PROYECTO FINAL DE CARRERA

UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

INTEGRACIÓN AUTOMÁTICA DE NODOS BLUETOOTH EN MANETS

Autora: **Mª Cruz Olivas Delgado**

Directores: **Miguel Ángel Mateo Pla**

Carlos Tavares Calafate

Valencia, Marzo 2007

A mi tío Emi.

Agradecimientos

En primer lugar quiero agradecer todo el tiempo que me han dedicado los directores de este proyecto, Carlos y Miguel. Su implicación y ganas han sido esenciales para el desarrollo del trabajo, pero sobretodo les agradezco su paciencia, su trato y su confianza en mí a lo largo de un año difícil a nivel personal.

A continuación me gustaría agradecer a mis compañeros en el laboratorio del Grupo de Redes de Computadores, por crear un clima de trabajo tan agradable en el que siempre me he sentido a gusto. Marga, David, Dani, Sergio, Marco, Valentina (y demás italianos que han pasado por aquí), y en especial Jorge, que me ha ayudado muchísimo en todo lo que he necesitado. También quiero mencionar a los profesores del GRC Pietro y Juan Carlos. Espero encontrarme en el futuro con un entorno de trabajo así, pero sé que va a ser difícil, habéis puesto el listón muy alto.

A Jose Luis, gracias por su ayuda en la realización de esta memoria, a la cual ha dedicado un tiempo que seguramente tenía destinado a *procrastinar*.

Mi familia me ha tenido que aguantar durante todo este tiempo, en especial mis padres, que conviven conmigo y nunca han dejado de apoyarme y velar por mí, os doy las gracias por todo. También mi hermana, que en la distancia se preocupa por mí y me transmite su cariño. A todos los demás gracias por vuestro apoyo y cariño, y a los que ya no están, por no haberse marchado del todo.

Gracias a mis amigos, tanto los de la Universidad como los de siempre, que me han acompañado todos estos años y esperan tanto como yo que acabe la carrera. Y en especial a Merche por sus tratamientos fisioterapéuticos (y muchas cosas más, por supuesto) y a Anna por estar ahí prácticamente desde que tengo uso de razón.

A Mario le tengo que agradecer más de lo que cabe en unas pocas líneas. Siempre ha confiado en mí, incluso en momentos en los que ni yo misma lo hacía. Ha habido varios momentos de esos a lo largo del desarrollo de este proyecto y de la carrera en general, gracias por no dejarme caer nunca y estar siempre a mi lado.

Índice de contenidos

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1– Motivación

En los últimos años se está investigando muchísimo en el campo de las redes móviles inalámbricas *ad hoc*, conocidas comúnmente por su acrónimo en inglés: MANET (*Mobile Ad Hoc Network*). El interés de las MANETs no se restringe a entornos científicos, si no también para el gran público. La causa más importante del auge de este tipo de redes es el desarrollo tecnológico en el campo de los dispositivos móviles (PDAs, teléfonos, ordenadores…), así como su generalización y gran volumen de ventas. Una MANET está formada por varios de estos dispositivos móviles, haciendo posible la comunicación entre ellos para llevar a cabo diversas tareas.

En general, las redes móviles *ad hoc* pueden ser usadas en todas aquellas situaciones caracterizadas por la carencia de una infraestructura fija, comunicación entre iguales (*peer-to-peer*) y soporte a movilidad. Algunas de las características más importantes este tipo de redes son la topología dinámica, dado el cambio continuo que sufre la red por la movilidad de sus nodos, el establecimiento de enlaces de ancho de banda limitado y capacidad variable, y capacidad de procesamiento en los nodos limitada. Además, el encaminamiento es distribuido, dado que no se dispone de ningún nodo centra, y adaptativo, dado que ha de adaptarse a los cambios de la red.

Dos de los problemas básicos de este tipo de redes son las limitaciones de energía de los nodos móviles, que dejan de funcionar en cuanto se agota su batería, y el tráfico generado por los algoritmos de encaminamiento. Este tráfico crece exponencialmente con cada nodo que se añade a la red.

Mediante el uso de la tecnología inalámbrica Bluetooth se pueden solucionar estos problemas. Bluetooth corresponde a la norma IEEE 802.15.1 y apareció en 1999 tras la colaboración entre grandes empresas del sector tecnológico móvil, como son Ericsson y Nokia. Inicialmente se concibió para eliminar el uso de cables al interconectar dispositivos (auriculares, impresoras, teclado, ratón) a un dispositivo central (PDA, ordenador, teléfono móvil…). Con el paso del tiempo el uso de Bluetooth se ha ido

extendiendo, de modo que en la actualidad podemos encontrar esta tecnología en prácticamente cualquier tipo de dispositivo del mercado, tanto dispositivos móviles como fijos.

Aunque Bluetooth posee un menor alcance y velocidad de transmisión que el estándar IEEE 802.11, conocido como Wi-Fi y que es la tecnología más extendida en MANETs, el atractivo de Bluetooth reside en su bajo coste y bajo consumo de energía. Entre sus principales características destaca el descubrimiento de dispositivos Bluetooth cercanos y de los servicios ofertados por éstos, lo que contribuye a su gran facilidad de uso y versatilidad. Además, Bluetooth permite formar pequeñas redes de área personal (PAN, *Personal Area Network*) con soporte para TCP. Estas redes PAN pueden permitir en la práctica la conexión de nodos móviles Bluetooth a la MANET sin participación activa en la misma, por lo que no añadirían tráfico de encaminamiento a la red, evitando la sobrecarga.

Por todas estas características, sobretodo por el bajo consumo y la posibilidad de no incrementar el tráfico debido a encaminamiento, es interesante la utilización de Bluetooth en redes inalámbricas *ad hoc*. La motivación principal de este proyecto es contribuir al crecimiento constante de los desarrollos y aplicaciones de las MANETs mediante la integración de la tecnología Bluetooth en dichas redes. Como Bluetooth ya se encuentra plenamente extendida y es conocida por el gran público, se convierte en una herramienta esencial para que, en un futuro próximo, se generalice la utilización de las MANETs en todo tipo de escenarios.

1.2 – Objetivos

En el apartado anterior se ha destacado la necesidad de integrar nodos Bluetooth en MANETs, dado el auge de esta tecnología y sus múltiples ventajas. Por lo tanto, a nivel de tecnología hardware, disponemos ya de todos los componentes. Lo que se hace necesario ahora es desarrollar servicios software que hagan realmente funcionales y realizables las MANETs para cualquier tipo de usuario. Es ahí donde se enclava este Proyecto Fin de Carrera, que pretende contribuir al desarrollo de estas tecnologías.

El principal objetivo del proyecto es el desarrollo de un servicio Bluetooth que permita a nodos con tecnología Bluetooth integrarse en una red *ad hoc* de manera sencilla y transparente para el usuario. Este servicio lo denominaremos **MANET_Gateway**. La aplicación sigue un modelo cliente/servidor. Por tanto, es necesario implementar dos programas diferentes: una programa cliente y otro servidor.

El servidor ofrece el servicio MANET_Gateway y se ejecutará en un nodo de la MANET. La aplicación cliente se utilizará desde un nodo que quiera tener acceso a la MANET. La implementación, tanto para el servidor como para el cliente, estará basada en una interfaz gráfica, que será simple e intuitiva con objeto de facilitar las tareas a realizar por el usuario.

Otro objetivo importante es la integración en el servicio de mecanismos de seguridad, dada la importancia de este tema en todo tipo de comunicaciones inalámbricas, donde se hace más difícil el control de ataques externos. Para ello, hay que estudiar los mecanismos de seguridad que ofrece Bluetooth y, a partir de ahí, desarrollar una

solución robusta y eficaz. Dicha solución ha de integrarse en la interfaz gráfica de forma sencilla, facilitando al usuario la gestión de la seguridad. La consecución de seguridad no se encontraba entre los objetivos principales de este proyecto, pero su posterior desarrollo ha hecho que ocupe un lugar destacado.

Por último, es importante realizar algunas pruebas con objeto de evaluar la eficacia de la solución propuesta y la calidad del enlace que se establece. Mediante el estudio de la gestión del enlace Bluetooth se propone diferenciar el tráfico en dicho enlace mediante técnicas de variación del tipo de paquete.

1.3 – Estructura de este documento

Después de esta breve introducción, en los dos capítulos siguientes se presentan las tecnologías utilizadas para este proyecto. El **capítulo 2** trata sobre las redes móviles *ad hoc* o MANETs, comentando sus características y múltiples aplicaciones. En el **capítulo 3** se analiza el estándar de comunicaciones Bluetooth. Se describe la pila de protocolos Bluetooth y el concepto de perfiles, comentando en profundidad únicamente el perfil PAN, esencial para el desarrollo del proyecto. Por último, se hacen algunas consideraciones sobre interferencias entre Bluetooth y Wi-Fi, y se justifica el uso en redes MANET por sus múltiples ventajas.

En el **capítulo 4** se detalla todo el desarrollo del proyecto, reseñando en primer lugar las características principales de BlueZ, que es la pila de protocolos oficial de Linux para Bluetooth. Posteriormente se explica la arquitectura cliente/servidor de la aplicación, las funciones de ambos, y cómo se comunican entre ellos. Por último se describe la implementación de seguridad, importante novedad en lo que a servicios Bluetooth se refiere.

El **capítulo 5** se consagra a las pruebas experimentales realizadas, donde se estudia el tiempo que emplea un cliente en conectarse a la MANET mediante nuestro servicio y la calidad y gestión del enlace establecido.

Las conclusiones se exponen en el **capítulo 6**, donde se hará un resumen de los principales resultados obtenidos y algunos comentarios finales, así como las posibles líneas futuras de trabajo relacionadas con este proyecto.

Por último, en el **anexo**, se describen todos los requerimientos software que necesita la aplicación para funcionar correctamente especificando, en cada caso, las versiones de los mismos.

Las **referencias bibliográficas** se han situado al final de cada capítulo, con objeto de hacer más cómoda su consulta para el lector.

Capítulo 2

Estado del arte en MANETs

2.1 – Historia y definición

La historia de las redes inalámbricas data de finales de la década de los 70, y su interés no ha hecho más que crecer desde entonces. En el área de las comunicaciones inalámbricas se pueden distinguir dos grandes grupos de redes atendiendo a su organización: las redes inalámbricas con infraestructura y las redes inalámbricas de igual a igual (*ad hoc*).

Un esquema de las redes con infraestructura se muestra en la figura 2-1. En estas redes las estaciones que pueden comunicarse entre ellas forman el BSS (*Basic Service Set*) y se conectan al sistema de distribución (DS, *Distribution System*) mediante un punto de acceso (AP, *Access Point*). El DS suele ser cableado y proporciona acceso a otras redes, pudiendo interconectar también varias BSS con lo que se formaría un ESS (*Extended Service Set*). Dentro de estas redes se sitúan también las redes celulares, ampliamente utilizadas en telefonía móvil, que dividen el área de cobertura en regiones denominadas celdas.

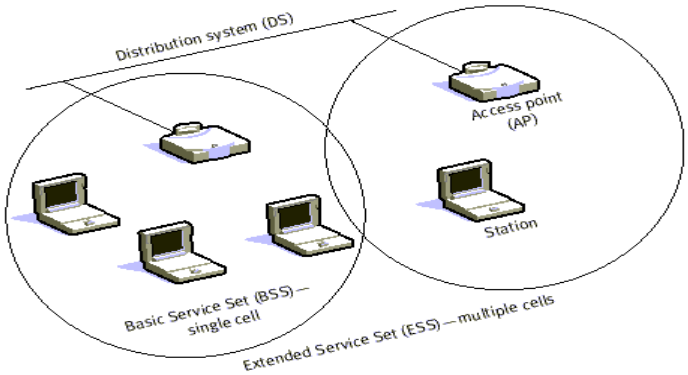


Figura 2-1: Red inalámbrica con infraestructura

En las redes inalámbricas *ad hoc* (figura 2-2) los terminales se comunican entre ellos sin estar conectados a un punto de acceso, constituyendo una IBSS (*Independent BSS*). Este tipo de redes nacieron en la década de 1980, donde fueron muy importantes las investigaciones militares realizadas para crear un sistema de comunicaciones multisalto inalámbrico que pudiera operar en una extensa área geográfica. Hacia el final de la década de los 90 su interés alcanzó la cima principalmente por el rápido crecimiento de Internet.

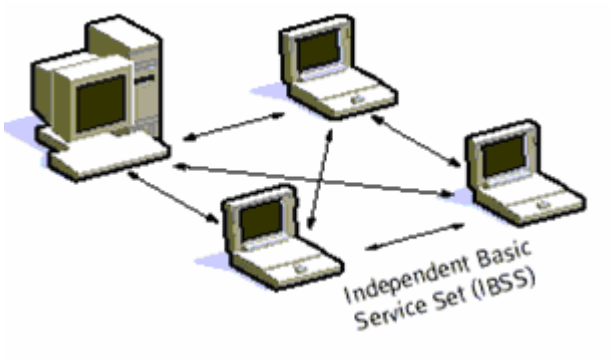


Figura 2-2: Red inalámbrica *ad hoc*

Las redes inalámbricas *ad hoc* se denominan comúnmente como MANET, que es el acrónimo en inglés de *Mobile Ad hoc Network*. Una red móvil *ad hoc* es pues una colección de nodos móviles autónomos que se comunican entre si mediante enlaces inalámbricos, posibilitando servicios de comunicaciones allí donde no existe una infraestructura de red fija ni administración centralizada. En este nuevo entorno descentralizado, los nodos son totalmente autónomos y participan en la toma de decisiones, realizando funciones propias del mantenimiento de la red y tomando parte en los algoritmos de encaminamiento. El término *ad hoc*, a pesar de tener muchas veces una connotación negativa y ser comparado con improvisado o no organizado, es usado en este contexto para expresar un gran nivel de flexibilidad.

En los últimos años se ha popularizado el uso de todo tipo de dispositivos móviles tales como PDAs, teléfonos, ordenadores portátiles, etc. Al desarrollo tecnológico ha ido aparejado un gran volumen de ventas y todo ello favorece el interés creciente en las MANETs, no exclusivamente para entornos científicos o militares como ocurría hasta hace poco, si no para el gran público. Es por ello que actualmente se investiga muchísimo en este campo. Con el objetivo de establecer estándares abiertos en este área emergente, el IETF (*Internet Engineering Task Force*, en castellano Grupo de Trabajo en Ingeniería de Internet) creó el grupo de trabajo MANET [1] para estandarizar protocolos y las especificaciones funcionales de las redes *ad hoc* inalámbricas.

2.2 - Características

En general, cualquier propuesta real aplicable a una MANET deberá tener en cuenta las restricciones impuestas por las características inherentes a este tipo de redes, a saber:

- Topología dinámica
- Enlaces de ancho de banda limitado y capacidad variable
- Limitaciones de energía
- Capacidad de procesamiento en los nodos
- Seguridad física limitada
- Encaminamiento distribuido y adaptativo

En los siguientes apartados se desarrollan más ampliamente estas características.

2.2.1 - Topología

Las redes de topología dinámica están formadas por nodos que se pueden conectar entre varios caminos, esto hace que si un camino está siendo usado se pueda usar otro, seleccionando los caminos de forma dinámica según el estado puntal de la red. Una MANET es una red dinámica, que sufre continuos e impredecibles cambios debido a la movilidad de sus nodos. De hecho, los nodos pueden agregarse de forma espontánea y desaparecer igualmente.

Los nodos en una MANET pueden ser origen o destino de una comunicación o actuar como encaminadores, e incluso pueden ser una pasarela para generar redes híbridas, como podría ser el caso de acceder a Internet a través de un nodo vecino. Asimismo, los nodos integrantes pueden ser de distintos tipos, como veremos a continuación.

2.2.2 – Características debidas a la heterogeneidad de los dispositivos

Hoy en día existen dispositivos móviles muy diferentes capaces de soportar infraestructuras de comunicaciones comunes, como por ejemplo ordenadores portátiles, PDAs o agendas electrónicas, teléfonos móviles que son casi mini computadoras, videoconsolas portátiles, etc. Evidentemente existen grandes diferencias de implementación física entre todos estos dispositivos que pueden afectar al rendimiento de la comunicación y al diseño de protocolos de comunicación.

La figura 2-3 muestra diferentes tipos de dispositivos móviles interactuando en una MANET, cada uno de los cuales con una capacidad de procesamiento, memoria, disco y duración de baterías distintos de los demás. Todas estas características definen la potencia del dispositivo, y en función de eso se caracteriza un dispositivo *ad hoc* en cuanto al papel que juega dentro de una red.

Por lo tanto, dentro de una misma red *ad hoc* suele haber dispositivos más potentes que otros. Algunos de ellos pueden trabajar como cliente/servidor, y otros únicamente como clientes.

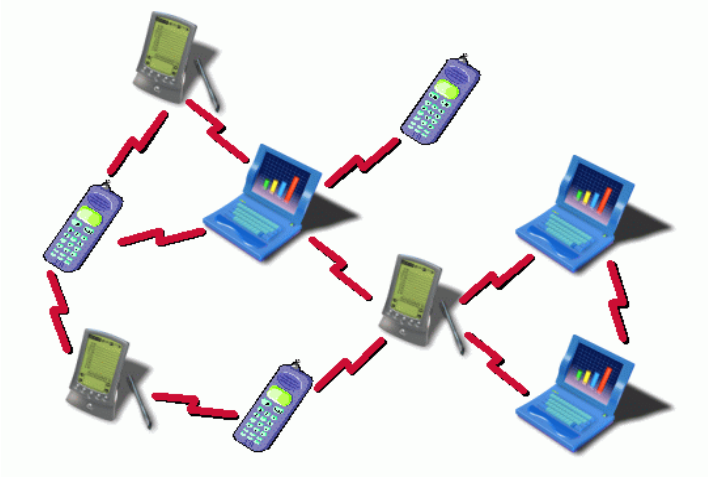


Figura 2-3: MANET formada por dispositivos de distintos tipos

Debido a que no existe ninguna infraestructura previa, los nodos tendrán que realizar funciones de enrutamiento; este es un aspecto muy delicado de cara al consumo de batería, ya que un nodo podría agotar su batería en beneficio de otros, con lo que se hace necesaria la capacidad de "auto examinarse" antes de aceptar una posible petición de reenvío de paquetes por parte de otra fuente de datos.

2.2.3 – Consumo de energía

Como ya se ha señalado, el consumo de energía es un tema muy importante en las redes *ad hoc* por las limitaciones que impone en el software y el hardware que se puede utilizar en las mismas.

Una solución pasa por disponer de mejores baterías a un precio razonable. A este respecto cabe decir que, en la actualidad, se están realizando investigaciones con el objetivo de crear una batería pequeña y barata, con una autonomía muy superior a la de las baterías de Litio (actualmente las más duraderas y versátiles), y que en el futuro pueda por ejemplo alimentar un móvil durante un mes.

Mientras tanto, cualquier desarrollo, estudio o implementación, tanto software como hardware, de soluciones dirigidas a las MANETs ha de ser muy cuidadoso con el consumo de energía que supondrá, intentando minimizar este gasto en la medida de lo posible.

Como veremos, este es un aspecto muy positivo de la tecnología Bluetooth, y una de las causas de su interés como parte importante en el desarrollo de las MANETs.

