el paquete.
n80211	Puntero al encabezado 80211.
payloadLength	Longitud del payload.
ts	Timestamp del paquete.

Devuelve:

0 si inyectó el paquete.
-1 si no pudo inyectar el paquete.
1 si no se repitió el paquete porque no correspondía.

```
int sendBeacon ( pcap_t *  pcapDev  )
```

Inyecta paquetes beacon en la interfaz inalámbrica.

Construye el paquete beacon a enviar. Se encarga de actualizar el número de secuencia del encabezado ieee (correspondiente a los beacon) en cada invocación. Hace uso de las funciones de gps.c para obtener los datos de posición, velocidad y rumbo para agregar al paquete. Finalmente inyecta el paquete en la interfaz.

Parámetros:

pcapDev	Puntero al descriptor del dispositivo a inyectar el paquete.

Devuelve:

0 si inyectó el paquete correctamente.
-1 si no pudo inyectar el paquete.

```
int sendBroadcastData ( pcap_t *  pcapDev,
                        __u8 *    payload,
                        int       payloadLength
                      )
```

Inyecta paquetes de datos broadcast en la interfaz inalámbrica.

Construye el paquete broadcast a enviar actualizando el número de secuencia del encabezado ieee correspondiente a los datos de broadcast en cada invocación e inyecta el paquete.

Parámetros:

pcapDev	Puntero al descriptor del dispositivo a inyectar el paquete.
payload	Puntero al payload del paquete a inyectar.
payloadLength	Longitud del payload.

Devuelve:

0 si inyectó el paquete.
-1 si no pudo inyectar el paquete.

```
int sendRepeatableBroadcastData ( pcap_t *  pcapDev,
                                  __u8 *    payload,
                                  int       payloadLength
                                  ,
```

```
                                    char *      alias,
                                    int         hops
                                )
```

Inyecta paquetes de datos broadcast como VOZILA_REPEATABLE en la interfaz inalámbrica.

Construye el paquete broadcast a enviar agregando el destinatario especificado, conststruye el encabezado para repetir y lo inyecta.

Parámetros:

pcapDev	**Puntero al descriptor del dispositivo a inyectar el paquete.**
payload	**Puntero al payload del paquete a inyectar.**
payloadLength	**Longitud del payload.**
alias	**Alias que debe repetir el paquete.**
hops	**Cantidad de saltos.**

Devuelve:

0 si inyectó el paquete.
-1 si no pudo inyectar el paquete.

```
int sendRepeatableUnicastData ( pcap_t *  pcapDev,
                                __u8 *    payload,
                                int       payloadLength
                                ,
                                __u8 *    dest,
                                char *    alias,
                                int       hops
                              )
```

Inyecta paquetes de datos unicast como VOZILA_REPEATABLE en la interfaz inalámbrica.

Construye el paquete unicast a enviar agregando el destinatario especificado, conststruye el encabezado para repetir y lo inyecta.

Parámetros:

pcapDev	**Puntero al descriptor del dispositivo a inyectar el paquete.**
payload	**Puntero al payload del paquete a inyectar.**
payloadLength	**Longitud del payload.**
dest	**Puntero a la dirección MAC destino.**
alias	**Alias que debe repetir el paquete.**
hops	**Cantidad de saltos.**

Devuelve:

0 si inyectó el paquete.
-1 si no pudo inyectar el paquete.

```
int sendUnicastData ( pcap_t *  pcapDev,
                      __u8 *    payload,
```

```
    int         payloadLength
                ,
    __u8 *      dest
)
```

Inyecta paquetes de datos unicast en la interfaz inalámbrica.

Construye el paquete unicast a enviar agregando el destinatario especificado y lo inyecta.

Parámetros:

pcapDev	Puntero al descriptor del dispositivo a inyectar el paquete.
payload	Puntero al payload del paquete a inyectar.
payloadLength	Longitud del payload.
dest	Puntero a la dirección MAC destino.

Devuelve:

0 si inyectó el paquete.
-1 si no pudo inyectar el paquete.

Documentación de las variables

char alias

Arreglo de caracteres para guardar un alias.

char beacondata

Puntero a datos del paquete beacon.

size_t beaconframeLength

Variable que contiene la longitud de un paquete beacon.

__u8 beaconframeToSend

Puntero al encabezado del paquete a enviar.

__u8 beaconheaderptr

Puntero al encabezado del paquete beacon.

size_t beaconinjectLength

Variable que contiene la longitud final del beacon a inyectar.