2.2.4 – Enrutamiento

Uno de los retos de las redes *ad hoc* es cómo encaminar los paquetes de información y, además, hacerlo eficientemente. El encaminamiento o enrutamiento es el mecanismo por el que en una red los paquetes de información se hacen llegar desde su origen a su destino final, siguiendo un camino o ruta a través de la red. En una red grande o en un conjunto de redes interconectadas el camino a seguir hasta llegar al destino final puede suponer transitar por muchos nodos intermedios.

Un protocolo de encaminamiento para redes *ad hoc* [2] debe poseer ciertas características que lo hacen diferente a los protocolos de encaminamiento utilizados en otras redes. Para empezar, debe ser distribuido: cuando todos los nodos son móviles, no tiene sentido tener un protocolo de encaminamiento centralizado. Cada nodo debe tener suficiente capacidad para tomar decisiones de enrutamiento con la ayuda de los otros nodos. También es importante que el protocolo tenga en cuenta cuestiones como el consumo de potencia o la seguridad. En el nivel de enrutamiento sería deseable dar soporte tanto a la autenticación de los nodos vecinos como al cifrado de los datos.

En MANETs los protocolos de encaminamiento generalmente se clasifican en proactivos, reactivos e híbridos (mezcla de los anteriores).

El concepto de enrutamiento proactivo significa que todos los nodos intercambian información periódica con el propósito de mantener una visión lo más completa, consistente y actualizada posible de la red. Cada nodo usa esta información para calcular el coste hacia cada posible destino por los diferentes caminos encontrados hasta ese destino, seleccionando el mejor de los mismos como el que se utilizará. De esta manera, al establecer comunicación con un destino, dicha comunicación tendrá una ruta asignada, lo que evita retardos en la búsqueda de la misma. La principal ventaja de este tipo de protocolos es que no hay un retardo cuando se requiere una ruta para un destino. En contrapartida, esto se relaciona normalmente con un tiempo de convergencia (tiempo que emplean los encaminadores de una red en conocer la topología de la misma una vez que se ha producido un cambio) más largo que en los protocolos reactivos, especialmente cuando la movilidad es alta. Un ejemplo de protocolo proactivo es el OLSR [3] (*Optimized Link State Routing Protocol*).

En los protocolos reactivos, cuando una ruta es requerida, por ejemplo cuando se inicia una comunicación con un nuevo nodo, el nodo emisor debe empezar un proceso de descubrimiento de la ruta hasta ese destino. Esto significa que debe diseminar una petición de ruta a través de la red y esperar una respuesta, lo que provoca un retardo y causa un considerable gasto de recursos. Este esquema, sin embargo, evita el mantenimiento constante de las rutas que no se utilizan y, en principio, usará más eficientemente la memoria. Un ejemplo de protocolo reactivo es el AODV [4] (*Ad hoc On Demand Distance Vector*).

2.2.5 – Seguridad

Los requisitos de seguridad de una red móvil *ad hoc* son los mismos que los existentes en redes tradicionales (confidencialidad, integridad, autenticación, no repudiación y disponibilidad). Sin embargo, las características generales de una MANET hacen del cumplimiento de los requisitos de seguridad un problema mucho más complejo de abordar, mostrando la dificultad de diseñar una solución general en términos de seguridad sobre un escenario móvil *ad hoc*.

La política de seguridad a aplicar en un entorno *ad hoc* dependerá, en gran medida, de la aplicación y del escenario concretos para los que se realiza el despliegue de la red, por tanto las propuestas de seguridad se centran en aspectos específicos del problema.

Generalizando se podrían identificar tres aspectos clave que deben ser cubiertos por cualquier política de seguridad en redes *ad hoc*: sistemas de detección de intrusos, seguridad de los protocolos de encaminamiento y servicios de gestión de claves.

Las técnicas de prevención, tales como el cifrado y la autenticación, son necesarias como primera línea de defensa. Sin embargo, una MANET presenta vulnerabilidades inherentes que no son fácilmente previsibles. La detección de intrusos permite establecer una segunda línea de defensa, y puede ser necesaria en beneficio del requisito de disponibilidad.

2.3 – Aplicaciones

La clave del éxito de una nueva tecnología de red depende de las aplicaciones que de ella puedan beneficiarse. Son numerosas las aplicaciones potenciales de las redes *ad hoc* que sacan partido de la ausencia de infraestructura y del hecho de la comunicación entre iguales para comunicarse con otros dispositivos móviles, aquí enumeramos algunas:

- Aplicaciones militares. Existen numerosas aplicaciones en campos de batalla de difícil acceso donde no existe, por supuesto, infraestructura previa. Estas redes suponen un importante avance para este tipo de situaciones ya que se pueden implementar en blindados, aviones y otros elementos móviles.
- Redes para zonas de difícil acceso. Estas aplicaciones se realizan en lugares donde no es posible o no resulta rentable instalar una red cableada debido a la topología del terreno, con lo cual una red de tipo *ad hoc* sería una buena alternativa como solución de cara a las comunicaciones.
- Servicios de emergencia. Estas aplicaciones se hacen necesarias en caso de desastre natural (huracanes, inundaciones, terremotos, erupciones, etc.) ya que no es posible asegurar un correcto funcionamiento de red cableada o infraestructura fija previa.
- Redes comunitarias o de empresa. Son redes donde los puntos de acceso pueden comunicarse entre ellos y enlazar nodos correspondientes a diferentes subredes, por ejemplo una comunidad de vecinos.

- Redes de sensores. Este tipo de redes permite infinidad de aplicaciones posibles (por ejemplo, en el campo de la domótica) donde un elemento central puede controlar varios dispositivos domésticos que implementan sensores. Las redes de sensores también pueden desplegarse en cualquier medio y realizar funciones de monitorización del mismo o trabajar a nivel industrial controlando el inventario de una fábrica en tiempo real.
- Servicios contextuales [5]. La tecnología inalámbrica también puede utilizarse para proporcionar servicios de información dependientes del contexto, denominadas *context-aware*. Los usuarios con dispositivos inalámbricos pueden recibir información adecuada en todo momento, dependiendo del lugar donde se encuentren. En museos, ferias, monumentos o centros comerciales el visitante recibiría información de la obra o producto que esté mirando a través de su terminal inalámbrico, que sería detectado de forma automática por el sistema.
- VANET (*Vehicular Ad hoc Network*) [6] o redes MANET entre coches. Los vehículos equipados con dispositivos inalámbricos pueden comunicarse con los dispositivos inalámbricos de uso personal tales como PDAs, intercambiando información almacenada mediante la formación de una red *ad hoc*. De igual forma, los usuarios pueden programar sus dispositivos de uso personal para buscar información de interés relacionada con restaurantes próximos y reserva de hoteles. La tecnología *ad hoc* también permite la formación de redes inalámbricas entre vehículos de forma transparente. Así, por ejemplo, diferentes vehículos pueden intercambiar y propagar mensajes de alerta del estado de las carreteras y accidentes rápidamente y a una gran distancia.

Referencias de este capítulo:

[1] *IETF MANET Working Group*:
http://www.ietf.org/html.charters/manet-charter.html.

[2] Lista y definición de los protocolos de encaminamiento para MANET en Wikipedia:
http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list.

[3] T.Clausen y P.Jacquet, "Optimized Link State Routing Protocol (OLSR)". *Request For Comments* 3626, *MANET working group*, octubre 2003:
http://www.ietf.org/rfc/rfc3626.txt.

[4] Charles E.Perkins, Elizabeth M.Belding-Royer y Samir R.Das, "Ad hoc On Demand Distance Vector (AODV)". *Request For Comments* 3561, julio 2003:
http://www.ietf.org/rfc/rfc3561.txt.

[5] "Context-Aware Applications Survey", Mari Korkea-aho, *Helsinki University of Technology*, recurso en línea: http://users.tkk.fi/~mkorkeaa/doc/context-aware.html.

[6] Definición de VANET en la Wikipedia: http://en.wikipedia.org/wiki/VANET.

Capítulo 3

Estándar de comunicaciones Bluetooth

3.1 – Introducción al estándar 802.15.1

3.1.1 – Definición y objetivos

Bluetooth [1] es un una tecnología que, después de haber sido desarrollada por el *Bluetooth Special Interest Group* (SIG), ha sido convertida en la norma IEEE 802.15.1 que define un estándar global de comunicación inalámbrica de área personal (WPAN) para la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia. La tecnología Bluetooth comprende hardware, software y requerimientos de interoperabilidad.

Los principales objetivos que se pretenden con esta norma son:

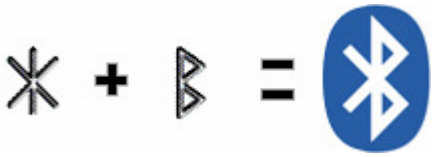
- Crear un sistema universal, capaz de operar en todo el mundo.
- Capacidad de establecer comunicación entre dispositivos que cumplan con la especificación Bluetooth, cualesquiera que sea su naturaleza: PC, teléfono móvil, accesorio de automóvil, PDA, etc.
- Facilitar la comunicación entre equipos móviles y fijos, eliminando cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas (*piconets*).
- Facilitar la sincronización de datos entre nuestros equipos personales.

3.1.2 – Etimología

El nombre *Bluetooth* proviene del rey danés del siglo X llamado Harald Blåtand (derivado de la palabra *blå* que significa "piel oscura" y *tand*, que significa "gran hombre"). Harald I unificó bajo su mandato las tribus en guerra de Noruega, Suecia y Dinamarca y fue reconocido por su capacidad de ayudar a la gente a comunicarse y su carácter humanitario. Posteriormente el nombre Blåtand fue traducido e interpretado en inglés como Bluetooth, "diente azul".

Análogamente al rey vikingo, Bluetooth intenta ser el puente entre diferentes sectores tecnológicos como el de los ordenadores, los teléfonos móviles y el resto de periféricos.

El logo de Bluetooth combina la representación de las runas nórdicas *Hagalaz* (trascrito por 'H') y *Berkana* (trascrito por 'B') en un mismo símbolo, es decir, las iniciales del rey Harald I.



3.1.3 – Historia

En 1994, Ericsson Mobile Communications, la compañía global de telecomunicaciones con base en Suecia, comenzó un estudio de la viabilidad de una interfaz de radio de baja potencia y bajo coste entre teléfonos móviles y otros accesorios, con el objetivo de eliminar los cables. El estudio era parte de un proyecto más amplio que investigaba cómo conectar diferentes dispositivos de comunicaciones a la red celular a través de un teléfono móvil. La compañía determinó que el último enlace de este tipo de conexión debería ser un enlace de radio de corto alcance. A medida que progresaba este proyecto, se hizo evidente que este tipo de enlace radio de corto alcance podía ser utilizado ampliamente en un gran de número de aplicaciones.

El trabajo de Ericsson atrajo la atención de IBM, Intel, Nokia y Toshiba. Estas compañías decidieron formar en febrero de 1998 un grupo especial de investigación denominado *Bluetooth Special Interest Group* (SIG) [2], con el objetivo de desarrollar, promover, definir y publicar las especificaciones de esta tecnología. En mayo del mismo año, se invitó a otras compañías a participar en el grupo: Microsoft, Lucent Technologies, 3COM y Motorola.

En julio de 1999, el grupo publicó la especificación Bluetooth 1.0, la cual constaba de dos documentos: el núcleo fundamental (*core*) y el perfil fundamental. El primer documento proporcionaba las especificaciones de diseño, tales como el interfaz radio, la capa de banda base, el gestor de enlace, el protocolo de descubrimiento de servicios, el nivel de transporte y la interoperabilidad con diferentes protocolos de comunicaciones. Por otra parte, el perfil fundamental proporcionaba las directrices para la interoperabilidad de aplicaciones Bluetooth.

El SIG creció hasta alcanzar más de 1800 miembros en abril de 2000. Del grupo inicial de compañías promotoras, 3COM se retiró y Lucent Technologies cedió su calidad de miembro a su filial Agere Systems. De todas las compañías adjuntas al SIG, Intel es la única de todas que no ha fabricado un producto basado en Bluetooth.

3.1.4 – Especificaciones técnicas

La frecuencia de radio con la que trabaja Bluetooth se sitúa en la banda de 2.4 GHz en el rango de 2.402 a 2.480 GHz con amplio espectro y saltos de frecuencia con posibilidad de transmitir en *full duplex* con un máximo de 1600 saltos/seg. Los saltos de frecuencia se dan entre un total de 79 frecuencias o canales con intervalos de 1 MHz; esto permite dar seguridad y robustez. Hay que tener en cuenta que la banda de los 2.4 GHz es la denominada ISM (*Industry Science Medicine*), que no necesita licencia y que está disponible en casi todo el mundo. En España esta banda estaba restringida hasta hace poco, permitiendo únicamente el uso de 23 canales en el caso de Bluetooth. Afortunadamente, en el reciente Cuadro Nacional de Atribución de Frecuencias (CNAF [3]) de 2005 se ha solventado este problema, permitiendo a los servicios de radiocomunicaciones operar en toda la banda de 2.4 GHz a 2.5 GHz.

Bluetooth puede integrarse en una amplia variedad de dispositivos y es, además, un módulo radio de baja potencia, consume un 97% menos que un teléfono móvil, y es inteligente: cuando el tráfico de datos disminuye el transmisor adopta modos de funcionamiento de bajo consumo de energía. Su alcance nominal es de 10 cm a 10 m en teoría, pero puede extenderse a 100 m mediante el aumento de la potencia de transmisión (pero con mayor distorsión). Según la potencia de transmisión, se distinguen 3 clases de productos:

- Clase 1: 100 mW / 20 dBm, con un rango de ~ 100 m.
- Clase 2: 2.3 mW / 4 dBm, con un rango de ~ 10 m.
- Clase 3: 1 mW / 0dBm, con un rango de ~ 1 m.

A partir de la versión 1.0 que se ratificó en julio de 1999, se han publicado sucesivas versiones:

- Versión 1.1:
  - Soluciona erratas de la especificación 1.0.
  - Añade el Indicador de Calidad de la Señal Recibida (RSSI, en sus siglas en inglés).
- Versión 1.2:
  - Implementa la técnica de salto en frecuencia, *Adaptative Frecuency Hoping*, para mejorar la resistencia a interferencias.
  - Introduce el tipo de enlace para aplicaciones de audio *extended Synchronous Connection* (eSCO) que mejora la calidad de voz.
  - Mejoras en el *Host Controller Interface* (HCI) para una sincronización más rápida de las comunicaciones.
- Versión 2.0:
  - Nueva versión compatible con la anterior 1.x.
  - Incorpora la tecnología *Enhanced Data Rate* (EDR), que incrementa las velocidades de transmisión hasta 3 Mbps.
  - Reducción del consumo de energía a pesar del incremento de velocidad.

Las frecuencias radio utilizadas por Bluetooth permiten transmitir a través de objetos sólidos no metálicos. El protocolo de banda base es una combinación de circuitos y paquetes que la hace apropiada para voz y datos, soportando transmisión de tres canales de voz, vídeo y datos con una capacidad de transmisión que varía según las versiones:

- Versión 1.1: 721 Kbps
- Versión 1.2: 1 Mbps
- Versión 2.0 + EDR: 2.1 ~ 3 Mbps

Se definen dos tipos de enlaces para soportar aplicaciones de voz y datos:

- Enlace asíncrono sin conexión (ACL, *Asynchronous Connectionless*):
  - Conexiones simétricas o asimétricas punto-multipunto entre maestro y esclavo.
  - Conexión utilizada para tráfico de datos.
  - Sin garantía de entrega, se retransmiten paquetes.
  - La máxima velocidad de envío es de 721 Kbps en una dirección y 57.6 Kbps en la otra.
- Enlace síncrono orientado a conexión (SCO, *Synchronous Connection Oriented*):
  - Conexiones simétricas punto a punto entre maestro y esclavo.
  - Conexión capaz de soportar voz en tiempo real y tráfico multimedia.
  - Velocidad de transmisión de 64 KB/s.

La tecnología Bluetooth se implementa con transceptores de corto alcance que tienen un precio objetivo de tan sólo 5$. Mediante Bluetooth se pueden crear pequeñas redes inalámbricas denominadas *piconets*, donde un dispositivo actúa como maestro e interconecta a un máximo de 7 dispositivos esclavos. El maestro puede actuar también como un *proxy*, *router* o puente entre una red fija y los esclavos a los que da servicio.

3.2 – Arquitectura del protocolo Bluetooth

3.2.1 – Pila de protocolos

La pila o *stack* de protocolos de Bluetooth es algo compleja (figura 3-1). Esto se debe a requisitos de versatilidad, permitiendo adaptarse mejor a todas las aplicaciones que puede soportar. La especificación Bluetooth utiliza una arquitectura de protocolos que divide las diversas funciones de red en un sistema de niveles. En conjunto, permiten el intercambio transparente de información entre aplicaciones diseñadas de acuerdo con dicha especificación y fomentan la interoperabilidad entre los productos de diferentes fabricantes.

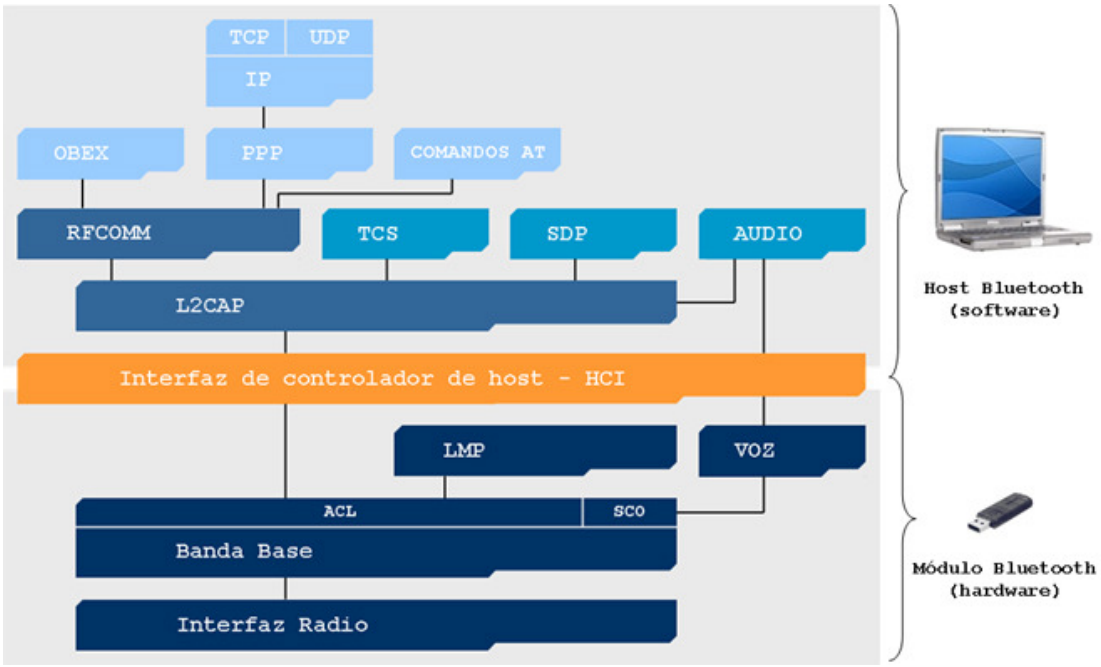


Figura 3-1: Pila de protocolos Bluetooth

La pila de protocolos Bluetooth se divide en dos zonas, cada una de las cuales se implementa en distintos procesadores:

- El **módulo Bluetooh** (hardware), encargado de las tareas relacionadas con el envío de información a través del interfaz de radiofrecuencia.
- El **host Bluetooth** (software), encargado de la parte relacionada con las capas superiores de enlace y aplicación.

Ambas zonas están comunicadas por le Interfaz de Controlador de Host (HCI, *Host Controller Interface*).

Sobre la capa de protocolos específicos de Bluetooth, cada fabricante puede implementar su capa de protocolos de aplicación propietarios. De esta forma, la especificación abierta de Bluetooth expande enormemente el número de aplicaciones que pueden beneficiarse de sus capacidades. Sin embargo, la especificación Bluetooth exige que, a pesar de la existencia de diferentes pilas de protocolos de aplicación

propietarios, se mantenga la interoperabilidad entre dispositivos que implementen diferentes pilas.

Las implementaciones de la pila de protocolos Bluetooth más conocidas son Widcomm, Toshiba Bluetooth Stack, IVT BlueSoleil Stack y Microsoft XP Bluetooth. Linux dispone tanto de pilas con licencia propietaria (Bluetooth Host Stack, Bluetooth Software Suite y BlueStack) como con licencia GPL (OpenBT, BlueZ, BlueDrekar y Affix). BlueZ es la que ha sido utilizada en este proyecto y se explica con más detalle en el capítulo 4.

La pila Bluetooth se basa en el modelo de referencia OSI (*Open Systems Interconnect*) de ISO (*International Standard Organization*) para interconexión de sistemas abiertos. La comparación entre las pilas OSI y Bluetooth (figura 3-2) es útil para relacionar partes de cada una, aunque la concordancia exacta entre ambas es reducida. Dado que el modelo de referencia es una pila ideal y bien particionada, compararlos sirve para resaltar la división de funciones que se da en Bluetooth.

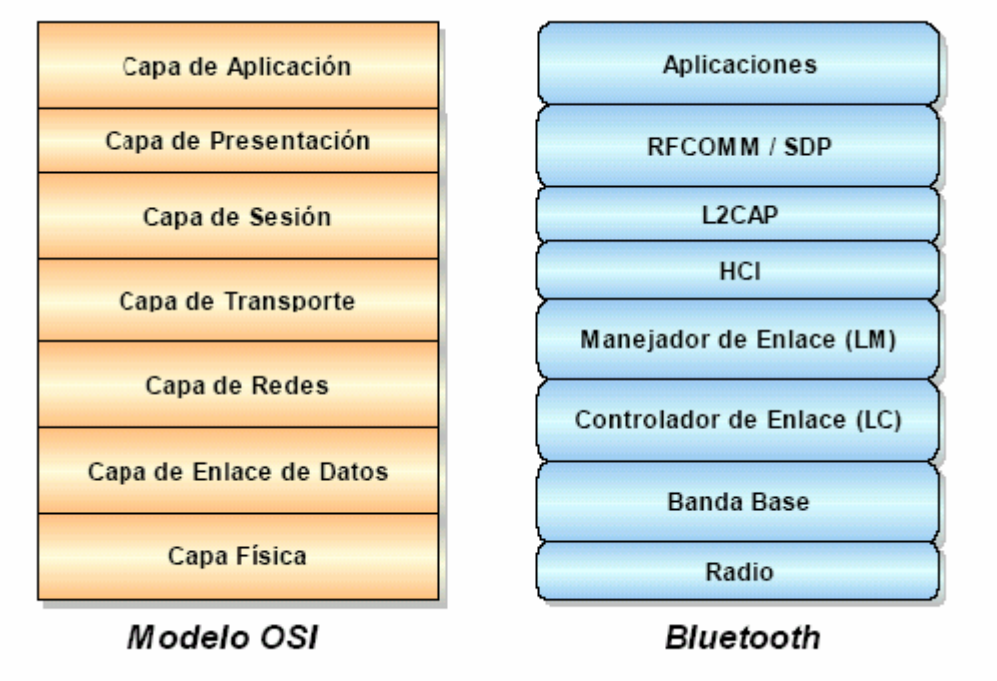


Figura 3-2: Modelo de referencia OSI y Bluetooth

A continuación se describe el funcionamiento de cada capa de la pila de protocolos Bluetooh.

3.2.2 – Radio

La capa de radio Bluetooth corresponde a la capa física de la pila de protocolos. Es la responsable de la transmisión a través del aire.

Como consecuencia de operar en la banda de 2.4 GHz ISM Bluetooth tiene que afrontar una gran cantidad de interferencias. Para mitigar este problema se usa *Frequency Hoping*, que consiste en que, en lugar de utilizar un solo canal durante toda la

transmisión, se va saltando de canal cada 625 µs. Así, si una interferencia afecta a un determinado canal, tendrá muy poca influencia sobre la transmisión total.

ISM rango regulatorio	2.400 - 2.4835 GHz
Numero de canales	79
Separación entre Canales	1 MHz
Frecuencias de Canales	2402 + k MHz, k = 0...78
Banda de Guarda Inferior	2 MHz
Banda de Guarda Superior	3.5 MHz

Tabla 3-1: Uso básico de frecuencias en Bluetooth

Bluetooth define 79 canales, cada uno de los cuales separado 1 MHz. El primer canal comienza en 2.402 GHz dejando 2 MHz de banda de guarda que asegura que no se transmitirá nada fuera de la banda ISM. El canal superior, que está situado en 2.48 GHz, deja una banda de guarda de 3,5 MHz.

3.2.2.1 -Modulación

La modulación empleada es GFSK (*Gaussian Frequency Shift Keying*). El índice de modulación debe estar entre 0.28 y 0.35. Un uno lógico es representado por una desviación de frecuencia positiva sobre la frecuencia nominal, y un cero lógico por una desviación de frecuencia negativa. La desviación de frecuencia mínima no debe ser menor de 115 KHz, y la máxima tiene que estar entre 140 y 175 KHz (ver figura 3-3).

El error de cruce por cero es la diferencia entre el periodo ideal del símbolo y el tiempo de cruce, que nunca debe ser inferior que ±1/8 de un periodo de símbolo. Se puede simplificar la demodulación limitando el discriminador FM. Este esquema de modulación permite la implementación de unidades de radio de bajo coste y se caracteriza por su robustez y su baja eficiencia espectral.

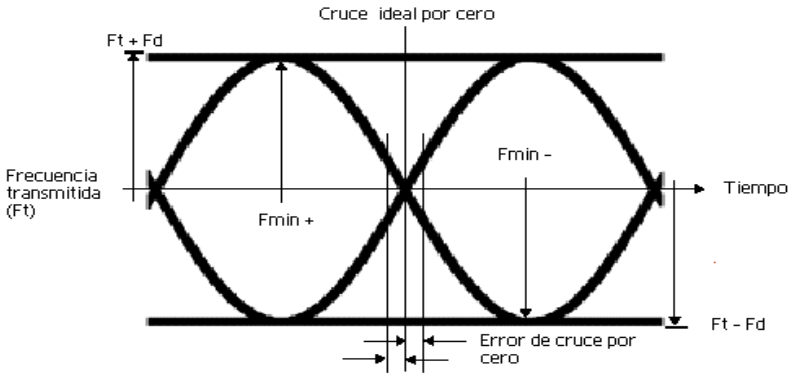


Figura 3-3: Modulación en Bluetooth

Existen otros esquemas de modulación definidos para la versión 2.0, que se pueden consultar en [4].

3.2.3 – Banda base

3.2.3.1 – Descripción general

La capa de banda base implementa el canal físico real. Esta capa también controla la sincronización de las unidades Bluetooth y la secuencia de saltos en frecuencia. Además, es la responsable de la información para el control de enlace a bajo nivel como el reconocimiento, control de flujo y caracterización de carga útil.

Bluetooth utiliza el esquema TDD (*Time-Division Duplex*) para la comunicación de varios dispositivos en modo *full duplex*. Se divide el canal en ranuras (*slots*) de 625 µs de duración. La información se transmite en paquetes, en saltos de frecuencia diferentes, para incrementar la protección frente a interferencias. En una transmisión cada paquete debe estar alineado con el inicio de una ranura. Un paquete ocupa normalmente una única ranura, pero puede ocupar hasta cinco consecutivas (en la figura siguiente se muestra la transmisión en una única ranura, en tres y en cinco).

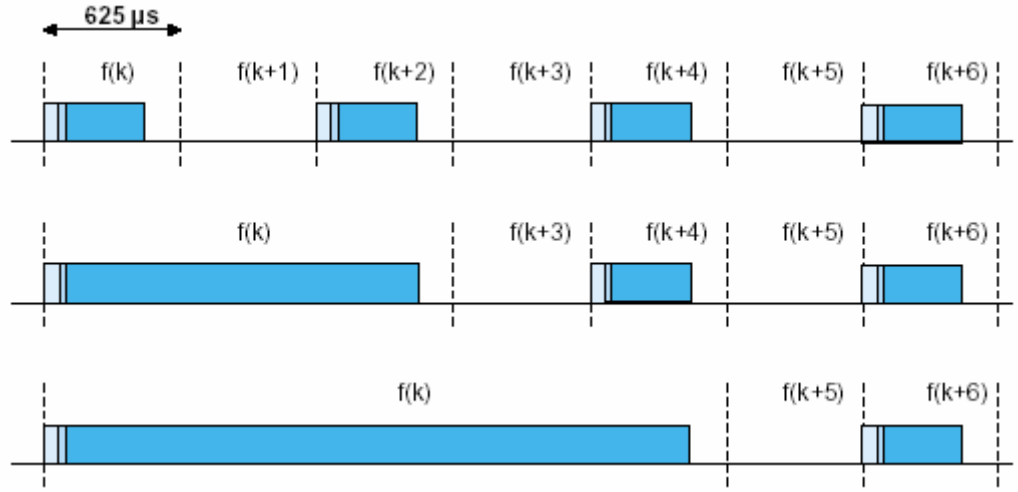


Figura 3-4: Transmisión de paquetes en una sola ranura y multiranura

Bluetooth emplea una combinación de conmutación de paquetes y de circuitos. Puede soportar un canal asíncrono, hasta tres canales síncronos simultáneos para voz o una combinación de ambos esquemas.

En Bluetooth se pueden establecer conexiones punto-a-punto (sólo dos unidades, ver figura 3-5a) o conexiones punto a multipunto (figura 3-5b). Dos o más unidades que comparten el mismo canal forman una *piconet*. En una *piconet* un dispositivo actúa como maestro y el resto como esclavos, hasta un máximo de siete. Todos los dispositivos conectados a la *piconet* están sincronizados con el canal en salto y tiempo. La unión de varias *piconet*s constituye una *scatternet* (figura 3-5c). Para ello, al menos una unidad actúa como maestro en una *piconet* y como esclavo en otra u otras.

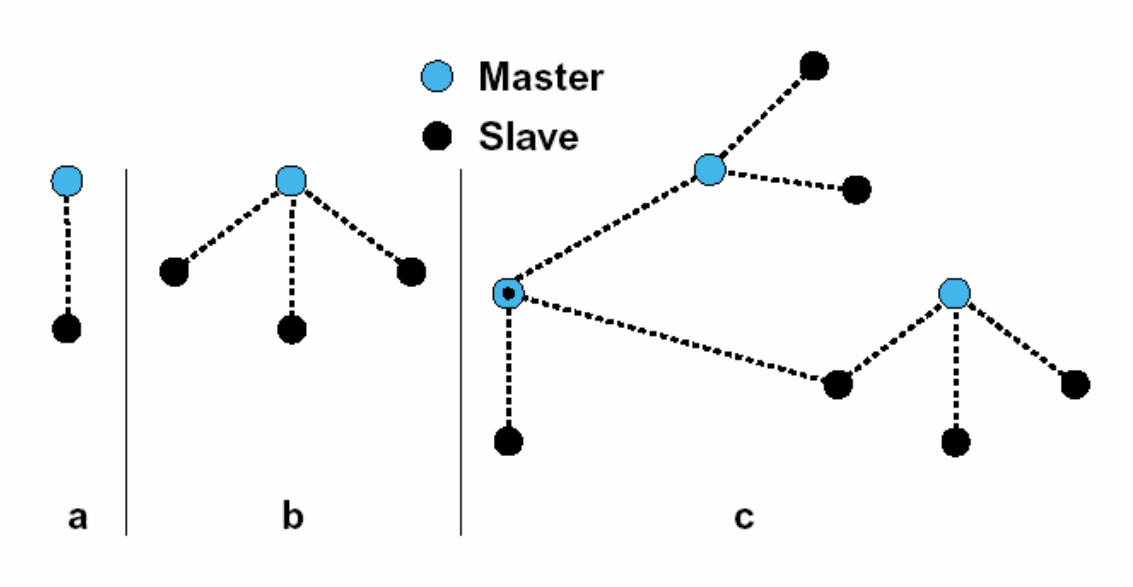


Figura 3-5: *piconet*s punto a punto (a), punto multipunto (b) y *scatternet* (c)

En funcionamiento normal, el maestro de la *piconet* reserva ranuras para sus esclavos, y los sondea de forma periódica. El maestro envía paquetes a un esclavo determinado en una ranura reservada temporalmente, y el esclavo en cuestión envía sus paquetes en la ranura siguiente. Para evitar fallos en la transmisión, el maestro inicia enviando en los *timeslots* pares y los esclavos en los impares. En la figura 3-6 se puede observar este esquema de transmisión.

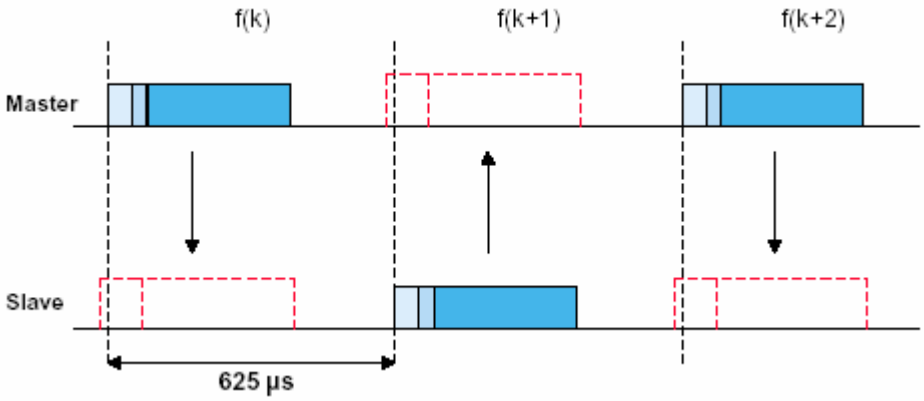


Figura 3-6: Funcionamiento TDD en una *piconet*

3.2.3.2 – Enlace físico

Como se dijo anteriormente, existen dos tipos de enlaces: síncronos y asíncronos, o SCO (*Synchronous Connection-Oriented*) y ACL (*Asynchronous Connection-Less*). En los enlaces SCO se reservan ranuras para el tráfico a intervalos regulares, por lo que se puede considerar una conmutación de circuitos. El maestro puede utilizar las ranuras no utilizadas por enlaces SCO para establecer enlaces ACL.

El enlace ACL es un enlace punto-a-multipunto entre el maestro y uno o más esclavos activos en la *piconet*. Este enlace de comunicación es un tipo de conexión de conmutación de paquetes. Todos los paquetes son retransmitidos para asegurar la integridad de los datos. El maestro puede enviar mensajes *broadcast* (de difusión) a todos los esclavos conectados dejando vacía la dirección de paquete; así, todos los esclavos recibirán el paquete.

Se pueden intercambiar distintos tipos de paquetes (Tabla 3-2). Los más importantes son:

- FHS: Es un paquete especial de control que, además de otros cometidos, informa de la dirección Bluetooth y el reloj del emisor. Este paquete se usa en respuestas al paginado del maestro, en las respuestas a *inquiries* y en los cambios de maestro a esclavo.
- DMx: Son paquetes de información, protegidos por códigos de recuperación de errores (FEC). Dependiendo de las ranuras necesarias para la transmisión "x" puede ser 1, 3 o 5.
- DHx: A diferencia de los paquetes DMx, no van protegidos con códigos FEC. Nuevamente, dependiendo de las ranuras necesarias para la transmisión "x" puede ser 1, 3 o 5.

En la siguiente tabla se muestran los distintos tipos de paquetes DM y DH para enlaces ACL y las velocidades de transmisión que ofrece Bluetooth (en teoría y sin tener en cuenta la versión 2.0, que llega a triplicar estos valores). Con DH5 se consigue la mayor velocidad de transmisión, pero sólo en condiciones óptimas dado que la interferencia tiene un fuerte efecto en dicha velocidad.

Tipo de paquete ACL	Cabecera (Bytes)	Carga útil de usuario (Bytes)	FEC	CRC	Velocidad de transmisión simétrica (max Kbps)	Velocidad de transmisión asimétrica (max Kbps)	
						Subida	Bajada
DM1	1	0-17	2/3	Si	108.8	108.8	108.8
DH1	1	0-27	0	Si	172.8	172.8	172.8
DM3	2	0-120	2/3	Si	258.1	387.2	54.4
DH3	2	0-180	0	Si	390.4	585.6	86.4
DM5	2	0-224	2/3	Si	286.7	477.8	36.4
DH5	2	0-338	0	Si	433.9	723.2	57.6

Tabla 3-2: Tipos de paquetes ACL y velocidad de transmisión

3.2.3.3 – Paquetes

Los datos enviados sobre el canal de la *piconet* son convertidos en paquetes. Éstos son enviados y el receptor los procesa comenzando por el bit menos significativo. Como se observa en la figura 3-7, el formato de paquete general consta de tres campos: código de acceso, cabecera y carga útil.

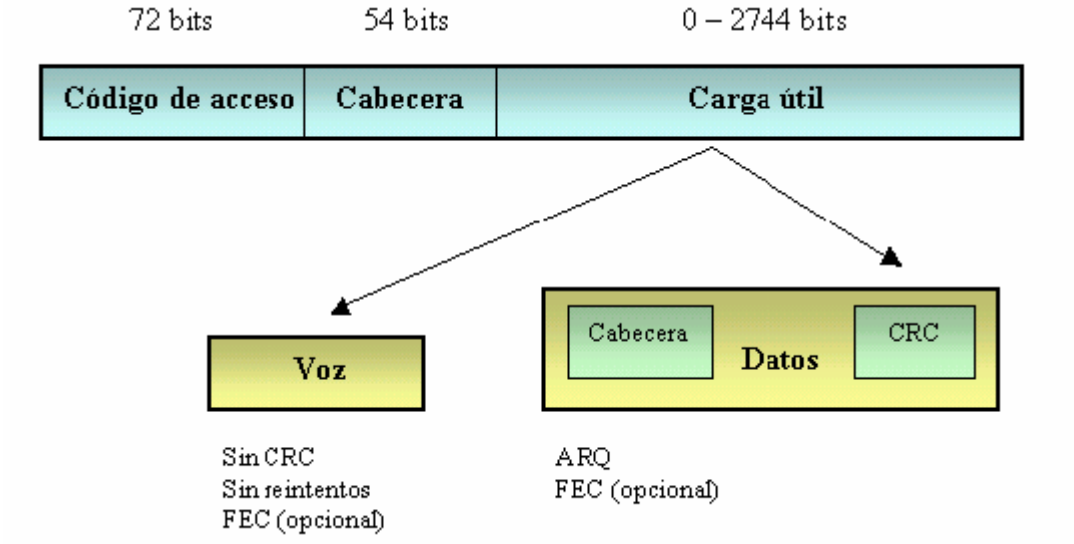


Figura 3-7: Formato de paquete general

- **Código de acceso**. Es usado para sincronización e identificación. Todos los paquetes comunes que son enviados sobre el canal de la *piconet* están precedidos del mismo código de acceso. Existen tres tipos diferentes de código:
  - **Código de acceso al canal (CAC)**: Para identificar los paquetes sobre el canal de la *piconet*.
  - **Código de acceso de dispositivo (DAC)**: Para procedimientos de señalización especiales, *paging* (servicio para transferencia de señalización o información en un sentido), entre otros.
  - **Código de Acceso de Búsqueda (IAC)**: Llamado IAC general cuando se quiere descubrir a otras unidades Bluetooth dentro del alcance radio, o IAC dedicado cuando se desea descubrir unidades de un tipo específico.
- **Cabecera**. La cabecera del paquete consta de seis campos (figura 3-8):
  - **Dirección**: Una dirección de dispositivo para distinguirlo de los demás dispositivos activos en la *piconet*.
  - **Tipo**: Define qué tipo de paquete es enviado.
  - **Flujo**: El bit de control de flujo es usado para notificar al emisor cuándo el buffer del receptor está lleno.
  - **ARQN**: *Acknowledge Receive Data* o reconocimiento de datos recibidos.
  - **SEQN**: *Sequential Numbering* o numeración secuencial para ordenar los datos sobre el canal.
  - **HEC**: Chequeo de redundancia cíclica de cabecera.