__u16 beaconsecNumber

Contiene el número de secuencia de un paquete beacon a enviar.

__u16 broadsecNumber = 0

Contiene el número de secuencia de un paquete broadast a enviar.

char hostName

Arreglo de caracteres que contiene el nombre correspondiente al host.

int idIndex = 0

Índice para agregar identificadores de paquetes VOZILA_REPEATABLE.

int idList

Lista con los últimos MAX_ID números de identificación de paquetes VOZILA_REPEATABLE.

__u8 ieeeHeader[]

Encabezado 802.11. Contiene las direcciones de origen, destino y red, así como un número de secuencia.

Esta variable se usa como plantilla para los paquetes salientes, donde en tiempo de ejecución se modifica la dirección de destino, subtipo de datos y número de secuencia según corresponda. A su vez, ni bien se inicializa la interfaz inalámbrica, se copia a esta plantilla los últimos 5 bytes de la dirección MAC de la placa.

char myAlias[]

Alias genérico de este nodo. Útil en funciones de repetidor.

Han sido implementados hasta el momento dos opciones:
R_OBU = Repeating On-Board Unit
NR_OBU = Non Repeating On-Board Unit

Nota:
Este parametro queda abierto a futuras opciones como R_RSU y NR_RSU por ejemplo.

__u8 radioTapHeader[]

Encabezado Radiotap para adjuntar a los paquetes salientes. Contiene instrucciones para el driver de la placa de red inalámbrica.

B.3. Referencia del Archivo receiver.c

Biblioteca de funciones para capturar y procesar los paquetes de la interfaz en modo monitor.

Contiene funciones para inicializar filtros de captura, capturar tramas y realizar el procesamiento posterior.
Se realiza el procesamiento según la información contenida en el campo VOZILA_SUBTYPE del encabezado ieee 802.11

Documentación de las funciones

```
void dump ( __u8 *  pu8,
            int     nLength
          )
```

Rutina para volcar datos de memoria, para depuración.

Parámetros:

pu8 Puntero a los datos.
nLength Largo de los datos.

```
int initReceiver ( pcap_t *  pcapDev  )
```

Inicializa "select" y el filtro de recepción en la interfaz inalámbrica.

Sólo pasarán los paquetes con dirección mac de la familia 66:xx:xx:xx:xx:xx Configura los file descriptors necesarios para utilizar "select" en la lectura de paquetes en la placa. Esto es necesario para bajar el uso de CPU, ya que pcap por defecto utiliza el 100% del CPU mientras espera por paquetes en la interfaz.

Parámetros:

pcapDev Puntero a la interfaz inalámbrica.

Devuelve:
0 si se configuró con éxito.
-1 si ocurrió un error.

```
int receiveBroadcast ( __u8 *          n80211,
                       int             payloadLength
                       ,
                       struct timeval  ts
                     )
```

Procesa los datos de broadcast recibidos.

Aquí se realizará la comunicación con las capas superiores, lo cual debe ser implementado.
A modo de demostración del prototipo, tan sólo se imprimen en pantalla los datos recibidos.

Parámetros:

n80211	Puntero al encabezado 802.11 del paquete recibido.
payloadLength	Largo de los datos (carga útil).
ts	Timestamp del paquete recibido, obtenido del encabezado pcap.

Devuelve:

0 si se procesaron los datos con éxito.
-1 si ocurrió un error.

```
int receiveData ( pcap_t *  pcapDev  )
```

Captura tramas en la interfaz inalámbrica.

Permanece suspendido mediante "select" hasta la llegada de un paquete a la interfaz, cuando convoca a los métodos de captura de pcap.
Detecta si la trama capturada fue en realidad generada en la propia interfaz (un paquete que se está inyectando).
Procesa el encabezado radioTap para conocer si el paquete contiene un FCS.
Procesa el encabezado 802.11 leyendo el subtipo de datos Vozila e invocando las funciones de procesamiento apropiadas en cada caso.

Parámetros:

pcapDev Puntero a la interfaz inalámbrica.

Devuelve:

0 si recibió en forma correcta.
1 si el paquete capturado es de la tarjeta local.
-1 si ocurrió un error durante la recepción.

```
int receiveUnicast ( __u8 *          n80211,
                     int             payloadLength
                     ,
                     struct timeval  ts
                   )
```

Procesa los datos unicast recibidos.