Figura 3-8: Formato de cabecera de paquete

- **Carga útil (*Payload*)**. La carga útil de un paquete se divide en dos campos:
  - **Campo de Voz**: Consta de datos de voz de longitud fija y existe en paquetes de alta calidad de voz y paquetes combinados de datos-voz. No es necesaria ninguna cabecera de carga útil.
  - **Campo de Datos**: Consta de tres partes, cabecera de carga útil, datos de carga útil y CRC (Código de Redundancia Cíclica).

3.2.3.4 – Direcciones Bluetooth

Cada *transceiver* (transmisor-receptor) Bluetooth tiene una única dirección de dispositivo de 48 bits asignada (BD_ADDR). La BD_ADDR es considerada la dirección MAC de los dispositivos Bluetooth, por lo que a partir de ahora hablaremos indistintamente de dirección Bluetooth, BD_ADDR o dirección MAC. Esta dirección debe ser obtenida a través del *IEEE Registration Authority*, y se divide en tres campos:

- LAP: Dirección baja de 24 bits.
- UAP: Dirección alta de 8 bits.
- NAP: Dirección no significativa de 16 bits.

Figura 3-9: Dirección de dispositivo Bluetooth

La dirección del dispositivo es conocida públicamente y puede ser obtenida a través de una llamada a la función *inquiry*, que veremos más adelante.

3.2.3.5 – Control de canal

En Bluetooth hay tres estados principales: *standby, connection y park*. Adicionalmente existen siete subestados, a saber: *page, page scan, inquiry, inquiry scan, master response, slave response* e *inquiry response*. Los subestados son estados internos que se utilizan para establecer conexiones y permitir el descubrimiento de dispositivos. Para pasar de un estado a otro se utilizan comandos del *Link Manager* o comandos internos

del *Link Controller*. La figura 3-10 muestra la representación de los estados y la interacción entre ellos.

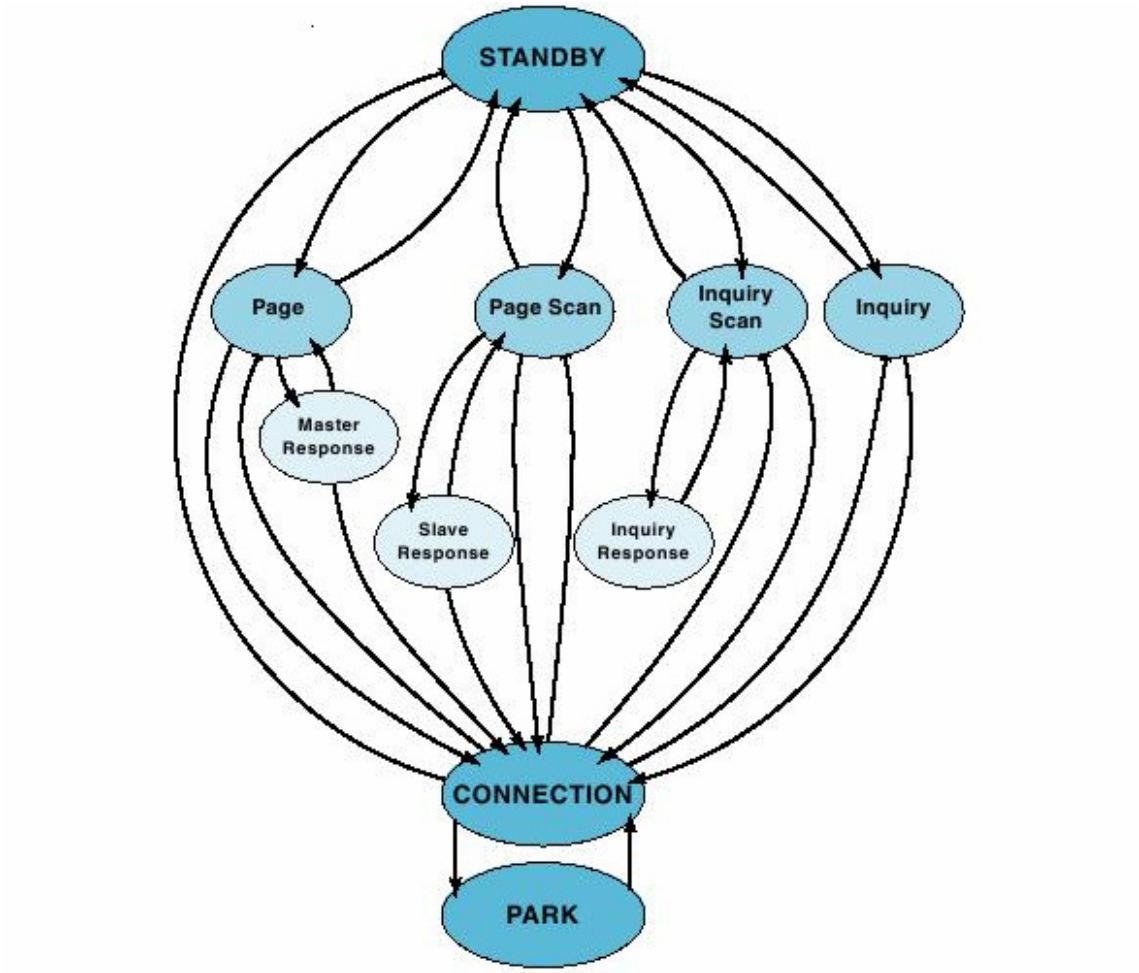


Figura 3-10: Diagrama de estados de un dispositivo Bluetooth

En Bluetooth se define un procedimiento de búsqueda que se usa en aplicaciones donde la dirección del dispositivo de destino es desconocida para la fuente. Esto puede ser usado para descubrir qué otras unidades Bluetooth están dentro del alcance radio. Durante un subestado de *inquiry* o búsqueda, la unidad de descubrimiento recoge la dirección del dispositivo y el reloj de todas las unidades que respondan al mensaje de búsqueda; entonces la unidad puede iniciar una conexión con alguna de las unidades descubiertas.

El mensaje de búsqueda difundido por la fuente no contiene información de ella. Sin embargo, puede indicar qué clase de dispositivos deberían responder. Una unidad que permita ser descubierta entra, regularmente, en un subestado de *inquiry scan* para responder a los mensajes de búsqueda.

Existen dos formas de detectar otras unidades. La primera, detecta todas las otras unidades en el rango de cobertura; la segunda, detecta un tipo específico de unidades. Los esclavos que se encuentran en el subestado de *page scan*, escuchan esperando su propio código de acceso de dispositivo. El maestro en el subestado *page* activa y conecta a un esclavo. El maestro trata de capturar al esclavo transmitiendo repetidamente el código de acceso de dispositivo en diferentes canales de salto. Debido

a que los relojes del maestro y del esclavo no están sincronizados, el maestro no sabe exactamente cuándo y en qué frecuencia de salto se activará el esclavo.

Después de haber recibido su propio código de acceso de dispositivo, el esclavo transmite un mensaje de respuesta. Este mensaje de respuesta es simplemente el código de acceso de dispositivo del esclavo. Cuando el maestro ha recibido este paquete, envía un paquete de control con información acerca de su reloj, dirección, clase de dispositivo, etc.

El esclavo responde con un nuevo mensaje donde envía su dirección. Si el maestro no obtiene esta respuesta en un determinado tiempo, él reenvía el paquete de control. Si el esclavo excede el tiempo de espera, entonces retorna al subestado de *page scan*. Si es el maestro quien lo excede, entonces retorna al subestado de *page* e informa a las capas superiores.

Cuando se establece la conexión, la comunicación se inicia con un paquete de sondeo desde el maestro hacia el esclavo. Como respuesta se envía un nuevo paquete de sondeo y, de esta forma, se verifica que la secuencia de salto y la sincronización sean correctas. En la figura 3-11 se muestra la inicialización de la comunicación sobre el nivel banda base.

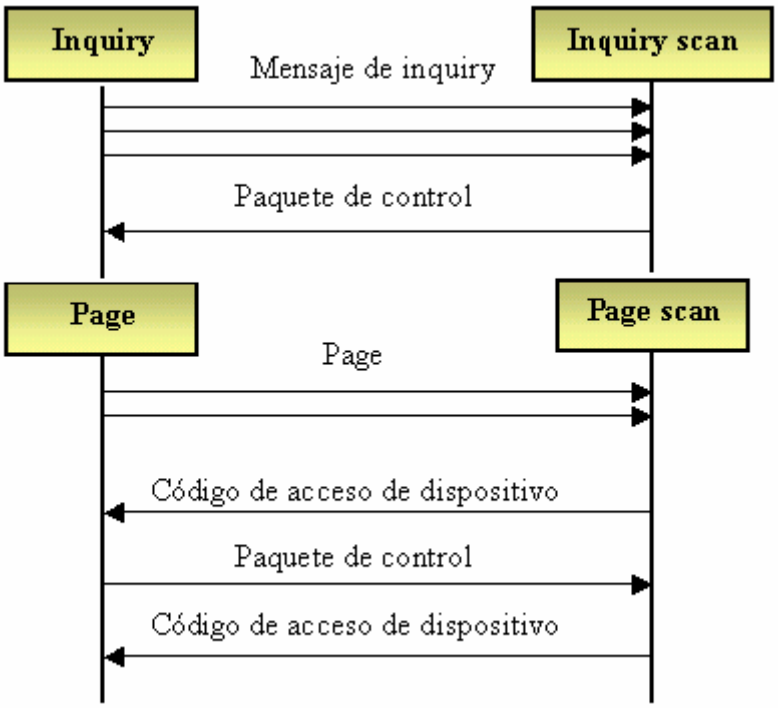


Figura 3-11: Iniciación de comunicación sobre el nivel de banda base

3.2.4 – Protocolo de gestión del enlace (LMP)

En el protocolo de gestión de enlace (*Link Manager Protocol*) se usan mensajes asociados con el establecimiento, seguridad y control. Los mensajes son enviados en la carga útil y no en los mensajes de datos de L2CAP. Los mensajes LMP son separados de los demás por medio de un valor reservado en uno de los campos de la cabecera de carga útil. Todos los mensajes LMP son extraídos e interpretados por la capa LMP del receptor. Esto significa que ningún mensaje es enviado a capas superiores.

Los mensajes LMP tienen mayor prioridad que los datos de usuario. Esto significa que, si la gestión de enlace necesita enviar un mensaje, éste no debe ser retrasado por otro tráfico. Solamente las retransmisiones de los paquetes del nivel de banda base pueden retrasar los mensajes LMP. Además, éstos no necesitan rutinas de reconocimiento ya que la capa banda base asegura un enlace confiable. El protocolo de gestión del enlace soporta mensajes para:

- Autenticación.
- Paridad.
- Cifrado.
- Temporización y sincronización.
- Versión y características.
- *Role Switch* (cambio de rol) para desempeño como maestro o esclavo dependiendo de si el dispositivo es quien inicia (maestro) o no (esclavo) el enlace con otro dispositivo.
- Petición de nombre.
- Desconexión.
- Modo **hold**: el maestro ordena al esclavo entrar en este estado para ahorro de potencia.
- Modo **sniff**: para envío de mensajes en ranuras temporales específicas.
- Modo **park**: para que el esclavo permanezca inactivo pero sincronizado en la *piconet*.
- Enlaces SCO.
- Control de paquetes multiranura.
- Supervisión de enlace.

3.2.5 –Interfaz de controlador de host (HCI)

La capa HCI (*Host Controller Interface*) actúa como frontera entre las capas de protocolo relativas al hardware (módulo Bluetooth) y las relativas al software (host Bluetooth). Proporciona una interfaz de comandos para la comunicación entre el dispositivo y el firmware del módulo Bluetooth y permite disponer de una capa de acceso homogénea para todos los módulos Bluetooth de banda base, aunque sean de

distintos fabricantes. Es la capa HCI la que hace posible, por tanto, que el host pueda controlar completamente el dispositivo Bluetooth.

La comunicación entre el host y el dispositivo se realiza a través de paquetes HCI, que contienen datos de usuario o comandos. Como respuesta a un comando, se envía de vuelta al host un paquete de evento. Por tanto, hay tres tipos de paquetes HCI:

- Paquetes de datos de usuario. Son transferidos por el nivel de banda base a la *piconet* y pueden ser de dos tipos:
  - Paquete de datos ACL (*ACL Data Packets*).
  - Paquete de datos SCO (*SCO Data Packets*).
- Paquetes de comandos (*HCI Command Packets*). HCI presenta comandos de control del enlace, de administración de la *piconet*, de acceso al hardware de la banda base, de acceso a la información de los dispositivos, de verificación del hardware, de control de flujo, de control de errores, etc. Algunos de estos comandos son:
  - HCI_reset: resetea el dispositivo Bluetooth.
  - HCI_create_connection: crea una conexión ACL.
  - HCI_read_BD_ADDR: lee la dirección Bluetooth del dispositivo.
  - HCI_disconnect: desconecta el enlace.
  - HCI_inquiry: invoca la búsqueda de dispositivos.
- Paquetes de eventos (*HCI Event Packets*). Contienen los resultados de los comandos que han sido ejecutados. También se envían cuando ocurre un evento en la *piconet*, por ejemplo después de que se haya realizado una conexión o cuando ocurre una desconexión o después de una respuesta al *inquiry*.

3.2.6 –Protocolo de control y adaptación de enlace lógico (L2CAP)

L2CAP se encuentra sobre el protocolo de gestión de enlace (LMP) y reside en la capa de enlace de datos. L2CAP permite a protocolos de niveles superiores y a las aplicaciones la transmisión y recepción de paquetes de datos L2CAP de hasta 64 kbytes, con capacidad de multiplexación de protocolo, operación de segmentación y reensamble y transmisión de información sobre la calidad de servicio.

Para cumplir sus funciones, L2CAP espera que la banda base suministre paquetes de datos en *full duplex*, que realice el chequeo de integridad de los datos y que reenvíe los datos hasta que hayan sido reconocidos satisfactoriamente. Las capas superiores que se comunican con L2CAP son, por ejemplo, el protocolo de descubrimiento de servicio (SDP), el RFCOMM y el control de telefonía (TCS).

La especificación L2CAP está definida únicamente para enlaces asíncronos sin conexión (ACL), y no puede existir más que un enlace entre dos dispositivos.

3.2.6.1 – Canales

L2CAP está basado en el concepto de canales. Localmente, cada canal lógico L2CAP tiene asociado un único identificador de canal (CID, *Channel Identifier*), aunque éste no coincide necesariamente con el CID usado por el dispositivo remoto para identificar el otro *endpoint* del mismo canal. Los CIDs están divididos en dos grupos, uno con identificadores reservados para funciones L2CAP (del 0x0000 al 0x003F) y otro con identificadores libres para implementaciones particulares.

Los canales de datos orientados a la conexión representan una conexión entre dos dispositivos, donde un CID identifica cada *endpoint* del canal.

Los canales no orientados a la conexión limitan el flujo de datos a una sola dirección. La señalización de canal es un ejemplo de un canal reservado. Este canal es usado para crear y establecer canales de datos orientados a la conexión y para negociar cambios en las características de esos canales. Su CID es el 0x0001.

Los niveles de la pila de protocolos por encima del L2CAP pueden ser identificados por el valor del PSM (*Protocol Sevice Multiplexor*), un campo abstracto que trabaja básicamente como un puerto TCP. Los dispositivos remotos solicitan una conexión a un PSM particular y L2CAP asigna un CID. Puede haber distintos canales abiertos transportando los mismos datos PSM.

3.2.6.2 – Segmentación y reensamblado

Los paquetes de datos definidos por el protocolo banda base están limitados en tamaño. Los paquetes L2CAP grandes deben ser segmentados en varios paquetes banda base más pequeños antes de transmitirse, y luego deben ser enviados a la gestión de enlace. En el receptor los pequeños paquetes recibidos de la banda base son reensamblados en paquetes L2CAP más grandes. Varios paquetes banda base recibidos pueden ser reensamblados en un solo paquete L2CAP seguido de un simple chequeo de integridad. La segmentación y reensamblado, SAR, funcionalmente es absolutamente necesaria para soportar protocolos usando paquetes más grandes que los soportados por la banda base. La figura 3-12 muestra la segmentación L2CAP.

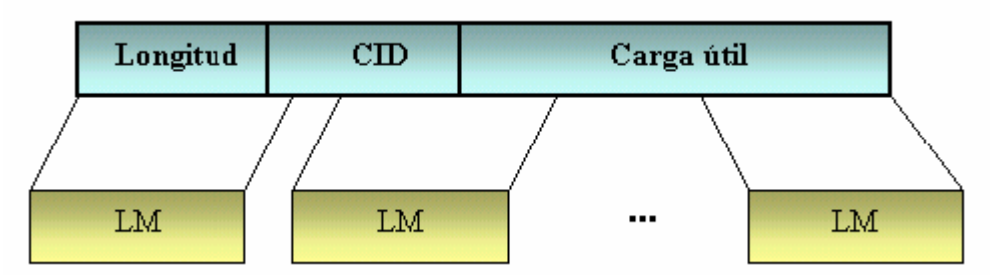


Figura 3-12: Segmentación L2CAP

3.2.6.3 – Eventos

Todos los mensajes y *timeouts* que entran en la capa L2CAP, son llamados eventos. Los eventos se encuentran divididos en cinco categorías: indicaciones y confirmaciones de capas inferiores, peticiones de señal y respuestas de capas L2CAP, datos de capas L2CAP, peticiones y respuestas de capas superiores, y eventos causados por expiraciones de tiempo.

3.2.6.4 – Acciones

Todos los mensajes y *timeouts* enviados desde la capa L2CAP son llamados acciones (en el lado del receptor estas acciones son llamadas eventos). Las acciones se encuentran divididas en cinco categorías: peticiones y respuestas a capas inferiores, peticiones y respuestas a capas L2CAP, datos a capas L2CAP, indicaciones a capas superiores, y configuración de *timers*.

3.2.6.5 – Formato del paquete de datos

Como se puede observar en la figura 3-13, los paquetes de canal orientado a la conexión están divididos en tres campos: longitud de la información, identificador de canal, e información.

Figura 3-13: Paquete L2CAP orientado a conexión

Los paquetes de canal de datos no orientados a la conexión son iguales a los paquetes orientados a la conexión pero adicionalmente incluyen un campo con información multiplexada de protocolo y servicio.