Aquí se realizará la comunicación con las capas superiores, lo cual debe ser implementado.
A modo de demostración del prototipo, tan sólo se imprimen en pantalla los datos recibidos.

Parámetros:

n80211	Puntero al encabezado 802.11 del paquete recibido.
payloadLength	Largo de los datos (carga útil).
ts	Timestamp del paquete recibido, obtenido del encabezado pcap.

Devuelve:

0 si se procesaron los datos con éxito.
-1 si ocurrió un error.

Documentación de las variables

int fd

File descriptos para usar con el comando select.

__u8 ieeeHeader

Encabezado 802.11. Contiene las direcciones de origen, destino y red, así como un número de secuencia.

Esta variable se usa como plantilla para los paquetes salientes, donde en tiempo de ejecución se modifica la dirección de destino, subtipo de datos y número de secuencia según corresponda. A su vez, ni bien se inicializa la interfaz inalámbrica, se copia a esta plantilla los últimos 5 bytes de la dirección MAC de la placa.

char myAlias

Alias genérico de este nodo. Útil en funciones de repetidor.

Han sido implementados hasta el momento dos opciones:
R_OBU = Repeating On-Board Unit
NR_OBU = Non Repeating On-Board Unit

Nota:
Este parametro queda abierto a futuras opciones como R_RSU y NR_RSU por ejemplo.

__u8 radioTapHeader

Encabezado Radiotap para adjuntar a los paquetes salientes. Contiene instrucciones para el driver de la placa de red inalámbrica.

fd_set read_fds

Set de file descriptors para usar con el comando select.

B.4. Referencia del Archivo neighbours.c

Biblioteca de funciones para el manejo de la tabla de vecinos.

Contiene funciones que permiten agregar nuevos vecinos, actualizar la información de los vecinos ya existentes en la tabla y borrar vecinos inactivos. Para ello mantiene actualizado el parámetros TTL de cada nodo, decrementándolo si no se han recibido paquetes desde él. Se deben utilizar estas funciones para manejar la tabla de vecinos ya que estas usan los semáforos necesarios para acceder a la memoria compartida.

Estructuras de datos

struct **neighbour**

Estrucutra que contiene los datos de un nodo vecino.

Campos de datos

__u8	**address** [6] Dirección mac.
char	**name** [MAX_NAME] Nombre.
char	**alias** [MAX_ALIAS] Alias.
struct timeval	**timestamp** Timestamp de la última recepción.
unsigned int	**beaconCount** Cantidad de paquetes de beacon recibidos.
unsigned int	**beaconLoss** Cantidad de paquetes de beacon perdidos.
unsigned int	**seqBeacon** Número de sequencia de la última recepción de Beacon.
unsigned int	**seqUniTo** Número de sequencia del último paquete unicast enviado al vecino.
unsigned int	**seqUniFrom** Número de sequencia del último paquete unicast

	recibido desde el vecino.
unsigned int	seqBroadFrom Número de sequencia de la última recepción de broadcast del vecino.
unsigned int	ttl Time To Live.
double	lat Latitud.
double	lon Longitud.
double	hdg Heading.
double	spd Speed.
__u8	flags Flags.

struct	neighbourTable Tabla que contiene a los vecinos.

Campos de datos

neighbour	list [MAX_NEIGHBOURS] Lista de vecinos.
int	index Primer lugar libre en la tabla.

Definiciones

#define	MAX_NEIGHBOURS 50 Máximo numero de vecinos en la tabla.
#define	MAX_NAME 20 Largo máximo del nombre de un host.
#define	MAX_ALIAS 15 Largo máximo del alias de un host.

#define	MAX_ID 50 Largo máximo de la lista de identificadores para mensajes VOZILA_REPEATABLE.
#define	TTL 10 Tiempo de vida de un vecino en la tabla en eventos.
#define	TTL_DECREMENT_INTERVAL 800000 Intervalo de tiempo entre decrementos de TTL, en micro segundos.
#define	LOS_FLAG 0x01 0x02 si el nodo está en alcance de radio.
#define	POS_FLAG 0x02 0x02 si contamos con datos de la posición.
#define	HDG_FLAG 0x04 0x04 si contamos con datos de rumbo.
#define	SPD_FLAG 0x08 0x08 si contamos con datos de la velocidad.
#define	RSU_FLAG 0x10 0x10 si es un Road Side Unit (nodo fijo).
#define	REP_FLAG 0x20 0x20 sisi el nodo es un repetidor.

Documentación de las funciones

void closeNBTable (void)

Función que cierra la tabla de vecinos liberando los recursos IPCS.

void decrementTTL (void)

Decrementa el TTL de cada vecino en la tabla.

Se debe llamar periódicamente para mantener la tabla libre de vecinos que ya no están a la vista.

neighbourTable getLOSneighbours (void)

crear una subtabla de vecinos sólamente con los vecinos que están en alcance directo.

Devuelve:

La nueva tabla de vecinos.

neighbourTable getNBTable (void)

Devuelve puntero a la tabla de vecinos.

Devuelve:
Puntero a la tabla de vecinos.

int initNBTable (void)

Inicializa la tabla de vecinos y los recursos IPCS.

Crea un segmento de memoria compartida para que todos los procesos puedan acceder a ella. Luego crea un semáforo para controlar el acceso a la tabla de vecinos. Por último inicializa adecuadamente la tabla de vecinos y devuelve el ID del semáforo asociado a la misma.

Devuelve:
El ID del semáforo asociado a la tabla de vecinos.
-1 en caso de error.

void printNBTable (void)

Imprime la tabla de vecinos en la salida estándar.

void printStats (void)

Imprime las estadísticas.

static int rmNeighbour (int neigIndex) [static]

Elimina un host de la lista de vecinos.

Esta función NO maneja los semáforos y debe tenerse en cuanta al invocarla.

Parámetros:

neigIndex **Indice del vecino en la tabla.**

Devuelve:
-1 si el indice es incorrecto
0 en caso contrario.

void saveStats (void)

Rutina para calcular el packet loss y guardar los resultados en disco.

Para cada vecino, calcula el packet loss durante 20 segundos, repitiendo el proceso 10 veces.
El tiempo total de ejecución de la rutina será por tanto 200 segundos y los resultados quedan disponibles en ./stats.csv

Nota:
El packet loss es calculado a partir de las tramas de beacon recibidas desde los vecinos. Se interpretan las discontinuidades en el número de secuencia como paquetes perdidos.

```
static int searchNeighbour ( __u8 * addrToFind ) [static]
```

Busca un miembro en la tabla de vecinos por su dirección MAC.

Parámetros:

addrToFind Puntero a la direccion MAC del vecino a buscar.

Devuelve:
Indice en la tabla del vecino a buscar si lo encontró.
-1 si no esta en la tabla.

```
int updateNeighbours ( __u8 *           n80211Header
                       ,
                       int              payloadLength
                       ,
                       struct timeval   timestamp
                     )
```

Actualiza la tabla de vecinos al recibir un Beacon.

Recibe el puntero al encabezado 802.11 del Beacon, el largo del payload y el timestamp del paquete. Actualiza los datos si el host ya pertenece a la tabla o crea una nueva entrada si no está. Devuelve -1 si no queda espacio en la tabla.

Parámetros:

n80211Header Puntero al encabezado 802.11 del Beacon.
payloadLength Largo del payload.
timestamp Timestamp del paquete.

Devuelve:
-1 si no queda espacio en la tabla.
0 si el vecino fue agregado a la tabla.
1 si ya existía en la tabla.

```
int updateNeighboursWithTable ( neighbourTable * rcvTable )
```

Actualiza la tabla de vecinos a partir de la tabla de vecinos de un tercero.

Se fija nodo a nodo si el vecino ya está en la tabla, y sino lo agrega.

Parámetros:

rcvTable Puntero a la tabla recibida.

Devuelve:
0 si terminó con éxito.
-1 si se llenó la tabla.

Documentación de las variables

nbSemId [static]

Identificador del semáforo que evita que escrituras simultáneas en memoria compartida.

nbSemKey [static]

Clave pedida al S.O. para generar el Id del semáforo.

nbTable [static]

Puntero a la tabla de vecinos.

Apuntará a un segmento de memoria compartida para que puedan acceder todos los procesos y funciones fuera de este archivo fuente.

nbTableId [static]

Identificador de memoria compartida para la tabla de vecinos.

nbTableKey [static]

Clave pedida al S.O. para generar el Id de memoria compartida.

struct sembuf p

Estructura para liberar un recurso compartido mediante semáforos del sistema.

struct sembuf v

Estructura para solicitar el acceso a un recurso compartido mediante semáforos del sistema.

B.5. Referencia del Archivo gps.c

Descripción detallada

Biblioteca de funciones para interfaz con el GPS del vehículo.