3.2.6.6 – Calidad de servicio (QoS)

La capa L2CAP transporta la información de calidad de servicio a través de los canales y brinda control de admisión para evitar que canales adicionales violen contratos de calidad de servicio existentes. Algunos esclavos pueden requerir un alto rendimiento o una respuesta rápida.

Antes de que un esclavo con grandes peticiones sea conectado a una *piconet*, el esclavo trata de obtener una garantía a sus demandas. Puede solicitar un determinado ratio de transmisión, tamaño del *buffer* de tráfico, ancho de banda, tiempo de recuperación de datos, etc. Por lo tanto, antes de que el maestro conecte a un nuevo esclavo o actualice la configuración de calidad, debe chequear si posee ranuras temporales y otros recursos libres.

3.2.7 –Protocolo de descubrimiento de servicios (SDP)

El descubrimiento de servicios hace referencia a la capacidad de buscar y encontrar servicios disponibles en dispositivos Bluetooth. A través de los servicios, dos dispositivos pueden ejecutar aplicaciones comunes e intercambiar datos.

Un servicio es cualquier entidad que puede ofrecer información, ejecutar una acción o controlar un recurso. Un servicio puede estar implementado como hardware, software o una combinación de ambos.

El protocolo SDP (*Service Discovery Protocol*) permite a una aplicación cliente obtener información sobre servidores SDP disponibles en otros dispositivos Bluetooth cercanos, enumerar los servicios que ofrecen y las características de dichos servicios. Para hacer más fácil la búsqueda, el SDP la habilita sin un previo conocimiento de las características específicas de los servicios. El SDP solamente soporta el descubrimiento del servicio, no la llamada del servicio.

3.2.7.1- Service Class

Un servicio concreto soportado por cierto dispositivo es una instancia de un *Service Class* o clase de servicio. El *Service Class* describe los servicios genéricos soportados por un dispositivo:

Para dar a conocer los servicios genéricos que soporta un dispositivo Bluetooth, este incorpora en la cabecera de nivel de banda base de sus paquetes un campo *Class of Device/Service* que contiene información acerca de su *Service Clas*s.

El campo reservado para el *Service Class* se compone de 11 bits, del bit 23 al 13. En la especificación de banda base 1.1 de Bluetooth, se describe la siguiente relación entre los bits marcados en el campo *Service Class* y los servicios genéricos soportados por el dispositivo (Tabla 3-3):

Bit no	Major Service Class
13	Limited Discoverable Mode [Ref #1]
14	(reserved)
15	(reserved)
16	Positioning (Location identification)
17	Networking (LAN, Ad hoc, ...)
18	Rendering (Printing, Speaker, ...)
19	Capturing (Scanner, Microphone, ...)
20	Object Transfer (v-Inbox, v-Folder, ...)
21	Audio (Speaker, Microphone, Headset service, ...)
22	Telephony (Cordless telephony, Modem, Headset service, ...)
23	Information (WEB-server, WAP-server, ...)

Tabla 3-3: ***Service Class***

3.2.7.2- Service Record

Toda la información relacionada con un servicio que mantiene un servidor SDP está contenida en un *Service Record* o registro individual de servicio.

Un *Service Record* consiste en una lista de atributos que describen características de un servicio: *Service Name*, *Service Description*, *Provider Name*, *Service Record Handle*, *Service Class ID List*, *Service Record State*, *Service ID*, *Protocol Description List*, *Browse Group List*, *Language Base Attribute ID List*, *Service Info Time To Live*, *Service Avaliability* y *Bluetooth Profile Descriptor List*.

El protocolo SDP permite realizar dos tipos de operaciones relacionadas con el descubrimiento de servicios en dispositivos Bluetooth: búsqueda y enumeración de servicios.

- La operación búsqueda de servicios (*Service Searching*) permite a un cliente SDP encontrar dispositivos que ofrecen un servicio específico.
- La operación enumeración de servicios (*Service Browsin*g) permite a un cliente SDP conocer los servicios ofrecidos por un determinado dispositivo.

En ambos casos, el resultado de la petición SDP devolverá al cliente que la originó una lista de servicios descubiertos acompañada por la definición de los mismos a través de sus *Service Records*.

3.2.7.3 – El protocolo

El protocolo de descubrimiento de servicio (SDP) usa un modelo petición/respuesta. A continuación se describen las distintas fases:

- Petición de búsqueda de servicio: se genera por el cliente para localizar registros de servicio que concuerden con un patrón de búsqueda dado como parámetro. Aquí el servidor examina los registros en su base de datos y responde con una respuesta a búsqueda de servicio.
- Respuesta a búsqueda de servicio: se genera por el servidor después de recibir una petición de búsqueda de servicio válida.
- Petición de propiedad de servicio: una vez el cliente ya ha recibido los servicios deseados, puede obtener mayor información de uno de ellos dando como parámetros el registro de servicio y una lista de propiedades deseadas.
- Respuesta a propiedad de servicio: el SDP genera una respuesta a una petición de propiedad de servicio. Ésta contiene una lista de propiedades del registro requerido.
- Petición de búsqueda y propiedad de servicio: se suministran un patrón de servicio con servicios deseados y una lista de propiedades deseadas que concuerden con la búsqueda.
- Respuesta de búsqueda y propiedad de servicio: como resultado se puede obtener una lista de servicios que concuerden con un patrón dado y las propiedades deseadas de estos servicios.

3.2.8 – RFCOMM

El protocolo RFCOMM (*Radio Frequency Communication*) es un protocolo de emulación de línea serie basado en el estándar ETSI TS 07.10. Proporciona una emulación de los puertos serie RS-232 sobre el protocolo L2CAP.

Este protocolo de "sustitución de cable serie" emula las señales de control y datos RS232 sobre la banda base, proporcionando capacidades de transporte a los servicios de niveles superiores que utilizan el cable serie como mecanismo de transporte.

Para los propósitos de RFCOMM, un camino de comunicación directa involucra siempre a dos aplicaciones que se ejecutan en dos dispositivos distintos extremos de la comunicación. Entre ellos existe un segmento que los comunica, en este caso, un enlace Bluetooth desde un dispositivo al otro. RFCOMM pretende soportar aquellas aplicaciones que utilizan los puertos serie de los dispositivos donde se ejecutan.

RFCOMM es un protocolo de transporte sencillo que soporta hasta 9 puertos serie RS232 y permite hasta 60 conexiones simultáneas (canales RFCOMM) entre dos dispositivos Bluetooth.

3.2.9 – Protocolo OBEX

OBEX (*OBject EXchange*) es un protocolo de nivel de sesión desarrollado originalmente por la asociación IrDA (*Infrared Data Association*) con el nombre de IrOBEX. Su objetivo es soportar el intercambio de objetos de forma simple y espontánea. OBEX se basa en el modelo cliente/servidor y es independiente del mecanismo de transporte, aunque en la implementación de OBEX sobre Bluetooth sólo se utiliza RFCOMM como nivel de transporte.

3.2.10 – Protocolos adoptados PPP

La especificación Bluetooth emplea varios protocolos existentes que se reutilizan para diferentes propósitos en los niveles superiores. El objetivo de la implementación de estos protocolos es permitir que aplicaciones antiguas funcionen con la tecnología inalámbrica Bluetooth y ayudar a asegurar un correcto funcionamiento e interoperabilidad de esas aplicaciones con aplicaciones modernas diseñadas específicamente para dispositivos Bluetooth.

Bluetooth utiliza el protocolo PPP desarrollado por el IETF (*Internet Engineering Task Force*), que define cómo se transmiten los datagramas IP sobre enlaces punto a punto, para garantizar la interoperabilidad de dispositivos Bluetooth con aplicaciones basadas en los protocolos TCP y UDP en última instancia.

3.2. 11 – Comandos AT

Bluetooth SIG ha definido el juego de comandos AT mediante los cuales un teléfono móvil y un módem pueden ser controlados en múltiples modelos de uso (como en el perfil de FAX o en el de auriculares). Los comandos AT son instrucciones codificadas y se denominan así por la abreviatura *attention*.

3.3 – Perfiles Bluetooth

El estándar Bluetooth fue creado para ser usado por un gran número de fabricantes e implementado en áreas limitadas. Para asegurar que todos los dispositivos que usen Bluetooth sean compatibles entre sí son necesarios esquemas estándar de comunicación en las principales áreas. Para evitar diferentes interpretaciones del estándar Bluetooth acerca de cómo un tipo específico de aplicación debería ser implementado, el SIG de Bluetooth ha definido modelos de usuario y perfiles de protocolo.

Un perfil define una selección de mensajes y procedimientos de las especificaciones Bluetooth y ofrece una descripción clara de la interfaz para servicios específicos. Un perfil puede ser descrito como una "rebanada" vertical completa de la pila de protocolos. Existen cuatro perfiles generales definidos en los cuales están basados directamente algunos de los modelos de uso más importantes y sus perfiles. Estos cuatro modelos son:

- Perfil Genérico de Acceso (GAP, *Generic Access Profile*).
- Perfil de Puerto Serie (SPP, *Serial Port Profile*).
- Perfil de Aplicación de Descubrimiento de Servicio (SDAP, *Service Discovery Application Profile*).
- Perfil Genérico de Intercambio de Objetos (GOEP, *Generic Object Exchange Profile*).

En la figura 3-14 se muestra la jerarquía de los perfiles, todos ellos contenidos en el Perfil Genérico de Acceso. A partir de los cuatro perfiles genéricos se definen perfiles específicos para modelos de uso. Estos perfiles Bluetooth son múltiples y variados, implementándose de manera opcional e independiente por cada fabricante y tipo de dispositivo. A continuación se citan algunos de ellos:

- Perfil de Telefonía Inalámbrica *(CTP, Cordless Telephony Profile).*
- Perfil de Acceso Telefónico a Redes (DUN, *Dial-Up Networking*).
- Perfil de Auriculares (HS, *HeadSet Profile*).
- Perfil de Fax (FP, *Fax Profile*).
- Perfil de Carga de Objetos (OPP, *Object Push Profile*).
- Perfil Básico de Impresión (BPP, *Basic Printing Profile*).
- Perfil de Acceso a Redes de Área Local (LAP, *Local Area Network Access Profile*).

Figura 3-14: Perfiles Bluetooth

Sin embargo para este trabajo sólo nos interesa el PAN (*Personal Area Networking*) Profile, pues ha servido como base para el desarrollo del mismo. Sus características se detallan a continuación, para más información acerca de los otros perfiles consúltese [5].

3.3.1 – Perfil de Red de Área Personal (PAN)

El perfil PAN [6] utiliza el protocolo BNEP [7] (*Bluetooth Network Encapsultaion Protocol*) para emular redes Ethernet, y es capaz de establecer varios tipos de configuraciones de red empleando:

- Emulación simple de red Ethernet mediante el encapsulado de tramas Ethernet en paquetes Bluetooth.
- IP sobre una *piconet* PAN.
- *Forwarding* del maestro cuando actúa como punto de acceso.

Esto significa que podemos obtener una IP dinámica, con soporte para IPv4 y IPv6, o usar uno de los dispositivos de la red como *router* o puente entre distintas tecnologías de red, creando un punto de acceso hacia una red que puede ser una LAN corporativa, Internet, GPRS u otro tipo de red de datos.

Asimismo el perfil PAN posibilita que pequeños dispositivos como PDAs, con una potencia y almacenamiento limitados, también puedan usar este protocolo y formar parte de cualquier red.

El perfil PAN depende del perfil de acceso genérico (GAP), por lo que cualquier dispositivo o aplicación que lo implemente ha de implementar también el perfil GAP.

En el perfil PAN se definen una serie de roles según el papel que efectúe el dispositivo dentro de la red:

- Usuario PAN (*PAN user*, **PANU**): Cliente de un NAP o un GN, explicados a continuación.
- Controlador de un grupo de red *ad hoc* (*Group ad hoc Network*, **GN**): Nodo que se encarga de hacer *forwarding* en una red de tipo *peer-to-peer* (una *piconet*, figura 3-15). Permite interconectar hasta 7 dispositivos activos (PANUs) en una red *peer-to-peer* real.

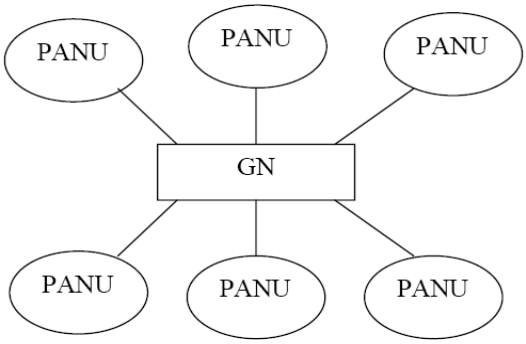


Figura 3-15: PAN con GN

- Punto de acceso a red (*Network Access Point*, **NAP**): Actúa como un *proxy*, *router* o puente entre una red fija (típicamente una LAN) y hasta 7 PANUs activos. En este escenario (figura 3-16), el controlador de host y la interfaz radio aparentan ser un bus de conexión directo a un dispositivo de interfaz de red con acceso a red. El NAP puede también configurar sus PANUs vía DHCP con lo que se pueden configurar de forma automática y acceder a la red.

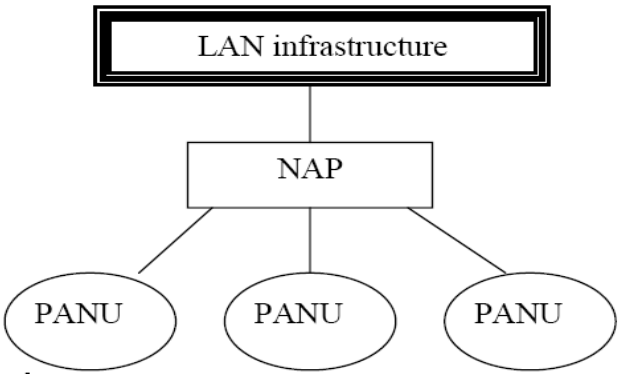


Figura 3-16: PAN con NAP

Algunos expertos señalan que el perfil PAN puede llevar a la desaparición de otros perfiles como DUN o LAP debido a su fácil configuración, y al hecho de que proporciona un fácil acceso a redes IP.

En las siguientes dos figuras se muestran los protocolos y las entidades usadas en las dos configuraciones en las que trabaja este perfil. Como se puede observar, BNEP trabaja sobre L2CAP.

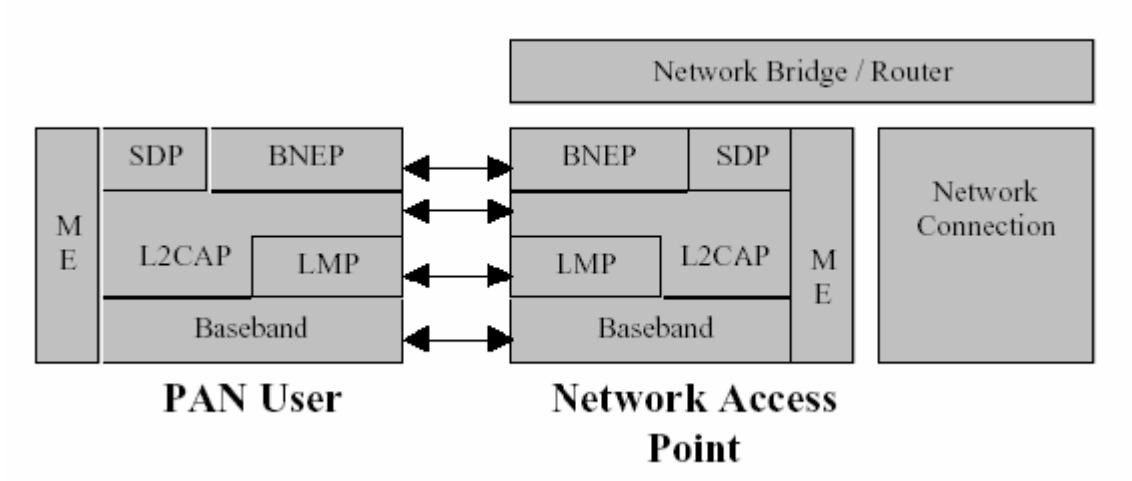


Figura 3-17: Uso de la pila de protocolos Bluetooth con el perfil PAN-NAP

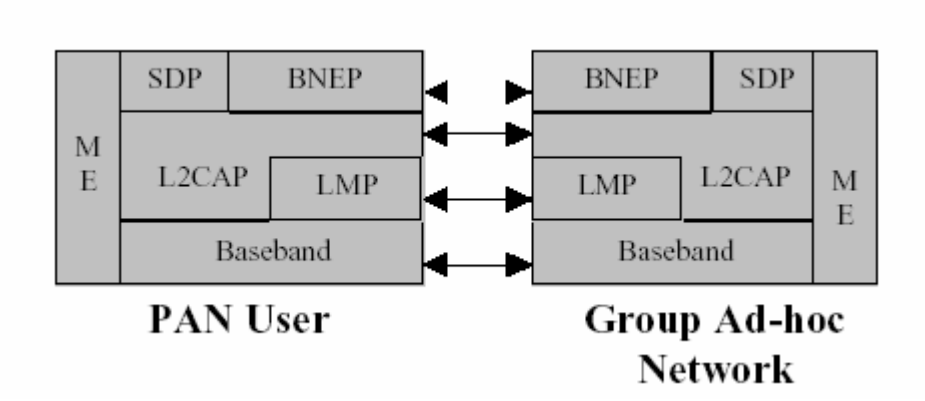


Figura 3-18: Uso de la pila de protocolos Bluetooth con el perfil PAN-GN

3-4 – Elementos de seguridad en Bluetooth

Bluetooth incorpora varios mecanismos de seguridad [8] que lo convierten en uno de los protocolos de comunicaciones más seguros y robustos frente a ataques y capturas de datos. Se definen mecanismos de seguridad en las siguientes capas del protocolo:

- Seguridad a nivel de banda base.
- Seguridad a nivel de enlace.

3.4.1 – Seguridad a nivel de banda base

Como ya se ha comentado, Bluetooth utiliza la técnica de salto de frecuencias (FHSS, *Frequency Hopping Spread Spectrum*) con el fin de evitar interferencias con otras tecnologías que operen en la misma banda de frecuencias. Esta técnica consiste en dividir la banda en 79 canales de longitud 1 MHz y realizar 1600 saltos por segundo.