Utiliza las funciones y estructuras de la biblioteca NMEA creada por Tim, http://nmea.sourceforge.net

---

Definiciones

#define	**RMC_LENGTH 75** Largo de la sentencia NMEA GPRMC.
#define	**GPS_REFRESH 10** Frecuencia de refresco del GPS, en Hz.

Funciones

int	**getGPRMCstring (char *)** Obtiene los datos actuales del GPS del vehículo en formato NMEA.
int	**initGPS (void)** Inicializa la memoria de los datos de GPS.
void	**closeGPS (void)** Libera los recursos compartidos IPCS.
void	**simulateLine (nmeaPOS, double, double, int)** Simula una trayectoria en línea recta.
void	**simulateCircle (nmeaPOS, double, double, double, int)** Simula una trayectoria circular.
nmeaINFO	**getnmeaInfo (void)** Devuelve la información actual de GPS.

Documentación de las funciones

void closeGPS (void)

Libera los recursos compartidos IPCS.

Debe ser invocada por el proceso principal antes de cerrar.

int getGPRMCstring (char * nmeaString)

Obtiene los datos actuales del GPS del vehículo en formato NMEA.

Genera la sentencia NMEA GPRMC donde encontramos datos de posición, dirección y velocidad.

Parámetros:

nmeaString **Puntero a la sentencia generada.**

Devuelve:

0 si hay datos de GPS.
-1 si no hay datod de GPS.

```
nmeaINFO getnmeaInfo ( void    )
```

Devuelve la información actual de GPS.

Devuelve:

Información del GPS en formato nmeaINFO *

```
int initGPS ( void    )
```

Inicializa la memoria de los datos de GPS.

Crea el segmento de memoria compartida y los semáforos de acceso.

Devuelve:

El identificador de semáforo.

```
void simulateCircle ( nmeaPOS  start,
                      double   spd,
                      double   hdg,
                      double   radius
                               ,
                      int      side
                    )
```

Simula una trayectoria circular.

Actualiza los datos de GPS en función del punto de partida, velocidad, dirección de partida y radio del círculo en metros.

Parámetros:

start	**Posición de partida, en NMEA Degree.**
spd	**Velocidad en km/h.**
hdg	**Rumbo.**
radius	**Radio de giro en m.**
side	**Sentido de giro: side = 1 círculo a la derecha, side !=1 círculo a la izquierda.**

```
void simulateLine ( nmeaPOS  start,
                    double   spd,
```

```
                double      hdg,
                int         count
            )
```

Simula una trayectoria en línea recta.

**Actualiza los datos de GPS en función del punto de partida, velocidad y dirección.
Refresca la memoria de la posición actual respetando la frecuencia de refresco del GPS.**

Parámetros:

start	**Posición de partida, en NMEA Degree.**
spd	**Velocidad en km/h.**
hdg	**Rumbo.**
count	**Cantidad de actualizaciones hasta detenerse. Si se llama con -1, entonces navegará infinitamente.**

Documentación de las variables

currentGPS

Puntero la última actualización del GPS, posición en NMEA degree.

gpsId

Identificador del segmento de memoria compartida.

gpsKey

Clave del sistema para solicitar un segmento de memoria compartida.

gpsSemId

Identificador del semáforo.

gpsSemKey

Clave del sistema para solicitar un semáforo.

struct sembuf p

Estructura para liberar un recurso compartido mediante semáforos del sistema.

struct sembuf v

Estructura para solicitar el acceso a un recurso compartido mediante semáforos del sistema.

B.6. Referencia del Archivo gui.c

Biblioteca de funciones para el manejo de la interfaz gráfica.

Implementa la interfaz gráfica del sismtema Vo!zila compuesta basicamentes por los siguientes sectores:
· Datos del host.
· Lista de vecinos con datos de gps.
· Representación de los vencinos en un mapa entorno al host.

Definiciones

#define	**MAX_RANGE 750** **Rango maximo del mapa (distancia del centro au un borde perpendicular.**
#define	**RECT_WIDTH 400** **Ancho total del mapa en la dirección horizontal.**
#define	**RECT_HEIGHT 400** **Ancho total del mapa en la dirección vertical.**
#define	**SECT_1_X 30** **Coordenada x del extremo superior izquierdo del sector 1.**
#define	**SECT_1_Y 30** **Coordenada y del extremo superior izquierdo del sector 1.**
#define	**SECT_2_X 30** **Coordenada x del extremo superior izquierdo del sector 2.**
#define	**SECT_2_Y 150** **Coordenada y del extremo superior izquierdo del sector 2.**
#define	**SECT_3_X 650** **Coordenada x del extremo superior izquierdo del sector 3.**
#define	**SECT_3_Y 30** **Coordenada y del extremo superior izquierdo del sector 3.**
#define	**RECT_X 650**

	Coordenada x del extremo superior izquierdo del mapa.
#define	RECT_Y 55 Coordenada y del extremo superior izquierdo del mapa.
#define	WINDOW_WIDTH 1100 Ancho de la ventana principal.
#define	WINDOW_HEIGHT 600 Altura de la ventana principal.
#define	RADIO_PUNTO 5 Radio del punto que representa un nodo en el mapa.

Documentación de las funciones

```
GdkPixbuf* create_pixbuf ( const gchar * filename )
```

Crea un buffer especial para almacenar la imágen del ícono de ventana.

Parámetros:

filename Nombre del archivo de imagen.

Devuelve:

Un objeto de tipo GdkPixbuf.

```
void dibujarPunto ( cairo_t * cr,
                    double   coordX,
                    double   coordY,
                    int      red,
                    int      green,
                    int      blue
                  )
```

Dibuja un punto en el mapa que representa a un nodo.

Parámetros:

cr Puntero a un objeto de tipo cairo_t
coordX Coordenada x del punto a dibujar
coordY Coordenada y del punto a dibujar
red Componente rojo del color del punto
green Componente verde del color del punto
blue Componente azul del color del punto

```
void dibujarRumbo ( cairo_t *  cr,
                    double     coordX_Origen,
                    double     coordY_Origen,
                    double     rumbo,
                    double     vel
                  )
```

Dibuja una linea que representa el rumbo del vehiculo con un largo proporcional a su velocidad.

Parámetros:

cr	Puntero a un objeto de tipo cairo_t
coordX_Origen	Coordenada x del origen de la linea
coordY_Origen	Coordenada y del origen de la linea
vel	Velocidad del vehiculo

```
void escribir ( cairo_t *  cr,
                double     coordX,
                double     coordY,
                double     tSize,
                char *     texto
              )
```

Escribe texto en una posición y con un tamaño de letra determinados.

Parámetros:

cr	Puntero a un objeto de tipo cairo_t
coordX	Coordenada x del inicio del texto
coordY	Coordenada y del inicio del texto
tSize	Tamaño del texto
texto	Puntero al texto a escribir

```
void neighbourGUI ( void    )
```

Función principal para el manejo de la interfaz gráfica.

Inicializa las variables principales, crea y da formato a las ventana, crea los handlers a los eventos, provoca el disparo de la actualización de la ventana a intervalos de tiempo regulares, muestra las ventanas e inicia el loop principal que espera los eventos.

Nota:

LogoGui.jpg debe estar en la carpeta del proyecto, siendo este el ícono de la ventana.

```
static gboolean on_expose_event ( GtkWidget *      widget
                                  GdkEventExpose * event,
                                  gpointer         data
                                )                          [static]
```

Actualiza la ventana.

Captura el evento que se dispara cuando es necesario actualizar la ventana y procede a actualizar la misma.

Parámetros:

widget **Puntero a la ventana a actualizar.**
event **Puntero al evento.**
data.

```
static gboolean refresh_nbWindow_handler ( GtkWidget * widget ) [static]
```

Dispara el evento de actualizar la ventana.

Chequea que la ventana no haya sido destruida y en ese caso dispara el evento de tipo on_expose_event.

Parámetros:

widget **Puntero a la ventana a actualizar.**

Devuelve:

TRUE si existe la ventana.
FALSE si la ventana fue destruida.

ANEXO C. CD

Contenido del CD

1. Documentación del proyecto (este documento)
2. Paper en formato IEEE
3. Poster
4. Documentación del código del prototipo en HTML
5. Archivos fuente del prototipo Vo!zila

Requerimientos del sistema

1. Lectora de CD
2. Visor de pdf compatible con Acrobat 6.0 o mayor
3. Explorador de Internet (Internet Explorer 6.0 o mayor, Mozila Firefox 2.0 o mayor)
4. Para compilar el prototipo:
    a. Ubuntu 9.10
    b. Compilador gcc
    c. Biblioteca libpcap-dev
    d. Biblioteca gtk-2.0+
    e. Tarjeta 802.11 con capacidad de modo Monitor