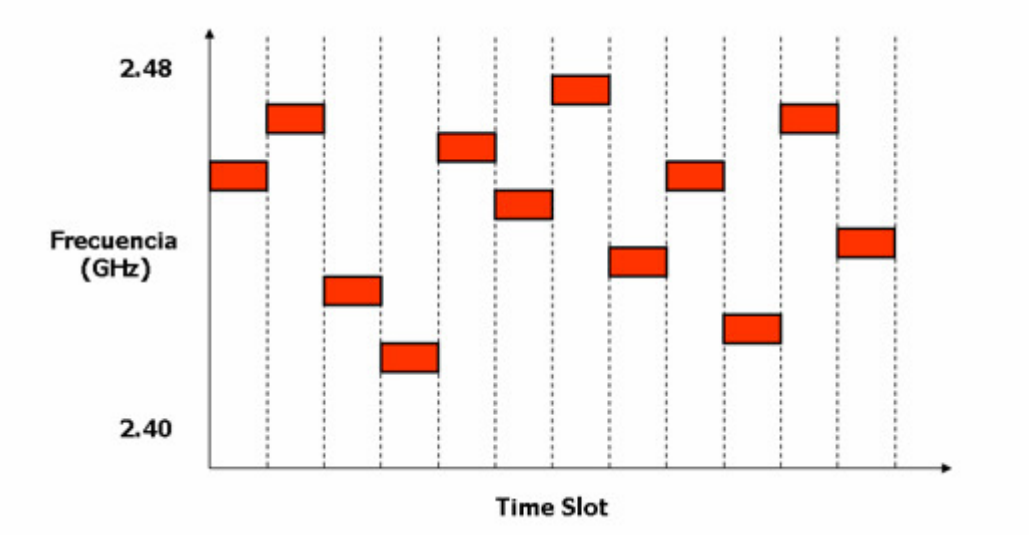


Figura 3-19: FHSS

Durante el proceso de establecimiento de la conexión en una *piconet*, el dispositivo maestro genera una tabla pseudo aleatoria con la secuencia o patrón de saltos de frecuencia que deben utilizar los dispositivos pertenecientes a la *piconet* durante las comunicaciones. El intercambio de la tabla de saltos desde el maestro hacia el esclavo (o esclavos) se realiza en un canal determinado del espectro de frecuencias, de forma que todos los dispositivos pueden acceder a ésta.

Cuando se establece la *piconet*, el dispositivo esclavo recibe un paquete FHS (*Frecuency Hop Synchronization*) que le permite sincronizar su reloj interno con el reloj del maestro agregando un desplazamiento a su reloj interno. Como los relojes funcionan con independencia, a lo largo de la comunicación se han de actualizar regularmente dichos desplazamientos.

Una vez comenzada la comunicación, el intercambio de paquetes de datos se realiza de acuerdo con el patrón de saltos de frecuencia establecido, y a una velocidad marcada

por el reloj interno. Esto significa que, en cada instante de tiempo, cada dispositivo escribirá o escuchará durante su ranura temporal en un determinado canal del espectro.

Cualquier dispositivo ajeno que no pertenezca a la *piconet* no podrá participar en la comunicación enviando paquetes o escuchando tráfico, ya que no dispone de la tabla con la secuencia de saltos utilizada en la *piconet* y, además, la probabilidad de adivinar cuál de todos los canales puede ser empleado para la comunicación en cada instante de tiempo es mínima.

Sin embargo, si un atacante pudiera disponer de la tabla de frecuencias generada por el dispositivo maestro de una *piconet*, éste podría sincronizar su módulo Bluetooth con el resto de dispositivos de la *piconet* y participar en la comunicación, capturando el tráfico e inyectando paquetes. Puesto que el intercambio de las tablas de secuencias de saltos se lleva a cabo en una frecuencia conocida, un dispositivo malicioso podría estar escuchando constantemente y capturar estas tablas de saltos de frecuencia para sincronizarse con una *piconet*.

Se puede concluir, por lo tanto, que la técnica de saltos de frecuencia empleada por Bluetooth refuerza en gran medida la seguridad del protocolo, pero en ningún caso garantiza totalmente la privacidad de la comunicación, ya que cabe la posibilidad de que cierto usuario no autorizado pueda conseguir acceso a la *piconet* y comprometa la confidencialidad del intercambio de datos y la integridad de los dispositivos participantes en la misma. Sin embargo, se trata de técnicas de ataque muy complejas y difíciles de llevar a la práctica en un entorno real.

3.4.2 – Seguridad a nivel de enlace

Se definen tres mecanismos de seguridad a nivel de enlace:

- Autenticación.
- Autorización.
- Cifrado de datos.

3.4.2.1 – Autenticación

La **autenticación** es el proceso por el cual un dispositivo Bluetooth verifica su identidad en otro dispositivo para poder acceder a los servicios que ofrece.

Todas las funciones de seguridad de nivel de enlace están basadas en el concepto de claves de enlace, las cuales son números pseudo aleatorios de 128 bits almacenados individualmente por cada par de dispositivos Bluetooth. La autenticación no requiere la intervención del usuario; implica un esquema de desafío/respuesta entre cada par de dispositivos que emplea una clave de enlace secreta común de 128 bits. Consecuentemente, este esquema se utiliza para autenticar dispositivos, no usuarios.

La primera vez que dos dispositivos intentan comunicarse, se utiliza un procedimiento de inicialización denominado emparejamiento (*pairing*) para crear una clave de enlace común de una forma segura. Para la primera conexión entre dos dispositivos, el procedimiento estándar de emparejamiento requiere que el usuario de cada dispositivo introduzca un código (cadena ASCII) de seguridad Bluetooth de hasta 16 bytes de

longitud, que debe ser el mismo en los dos casos. En primer lugar un usuario introduce el código de seguridad y, en segundo lugar, el otro usuario debe confirmar el mismo código de seguridad.

El código de seguridad Bluetooth a menudo es conocido como clave PIN (*Personal Identification Number*), aunque no se trata de un código que el usuario deba memorizar para mantenerlo en secreto, ya que se introduce una sola vez.

A partir del código de seguridad Bluetooth (PIN), se obtiene la clave de enlace común a dos dispositivos del siguiente modo:

1) Se genera una clave de inicialización común Kinit de 128 bits usando el algoritmo E22 a partir del código de seguridad Bluetooth (PIN), la longitud del mismo, la dirección BD_ADDR de 48 bits y un número aleatorio IN_RAND.

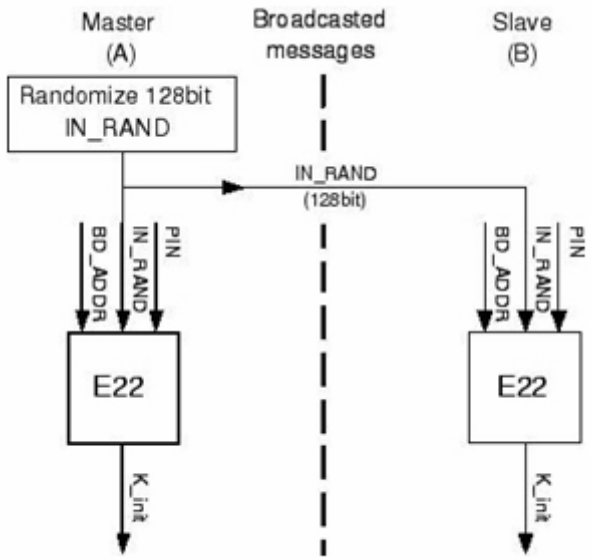


Figura 3-20: Generación de la clave Kinit

2) Se genera la clave de enlace Kab (figura 3-21) usando el algoritmo E21. Los dispositivos usan la clave de inicialización Kinit para intercambiar dos nuevos números aleatorios de 128 bits, conocidos como LK_RAND A y LK_RAND B. Cada dispositivo genera un número aleatorio y se lo envía al otro dispositivo haciendo un XOR previo bit-a-bit con Kinit. Dado que ambos dispositivos conocen Kinit, cada dispositivo conoce ambas LK_RAND. A partir de la dirección BD_ADDR y el LK_RAND, el algoritmo E21 obtiene la clave de enlace Kab.

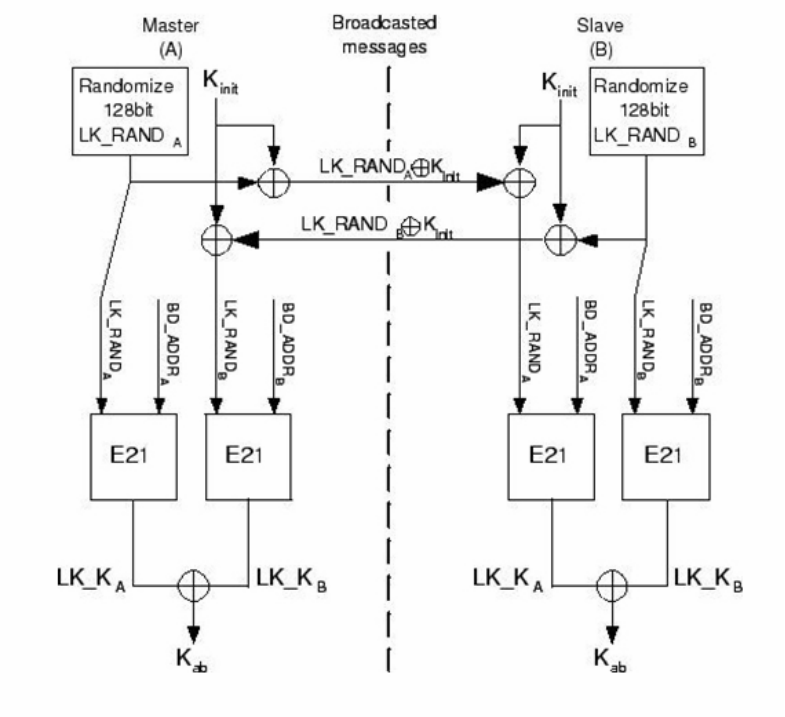


Figura 3-21: Generación de la clave Kab

3) La clave de enlace común se almacena temporalmente en los dispositivos emparejados. Mientras esta clave de enlace esté almacenada en ambos dispositivos, no es necesario repetir el emparejamiento en futuras conexiones. Si, por alguna razón, uno de los dos dispositivos ha borrado la clave de enlace común, debe repetirse el emparejamiento y los usuarios deben introducir de nuevo cualquier código de seguridad Bluetooth.

Una vez que los dispositivos emparejados disponen de la clave de enlace, utilizan esta clave común para autentificarse automáticamente en las sucesivas conexiones. El proceso de autenticación (figura 3-23) está basado en el esquema desafío/respuesta y transcurre de la siguiente forma:

1) El dispositivo reclamante envía su dirección BD_ADDR al dispositivo verificador.
2) El verificador devuelve un desafío aleatorio de 128 bits al demandante.
3) El reclamante usa el algoritmo E1 para generar la respuesta de autenticación (SRES) de 32 bits, usando como parámetros de entrada la dirección BD_ADDR del reclamante, la clave de enlace Kab almacenada y el desafío. El verificador realiza la misma operación en paralelo.
4) El reclamante devuelve la respuesta SRES al verificador.
5) El verificador comprueba la respuesta SRES recibida por el reclamante con la respuesta SRES calculada por él.
6) Si los valores de SRES coinciden, el verificador establece la conexión.

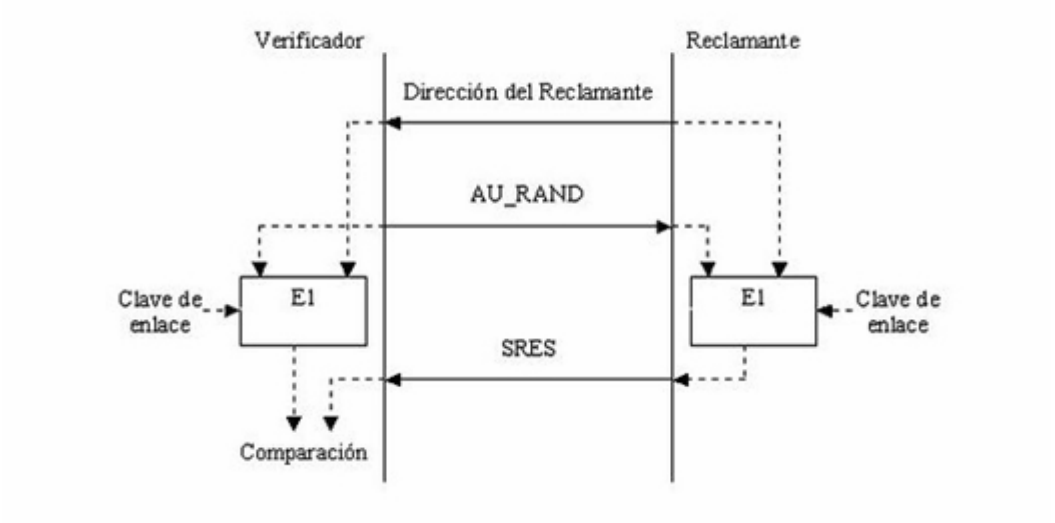


Figura 3-22: Proceso de autenticación

La especificación de Bluetooth establece que si se produce un fallo durante el proceso de autenticación, y para prevenir que un atacante pruebe claves de enlace aleatorias en un ataque de fuerza bruta, debe transcurrir cierto período de espera antes de que se pueda llevar a cabo un nuevo intento de autenticación. Para cada sucesivo intento fallido, el tiempo de espera aumenta exponencialmente.

3.4.2.2 – Autorización

La **autorización** es el procedimiento que determina los derechos que tiene un dispositivo Bluetooth para acceder a los servicios que ofrece un sistema.

El mecanismo de autorización en dispositivos Bluetooth se lleva a cabo mediante "niveles de confianza". Los dispositivos tienen tres niveles de confianza, los cuales determinan la capacidad de acceso a los servicios: total, parcial o restringida y nula.

- Un dispositivo de confianza total mantiene una relación de emparejamiento y dispone de acceso sin restricciones a todos los servicios.
- Un dispositivo de confianza restringida mantiene una relación de emparejamiento y sólo dispone de acceso restringido a uno o varios servicios, pero no a todos.
- Un dispositivo no confiable es aquel que puede o no mantener tener una relación de emparejamiento pero que no es de confianza. No se le permite el acceso a ningún servicio.

En el caso de que un determinado dispositivo de confianza total o parcial intente acceder a un servicio autorizado, no se requiere ningún procedimiento de confirmación, por lo que accede de forma transparente.

En el caso de que un determinado dispositivo no confiable intente acceder a un servicio restringido, se requiere un procedimiento explícito de confirmación por parte del usuario para permitir o denegar el acceso a ese dispositivo durante la sesión de conexión actual. Nótese que, para algunos servicios, es posible conceder permisos de acceso temporal a dispositivos no emparejados previamente.

3.4.2.3 – Cifrado de datos

El **cifrado de datos** protege la información que se transmite en un enlace entre dispositivos Bluetooth. Garantiza la confidencialidad del mensaje transmitido, de forma que si el paquete es capturado por un usuario que no posea la clave de descifrado, el mensaje le resultará ininteligible.

Su implementación es opcional, pero necesita que se haya producido anteriormente una autenticación. El maestro y el esclavo deben ponerse de acuerdo en utilizar cifrado o no. En caso afirmativo, deben determinar el tamaño de la clave de cifrado, para lo cual, maestro y esclavo intercambian mensajes hasta alcanzar un acuerdo. No siempre es posible llegar a un acuerdo sobre el tamaño de la clave, en este caso se indica a las unidades Bluetooth que no se les permite comunicarse utilizando cifrado en el enlace.

Tras esta negociación comienza el proceso de cifrado:

El maestro genera una clave de cifrado Kc de 128 bits usando el algoritmo E3, el cual requiere como parámetros de entrada un número aleatorio de 128 bits, la clave de enlace Kab generada durante el procedimiento de emparejamiento y número COF (*Ciphering Offset)* de 96 bits basado en el valor temporal ACO (*Authenticated Ciphering Offset*) calculado durante el procedimiento de autenticación.

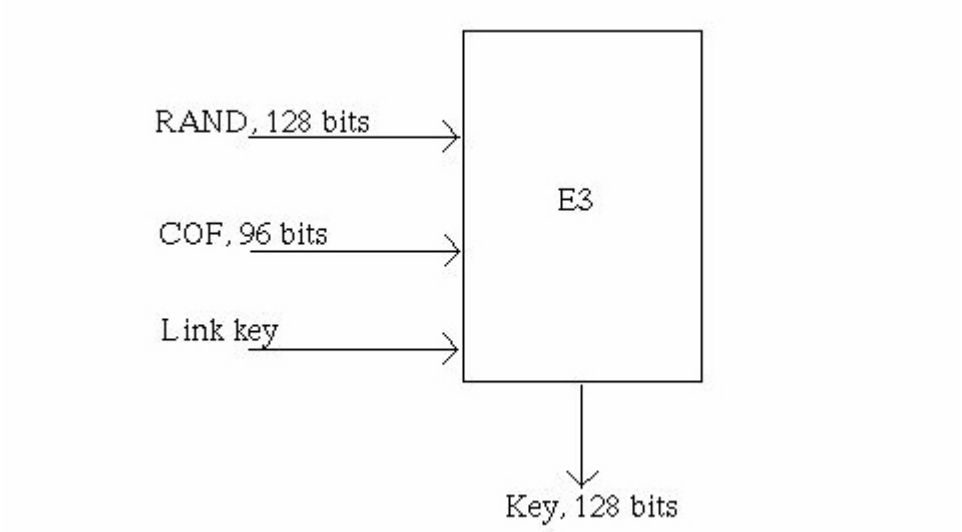


Figura 3-23: Generación de la clave de cifrado Kc

Una vez que la clave de cifrado se ha generado con éxito, el maestro se encuentra en condiciones de transmitir datos cifrados, para lo cual debe detener temporalmente el tráfico de datos de los niveles superiores y así evitar la recepción de datos corruptos.

3.4.2.4 – SAFER+

SAFER+ (*Secure And Fast Encryption Routine*) es un algoritmo simétrico de cifrado de datos de tipo IBC (*Iterated Block Ciphers*) que utiliza bloques de 128 bits.

Bluetooth hace uso del algoritmo de cifrado SAFER+ durante la generación de las claves de autenticación y cifrado.

Sin embargo, SAFER+ no es utilizado para cifrar el enlace de datos. Para esta función, Bluetooth utiliza el algoritmo 4LFSR, que es un cifrador de flujo adecuado para cifrado rápido de datos.

3.4.2.5 – Modos de seguridad

Una vez descritos los 3 mecanismos que emplea Bluetooth para reforzar la seguridad a nivel de enlace (autenticación, autorización y cifrado de datos), se definen 3 modos de seguridad a nivel de enlace en función de la implementación de los mismos:

- **Modo 1**: Ausencia de seguridad. Todos los mecanismos de seguridad (autenticación y cifrado) están deshabilitados. Además, el dispositivo se sitúa en modo promiscuo, permitiendo que todos los dispositivos Bluetooth se puedan conectar a él. Este modo lo emplean dispositivos que no tienen aplicaciones críticas.

  Ninguna parte del tráfico de datos es cifrada.

- **Modo 2**: Proporciona seguridad en los servicios a nivel de L2CAP. Utiliza mecanismos de seguridad después de establecerse el canal de comunicación: autorización. Un gestor de seguridad controla el acceso de los dispositivos a los diferentes servicios, en función de su nivel de confianza.

  La interacción con el usuario se limita a solicitar confirmación de la autorización de acceso a servicios restringidos por parte de dispositivos no autorizados.

  El tráfico de difusión no está cifrado, mientras que el tráfico punto-a-punto se cifra según las claves individuales generadas durante la conexión.

- **Modo 3**: Proporciona seguridad en el dispositivo a nivel de LMP. Utiliza mecanismos de seguridad antes de establecerse el canal de comunicación: autenticación. Requiere emparejamiento de dispositivos y existencia de clave de enlace compartida para validar la conexión entre dispositivos.

  La interacción con el usuario requiere la introducción de un código PIN para llevar a cabo el emparejamiento de dispositivos.

  Todo el tráfico se cifra con la clave de cifrado generada.

3.5 – Consideraciones finales

Una de las líneas de trabajo más importantes en redes Bluetooth de estos últimos años ha perseguido armonizar esta tecnología con IEEE 802.11, más conocida como Wi-Fi. Como las dos utilizan la banda libre de frecuencias de 2.4 GHz, cabría esperar en un principio que la interferencia mutua puede ser un problema. El grupo de trabajo TG2 [9] (*Task Group 2*) del IEEE 802.15 se dedica específicamente a esta cuestión, proponiendo mecanismos de coexistencia entre las dos tecnologías.

En [10] se estudia la interacción entre las tecnologías Bluetooth y IEEE 802.11b. En dicho trabajo se obtienen interesantes datos acerca del mutuo impacto entre Bluetooth y IEEE 802.11b en términos de caudal (*throughput*) TCP. Estos resultados se exponen en la figura 3-24, también extraída de [10]. Esta figura muestra como cuando dos dispositivos IEEE 802.11b operan sin interferencias de Bluetooth el *throughput* conseguido es de alrededor de 5 Mbit/s. En un entorno similar, cuando dos dispositivos Bluetooth operan sin interferencia de otros dispositivos IEEE 802.11b el *throughput* conseguido está alrededor de 0.5 Mbit/s. Sin embargo, cuando ambos pares de dispositivos se encuentran cerca y transmiten al mismo tiempo se produce un impacto mutuo considerable entre ellos. Esto se traduce en que el *throughput* entre los dos dispositivos IEEE 802.11b cae hasta los 4.2 Mbit/s y el de los dos dispositivos Bluetooth hasta los 0.25 Mbit/s, es decir, a la mitad.

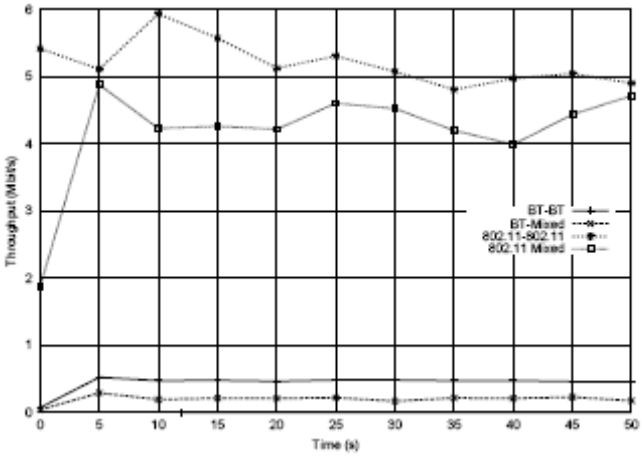


Figura 3-24: Impacto mutuo entre Bluetooth y IEEE 802.11 en términos de *throughput*

Hoy en día existen equipos en el mercado que integran tanto Wi-Fi como Bluetooth (por ejemplo, la solución de Intel Centrino). Asimismo hay que señalar que los protocolos IEEE 802.11a y IEEE 802.11h operan en la banda de 5 GHz, y por lo tanto, no producirían interferencias con Bluetooth. Sin embargo los protocolos IEEE 802.b y IEEE 802.g son los más populares y extendidos actualmente, y estos sí trabajan en la banda ISM de los 2.4 GHz. Para el año 2008 está previsto una mejora de todos estos protocolos Wi-Fi denominada IEEE 802.n, que alcanzaría una velocidad 10 veces superior a la IEEE 802.11g y que operaría tanto en la banda de 2.4 GHz como en la de 5 GHz, con lo que para su utilización en redes híbridas Bluetooth/Wi-Fi se podría seleccionar la banda de 5 GHz y evitar así las interferencias. Por último, señalar que la utilización de dispositivos Bluetooth 2.0, que alcanzan mayores velocidades de

transmisión, aseguran un *throughput* razonable para la funcionalidad que se requiere en este proyecto, aunque en la práctica este se pueda ver reducido por causa de interferencias.

Asimismo, también se producen interferencias entre *piconets* Bluetooth. En [11] los autores demuestran el impacto en el *throughput* debido a estas interferencias inter*piconet* y señalan que las condiciones ideales no siempre se consiguen y, por lo tanto, las aplicaciones han de ser diseñadas teniendo en cuenta las limitaciones en el ancho de banda de Bluetooth y sus fluctuaciones. Este último tipo de interferencias nos preocupan menos en este proyecto puesto que se ha considerado un escenario en el que están presentes dispositivos con tecnología IEEE 802.11 pero no múltiples *piconets* Bluetooth operando.

En este trabajo se asume que las interferencias pueden disminuir el ancho de banda disponible sin limitar la funcionalidad de la arquitectura; para confirmarlo hemos realizado pruebas en entornos exigentes, cuyos resultados se muestran más adelante, en el capítulo 5.

Pese a los inconvenientes comentados, el uso de la tecnología Bluetooth se hace imprescindible en el despliegue de redes *ad hoc*. Su principal baza es el bajo consumo y bajo coste, lo que le hace ideal para dispositivos móviles y ha posibilitado su popularidad y generalización en los mismos. Otra característica reseñable es la capacidad para encontrar dispositivos con la misma tecnología mediante un simple procedimiento de *inquiry*, así como descubrir los servicios que ofrecen dichos dispositivos y conectar con el que se desee de una forma muy sencilla para el usuario y sin necesidad de configuración previa. Esto último es especialmente importante para redes *ad hoc*, donde no existe ningún tipo de infraestructura previa.

Referencias de este capítulo:

[1] Página web oficial de Bluetooth: http://www.bluetooth.com.

[2] Página web oficial de los miembros del SIG: http://www.bluetooth.org.

[3] Cuadro Nacional de Atribución de Frecuencias (CNAF): http://www.mityc.es/Telecomunicaciones/Secciones/Espectro/cnaf/

[4] *Bluetoot Special Interest Group*, "Bluetooth Specification Core 2.0 + EDR", noviembre 2004.

[5] Tutorial de los perfiles Bluetooth en Palowireless "*Bluetooth Resource Center*": http://www.palowireless.com/infotooth/tutorial/profiles.asp.

[6] *Bluetooth Special Interest Group*, "Personal Area Networking Profile", junio 2001.

[7] *Bluetooth Special Interest Group*, "Bluetooth Network Encapsulation Protocol (BNEP) Especification", junio 2001.

[8] *Bluetooth Special Interest Group*, "Bluetooth Security Architecture. Version 1.0", julio 1999.

[9] Mecanismos de coexistencia entre redes WPAN y WLAN propuestos por el *IEEE 802.15 Coexistence Task Group 2 (TG2) for Wireless Personal Area Networks*: http://www.ieee802.org/15/pub/TG2-Coexistence-Mechanisms.html.

[10] Carlos T. Calafate, Roman García García y Pietro Manzoni, "Optimizing the implementation of a MANET routing protocol in a heterogeneous environment". En *The 8th Symposium on Computers and Communications (ISCC'2003)*, Kemer – Antalya, Turquía, julio 2003.

[11] C. de Morais Cordeiro, D. Sadok y D.P. Agrawal, "Piconet interference modelling and performance evaluation of Bluetooth MAC protocol". En *Proc. of IEEE GLOBECOM*, San Antonio, TX (25-29 de noviembre de 2001).

Capítulo 4

Realización del proyecto

4.1 – Introducción

En este capítulo se comenta el desarrollo del servicio Bluetooth **MANET_Gateway**, describiendo las implementaciones tanto del servidor que ofrece el servicio como del cliente que se conecta al servidor mediante el mismo. Dicho desarrollo se ha realizado para plataformas Linux, haciendo uso de la pila de protocolos BlueZ que viene incluida en el *kernel* de Linux.

El núcleo de la aplicación se ha desarrollado enteramente en lenguaje de programación **C** puesto que es el utilizado en BlueZ, mientras que la interfaz gráfica ha sido realizada en lenguaje **C++** utilizando la librería **Qt** de Trolltech [1], versión 3.3.4. Qt es una librería gráfica de desarrollo software.

En el apartado 4.2 se comentarán las características de **BlueZ** y sus herramientas principales. Se hace especial hincapié en el demonio `pand` puesto que es la base del desarrollo.

En los siguientes apartados se describirán las herramientas y tareas asociadas a la realización del proyecto. Primero se mostrará una definición de la **arquitectura del sistema**, donde se describen las características del sistema, el escenario o escenarios de aplicación del mismo, y un diagrama básico de comunicación entre el cliente y el servidor.

Seguidamente se describe el **funcionamiento** en sí del sistema, con un repaso a las funciones importantes implementadas por el servidor y por el cliente, respectivamente, así como una descripción en detalle de la comunicación entre ambos y la configuración del *gateway*.

Aunque inicialmente no estaban entre los objetivos principales del proyecto, los mecanismos de seguridad que se han implementado han merecido finalmente un esfuerzo considerable. La definición de estos mecanismos se realiza en un apartado independiente. Primero se realizó un estudio previo y después se desarrolló una solución de seguridad propia que tiene como eje la autenticación mediante clave y la comprobación de MACs permitidas.

Por último, en el apartado **Sumario**, se realiza una recapitulación de todo lo expuesto.

4.2 – La pila de protocolos BlueZ para Linux

4.2.1 – Definición y características

BlueZ [2] es la pila oficial de protocolos Bluetooth para Linux. Fue desarrollada inicialmente por Qualcomm y en la actualidad está liberada bajo Licencia Pública General GPL [3] (*GNU General Public License*), lo que significa que puede ser copiado, estudiado, modificado y redistribuido libremente.

BlueZ es parte del *kernel* oficial de Linux desde la versión 2.4 estando, por tanto, incluido en cualquier distribución moderna de Linux, es decir, no hará falta instalar nada para disponer de ella.

Algunas de las principales características de BlueZ son:

- Arquitectura flexible, eficiente y modular.
- Soporte para múltiples dispositivos BlueZ.
- Proceso multitarea de datos.
- Abstracción del hardware.
- Interfaz de *sockets* estándar para todas las capas.
- Soporte de seguridad a nivel PSM.
- Multiplataforma: x86 (simple y multiprocesador), SUN, SPARC, ARM, PowerPC, Motorolla, DragonBall.
- Funcionamiento en todas las distribuciones Linux: RedHat, Debian, Suse, etc.
- Gran cantidad de dispositivos soportados (PCMCIA, UART, USB).
- Soporte L2CAP, SDP, RFCOMM Y SCO.
- Disponibilidad de un emulador Bluetooth y dispositivos de configuración y prueba.
- Soporte para los siguientes perfiles de uso: GAP, DUN, LAN, SPP, PAN, Headset, OBEX (FTP), OBEX (OPP).
- Listas de correos participativas, con desarrolladores en todo el mundo contribuyendo al soporte y programación con BlueZ.

La principal desventaja de BlueZ es su falta de documentación. Esto se puede solucionar estudiando el código fuente directamente, lo que es muy costoso. Para empezar a programar con BlueZ existe una muy buena introducción en [4], muy útil

para los conceptos básicos. El código fuente se encuentra coloreado y bien clasificado, lo que facilita tareas como encontrar lo que se busca e ir de una parte a otra, en [5].

El uso de BlueZ en este proyecto frente a otras pilas mencionadas en el apartado 3.2.1, queda plenamente justificado debido a todas las funcionalidades que hemos comentado, destacando el hecho de ser software libre, ir incluido en el núcleo de Linux a partir de la versión 2.4 y el constante desarrollo de aplicaciones basadas en BlueZ.

4.2.2 – Paquetes de BlueZ

BlueZ se distribuye en un conjunto de paquetes, aunque el núcleo depende de la distribución del *kernel* Linux que estemos usando. Para versiones anteriores a la 2.4 que no incluyen la funcionalidad Bluetooth existen parches en la página web de Marcel Holtmann [6].

Además del soporte del núcleo, los paquetes que pueden usarse en función de las necesidades finales del usuario son:

- bluez-libs: Librerías necesarias para el desarrollo de aplicaciones y para el resto de paquetes BlueZ y aplicaciones que se enlacen dinámicamente a las librerías.
- bluez-utils: Aplicaciones de control para los dispositivos Bluetooth. Necesario para realizar *inquiry* o comunicaciones en general.
- bluez-sdp: Contiene las librerías, herramientas y el servidor SDP (sdpd) que conforman toda la funcionalidad SDP.
- bluez-pan: Programas, demonios y *scripts* necesarios para los perfiles DUN, LAN y BNEP-PAN.
- bluez-hcidump: Órdenes útiles para depurar y estudiar el funcionamiento general de los dispositivos Bluetooth usando HCI.
- bluez-hciemu: Contiene el emulador. Permite a los programadores probar su código sin un dispositivo Bluetooth real.
- bluez-bluefw: Contiene el firmware de varios dispositivos Bluetooth.

Todos estos paquetes se pueden descargar desde la página oficial de BlueZ en la sección de descargas [7].

4.2.3 – Herramientas de BlueZ

Como ya se ha comentado la documentación de esta pila de protocolos es inexistente y por tanto todo el aprendizaje del API se ha realizado a partir del propio código fuente. Afortunadamente, el núcleo de BlueZ viene acompañado por un conjunto de herramientas que permiten ejecutar las funciones Bluetooth implementadas en la pila de protocolos desde un *shell* o consola de órdenes. El primer paso para determinar cuáles son las funciones que nos interesan es el estudio de estas herramientas:

- `hciconfig`: configuración de dispositivos Bluetooth locales.
- `hcid`: demonio de la interfaz HCI.

- `hcitool`: gestión del enlace con otros dispositivos Bluetooth, detección de dispositivos remotos y resolución de nombres, entre otras funciones.
- `hcidump`: *sniffer* local del tráfico HCI, tanto entrante como saliente, por el dispositivo Bluetooth instalado en el sistema.
- `l2ping`: envío de solicitudes *echo request* (ping) a nivel L2CAP.
- `sdptool`: gestión de SDP, descubrimiento de servicios Bluetooth en dispositivos remotos.
- `sdpd`: demonio (*daemon*) del protocolo de descubrimiento de servicios SDP. Se encarga de proporcionar acceso a los servicios Bluetooth locales.
- `rfcomm`: gestión de conexiones rfcomm.
- `pand`: gestión de conexiones PAN (*Personal Area Network*).

A continuación se describe brevemente el funcionamiento de las más importantes.

4.2.3.1 - hciconfig

La herramienta que permite llevar a cabo todas las opciones de configuración del dispositivo Bluetooth local es `hciconfig`. Auque es utilizada fundamentalmente para activar y desactivar un dispositivo y para obtener o modificar todos los parámetros de funcionamiento, permite enviar cualquier orden Bluetooth al dispositivo indicado. Su sintaxis y opciones son:

```
hciconfig - HCI device configuration utility
Usage:
      hciconfig
      hciconfig   [-a] hciX   [command]
Commands:
      up                    Open and initialize HCI device
      down                  Close HCI device
      reset                 Reset HCI device
      rstat                 Reset statistic counters
      auth                  Enable Authentication
      noauth                Disable Authentication
      encrypt               Enable Encryption
      noencrypt             Disable Encryption
      piscan                Enable Page and Inquiry scan
      noscan                Disable scan
      iscan                 Enable Inquiry scan
      pscan                 Enable Page scan
      ptype       [type]    Get/Set default packet type
      lm          [mode]    Get/Set default link mode
      lp          [policy]  Get/Set default link policy
      name        [name]    Get/Set local name
      class       [class]   Get/Set class of device
      voice       [voice]   Get/Set voice setting
      inqparms    [win:int] Get/Set inquiry scan window and interval
```

```
      pageparms   [win:int]   Get/Set page scan window and interval
      pageto      [to]        Get/Set page timeout
      aclmtu      <mtu:pkt>   Set ACL MTU and number of packets
      scomtu      <mtu:pkt>   Set SCO MTU and number of packets
      features                Display device features
      version                 Display version information
      revision                Display revision information
```

Las órdenes más interesantes de esta instrucción son: `noscan`, `piscan`, `name` y `class`. Tendremos que analizar la implementación de estas órdenes para adaptarlas a las necesidades de nuestra aplicación. `Noscan` y `piscan` las utilizaremos para poner en modo invisible los dispositivos al iniciar el cliente, lo que mejorará el rendimiento al disminuir los dispositivos encontrados, como se demuestra en el capítulo 5. Por su parte, `name` y `class` se utilizarán para mostrar datos del dispositivo local tanto en el cliente como en el servidor.

4.2.3.2 - hcid

La aplicación `hcid` es el demonio que se encarga de gestionar los dispositivos Bluetooth. En el directorio `/etc/bluetooth` se encuentra el fichero `hci.conf` que permite definir los parámetros básicos de los dispositivos. Un ejemplo de este archivo es el siguiente:

```
# HCId options
options {
      # Automatically initialize new devices
      autoinit yes;

      # Security Manager mode
      #   none - Security manager disabled
      #   auto - Use local PIN for incoming connections
      #   user - Always ask user for a PIN
      security none;

      # Pairing mode
      #   none  - Pairing disabled
      #   multi - Allow pairing with already paired devices
      #   once  - Pair once and deny successive attempts
      pairing multi;

      # PIN helper
      pin_helper /bin/bluepincat;

      # D-Bus PIN helper
      #dbus_pin_helper;
}

# Default settings for HCI devices
device {
      # Local device name
      name "Luke Skywalker GRC";

      # Local device class
```

```
        class 0x100100;

        # Default packet type
        #pkt_type DH1,DM1,HV1;

        # Inquiry and Page scan
        iscan enable;
        pscan enable;

        # Default link mode
        #   none   - no specific policy
        #   accept - always accept incoming connections
        #   master - become master on incoming connections,
        #            deny role switch on outgoing connections
        lm accept;

        # Default link policy
        #   none    - no specific policy
        #   rswitch - allow role switch
        #   hold    - allow hold mode
        #   sniff   - allow sniff mode
        #   park    - allow park mode
        lp rswitch,hold,sniff,park;

        # Authentication and Encryption (Security Mode 3)
        auth disable;
        encrypt disable;
}
```

Al hacer cualquier cambio de configuración en este fichero se deben reiniciar los servicios Bluetooth para que los cambios tengan efecto. Esto se hace desde consola tal y como se muestra en la figura 4-1.

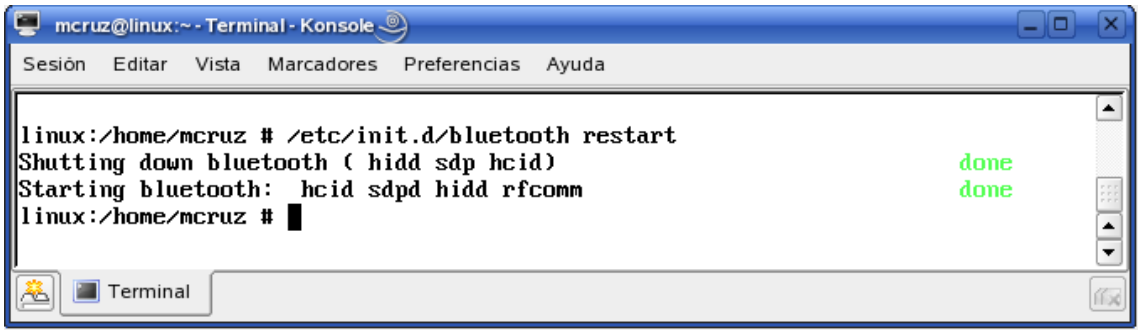


Figura 4-1: Reiniciar los servicios Bluetooth desde consola

4.2.3.3 – hcitool

La herramienta principal de la pila BlueZ es sin duda `hcitool`. Permite realizar operaciones relativas a la gestión del enlace con otros dispositivos Bluetooth, como buscar dispositivos cercanos, obtener información sobre un dispositivo remoto, crear una conexión y algunos otros. A continuación se muestra la ayuda en línea:

```
hcitool - HCI Tool
```

```
Usage:
      hcitool [options] <command> [command parameters]
Options:
      --help      Display help
      -i dev      HCI device
Commands:
      dev         Display local devices
      inq         Inquire remote devices
      scan        Scan for remote devices
      name        Get name from remote device
      info        Get information from remote device
      cmd         Submit arbitrary HCI commands
      con         Display active connections
      cc          Create connection to remote device
      dc          Disconnect from remote device
      sr          Switch master/slave role
      cpt         Change connection packet type
      rssi        Display connection RSSI
      lq          Display link quality
      tpl         Display transmit power level
      afh         Display AFH channel map
      lst         Set/display link supervision timeout
      auth        Request authentication
      enc         Set connection encryption
      key         Change connection link key
      clkoff      Read clock offset
      clock       Read local or remote clock
```

Las órdenes más interesantes de esta herramienta son: `inq`, `scan`, `dev`. Tanto `inq` como `scan` realizan un proceso de *inquiry* pero `scan` ofrece además la información del nombre del dispositivo, así como `dev` proporciona información de nuestro propio dispositivo local.

4.2.3.4 – sdptool

Otra aplicación que necesitaremos es `sdptool`, que nos permitirá realizar las funciones relacionadas con el registro y el descubrimiento de servicios.

La sintaxis y opciones son:

```
sdptool - SDP tool
Usage:
      sdptool [options] <command> [command parameters]
Options:
      -h          Display help
      -i          Specify source interface
Commands:
      search            Search for a service
```

```
browse            Browse all available services
records           Request all records
add               Add local service
del               Delete local service
get               Get local service
setattr           Set/Add attribute to a SDP record
setseq            Set/Add attribute sequence to a SDP record
```

Las órdenes más interesantes de esta herramienta son: `search`, `add`, `del`. La orden `search` es la que implementa el descubrimiento de servicios. Primero realiza un *inquiry* y después busca en el servidor SDP de cada dispositivo si está registrado el servicio en cuestión. `Add` realiza el registro de un servicio en el servidor SDP local y `del` lo elimina.

4.2.3.5 – hcidump

Es una herramienta que visualiza en pantalla todos los paquetes recibidos y enviados por un dispositivo Bluetooth específico. Es particularmente útil cuando se quiere analizar el funcionamiento de un dispositivo o depurar a bajo nivel posibles problemas de protocolos de comunicación, por lo que lo podremos utilizar en diversas fases de la implementación, ya sea para certificar el buen funcionamiento o, más frecuentemente, para localizar algún problema.

```
HCIDump - HCI packet analyzer ver 1.5
Usage: hcidump [OPTION...] [filter]
      -i, --device=hci_dev          HCI device
      -p, --psm=psm                 Default PSM
      -s, --snap-len=len            Snap len (in bytes)
      -r, --read-dump=file          Read dump from a file
      -w, --save-dump=file          Save dump to a file
      -a, --ascii                   Dump data in ascii
      -x, --hex                     Dump data in hex
      -R, --raw                     Raw mode
      -t, --ts                      Display time stamps
      -?, --help                    Give this help list
          --usage                   Give a short usage message
```

4.2.3.6 – pand

Con el demonio `pand` se pueden crear redes PAN conectando hasta 7 dispositivos Bluetooth de forma sencilla.

Como ya vimos en el capítulo 3, el perfil PAN especifica tres roles que pueden existir en una PAN (*Personal Area Network*): NAP (*Network Access Point*), GN (*Group ad hoc Network*) y PANU (*PAN User*). El NAP o el GN son los proveedores del servicio que coordinan el tráfico en la PAN. El NAP también es un punto de acceso a otra red. Los PANUs son los clientes o usuarios del servicio PAN.

Para establecer una PAN con BlueZ [8] primero hemos de cargar el módulo BNEP escribiendo la siguiente línea de órdenes (como superusuario):

```
#modprobe bnep
```

Recordemos que BNEP es el protocolo de encapsulamiento definido en el perfil PAN y es el que permite a las tramas Ethernet trabajar en enlaces Bluetooth, haciendo posible el acceso a redes Ethernet.

El demonio `pand` tiene muchas opciones y es usado tanto por los clientes como por los servidores. Estas opciones se muestran a continuación:

```
Bluetooth PAN daemon version 2.19
Usage:
      pand <options>
Options:
      --show --list -l              Show active PAN connections
      --listen -s                   Listen for PAN connections
      --connect -c <bdaddr>         Create PAN connection
      --autozap -z                  Disconnect automatically on exit
      --search -Q [duration]        Search and connect
      --kill -k <bdaddr>            Kill PAN connection
      --killall -K                  Kill all PAN connections
      --role -r <role>              Local PAN role (PANU, NAP, GN)
      --service -d <role>           Remote PAN service (PANU, NAP, GN)
      --ethernet -e <name>          Network interface name
      --device -i <bdaddr>          Source bdaddr
      --nosdp -D                    Disable SDP
      --auth -A                     Enable authentication
      --encrypt -E                  Enable encryption
      --secure -S                   Secure connection
      --master -M                   Become the master of a piconet
      --nodetach -n                 Do not become a daemon
      --persist -p[interval]        Persist mode
      --cache -C[valid]             Cache addresses
      --pidfile -P <pidfile>        Create PID file
```

Para cumplir nuestros requisitos hay que activar el demonio PAN en un nodo y configurarlo como NAP (GN) y como *master*. Esto se puede hacer en una sola línea de órdenes:

```
#pand -listen -master -role nap
```

Esto indica al demonio PAN escuchar cualquier conexión, efectuando cambio de rol a *master* y actuando como NAP, en este caso. Podemos cambiar `nap` por `gn` si preferimos que sea el GN quien coordine la PAN.

También se puede habilitar la conmutación *master/slave* en el fichero de configuración mencionado anteriormente `/etc/bluetooth/hcid.conf` poniendo en la línea correspondiente:

```
lm accept, master
```

Para establecer la conexión entre un PANU y un NAP (GN) hay que ejecutar en el nodo cliente:

```
#pand -connect <destination BT device address>
```

Si, por ejemplo, nuestro nodo configurado como NAP tiene la dirección 00:37:5C:68:A2:10, en nuestro cliente ejecutaríamos la orden:

```
#pand -connect 00:37:5C:68:A2:10
```

Cuando la conexión se ha establecido se crea en ambos dispositivos la interfaz virtual "bnep0". Este interfaz puede ser configurado como cualquier otro dispositivo de red utilizando el programa `ifconfig`.

Por ejemplo, en el NAP:

```
#ifconfig bnep0 10.0.0.1
```

Y en el PANU:

```
#ifconfig bnep0 10.0.0.2
```

Esto crea una red IP privada en el rango de direcciones 10.x.x.x. Ahora es posible utilizar el *ping* para comprobar la calidad del enlace.

Si añadimos otro cliente a la red, en el dispositivo que actúa como NAP aparecerá la interfaz "bnep1". Un a conexión con un tercer cliente añadiría la interfaz "bnep2" y así sucesivamente. El inconveniente es que cada una de estas interfaces ha de configurarse con una dirección IP distinta, con lo que nos podemos encontrar con hasta 7 redes distintas para un mismo adaptador físico Bluetooth. En la figura 4-2 se ilustra esta situación para dos clientes conectados. El PANU 1 y el PANU 2 no pueden comunicarse entre ellos, al no pertenecer al mismo rango de direcciones IP locales.

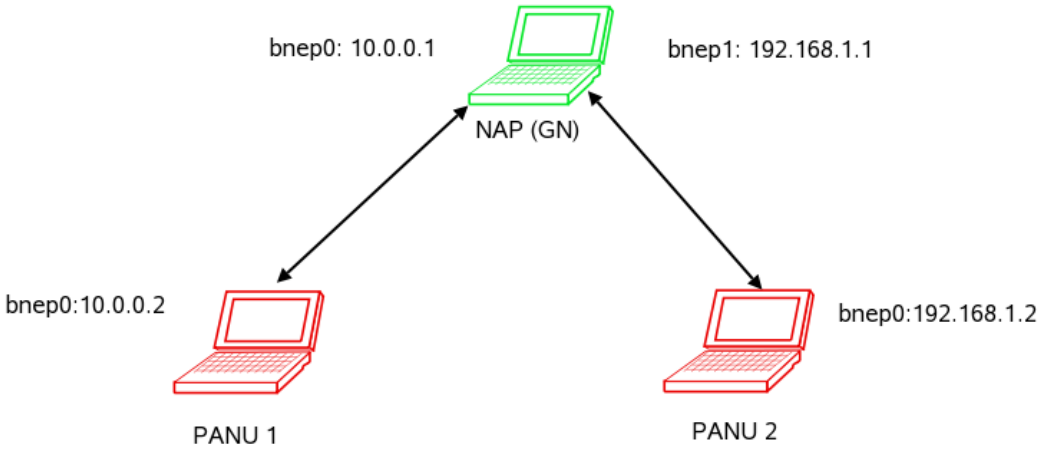


Figura 4-2: Red PAN con distintas direcciones IP

4.3 – Arquitectura del sistema

El servicio de integración automática de terminales Bluetooth en MANETs que se ha implementado se denomina **MANET_Gateway**. Un *gateway* o puerta de enlace proporciona a los dispositivos conectados a él (en nuestro caso, dispositivos Bluetooth) acceso hacia una red exterior (en nuestro caso, una MANET), realizando para ello operaciones de traducción de direcciones IP, es decir, NAT (*Network Address Translation*).

Los nodos de la MANET que actúan como servidores Bluetooth poseen una interfaz de red por la que se comunican con el resto de nodos de la MANET (normalmente Wi-Fi) y una interfaz Bluetooth por la que se conectarán los clientes que así lo deseen.

Aunque la implementación está diseñada para este escenario tipo, el servidor que actúa como *gateway* puede estar conectado a cualquier otro tipo de red, por lo que puede ofrecer a los clientes acceso a otros tipos de red, como Internet o a una LAN cableada local. Este hecho amplía las posibilidades y la funcionalidad del servicio MANET_Gateway, puesto que posibilita la integración automática de terminales Bluetooth en **cualquier red**, no sólo en MANETs.

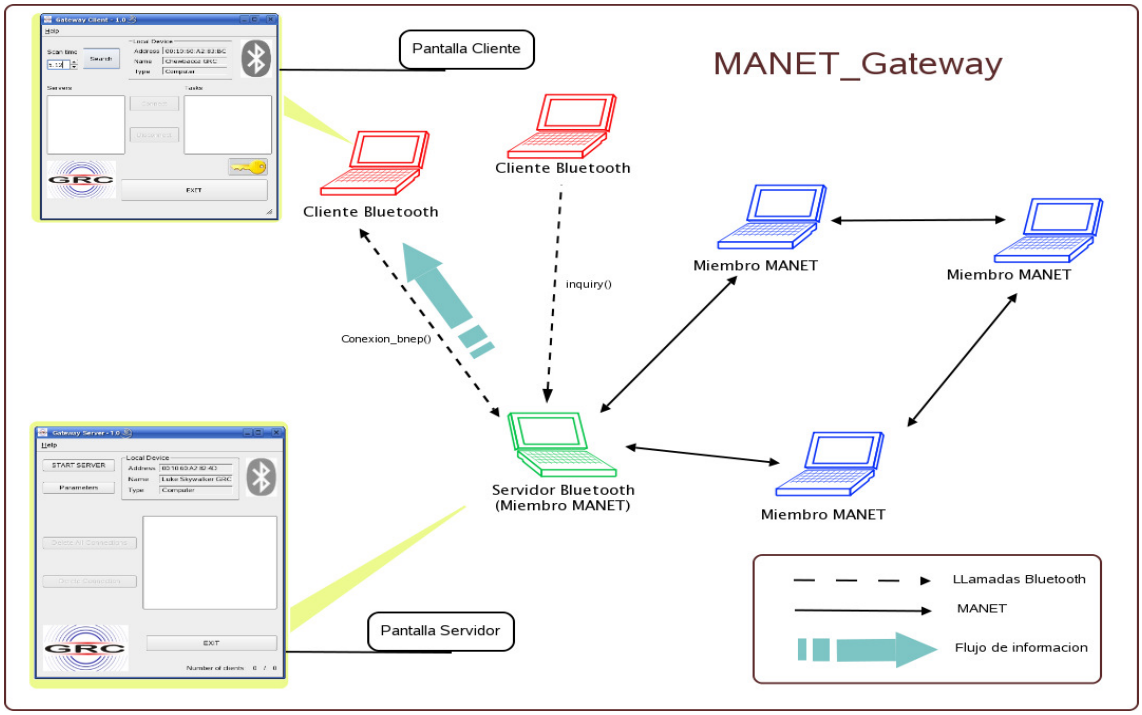


Figura 4-3: Arquitectura del sistema

En la figura 4-3 se muestra un esquema del sistema propuesto, donde es posible distinguir tres tipos de elementos: clientes Bluetooth esperando a ser configurados para participar en la MANET (en rojo), el servidor que ya forma parte de la red *ad hoc* y actuará de *gateway* para los clientes (en verde) y miembros participantes de la MANET (en azul). Los dos clientes Bluetooth representados en la figura están realizando distintas funciones: uno realiza el *inquiry* y otro establece la conexión con el servidor.

Para que el cliente Bluetooth se conecte a la MANET deberá buscar, entre todos los dispositivos Bluetooth, aquellos que ofrezcan el servicio MANET_Gateway. Para ello debe ejecutar primero la llamada *inquiry*, con lo que obtendrá una lista de dispositivos Bluetooth accesibles. A continuación buscará en cada dispositivo de esta lista el servicio MANET_Gateway. Posteriormente presentará al usuario un listado de servidores disponibles, permitiendo al usuario conectarse al que elija. Las tareas a realizar por el cliente se muestran en pantalla conforme se realizan y son las siguientes:

- Conectar al servidor.
- Obtener parámetros de configuración.
- Configurar la interfaz virtual de red "bnep0" (dirección IP/máscara de red).

Si todo ha ido bien el cliente indica al usuario que su acción ha tenido éxito o, en caso contrario, le enseñará un mensaje de error. El usuario tiene la posibilidad de detener su participación en la MANET en cualquier momento, reponiéndose la configuración a su estado original.

En cuanto al servidor, es visible por los demás dispositivos y registra el servicio con SDP. Estará escuchando conexiones entrantes por el puerto L2CAP 15 (el utilizado por el demonio `pand`) y permite que varios clientes estén conectados en paralelo, dando servicio hasta un máximo de 7 clientes simultáneos. En nuestra implementación, el servidor asigna direcciones IP consecutivas a cada cliente según éstos se vayan conectando. Para poder actuar de *gateway*, el servidor crea un *bridge* (puente) y realiza una serie de operaciones que se explican más adelante.

Además, se ha dotado al servicio de dos sistemas de seguridad opcionales. Estos sistemas consisten en realizar un filtrado de direcciones Bluetooth permitidas por el servidor y poder asociar una clave. Se pueden utilizar ambos sistemas, sólo uno de ellos o ninguno, según los parámetros de configuración del servidor.

El diagrama de la figura 4-4 representa la comunicación entre el cliente y el servidor. Todas las funciones reseñadas en el diagrama se pueden ver con más detalle en los puntos siguientes.

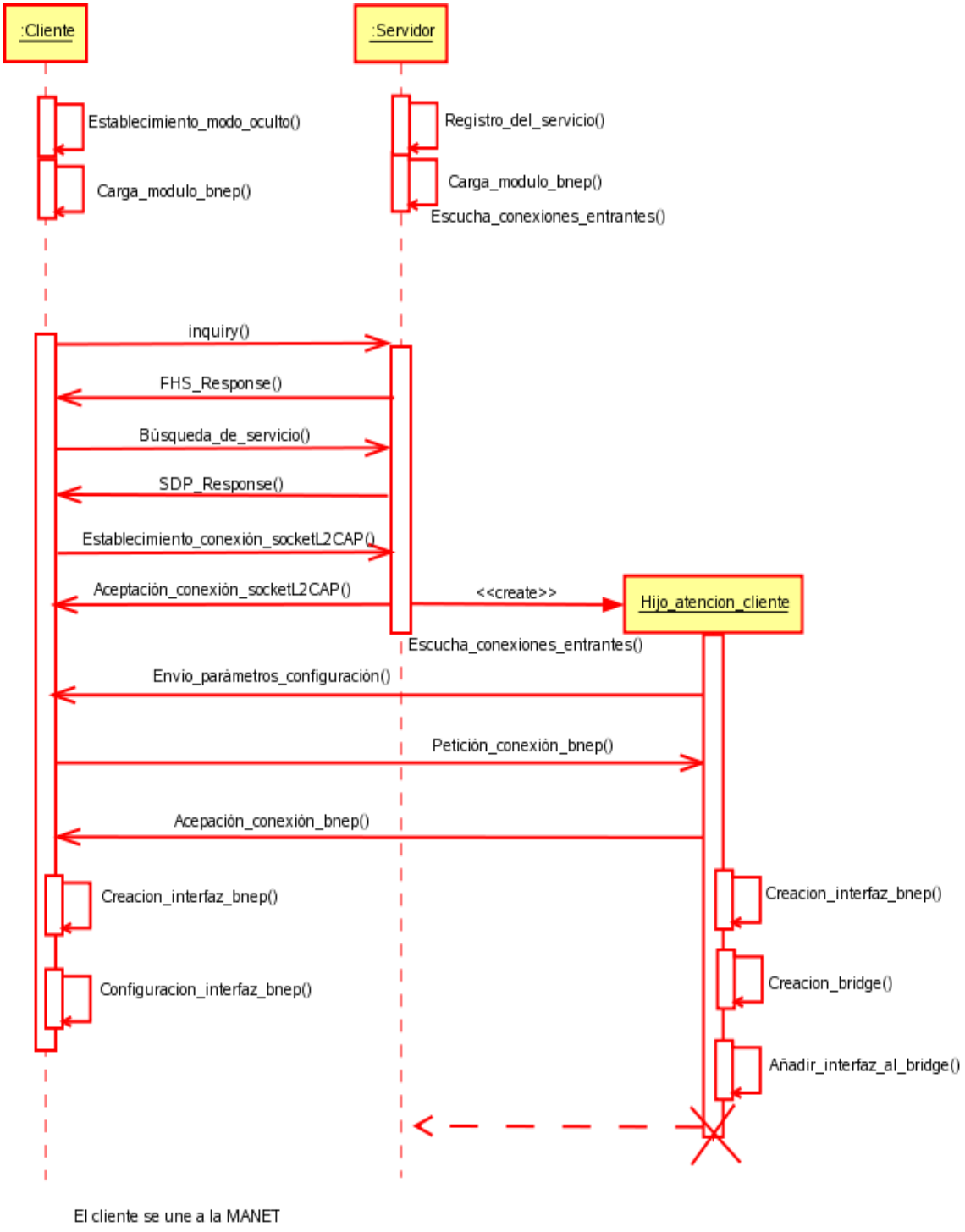


Figura 4-4: Diagrama de comunicación entre cliente y servidor

4.4 – Funcionamiento

En este apartado primero veremos por separado las funciones más importantes que implementan el cliente y el servidor, a fin de tener una idea más en detalle de lo que hace cada uno. En el apartado 4.4.3 se detalla el proceso de comunicación entre ambos y las acciones llevadas a cabo para la correcta configuración tanto del cliente como del servidor.

4.4.1 – Servidor

4.4.1.1 – Interfaz gráfica

En la figura 4-5 se muestra la ventana principal del servidor "Gateway Server 1.0", así como sus opciones.

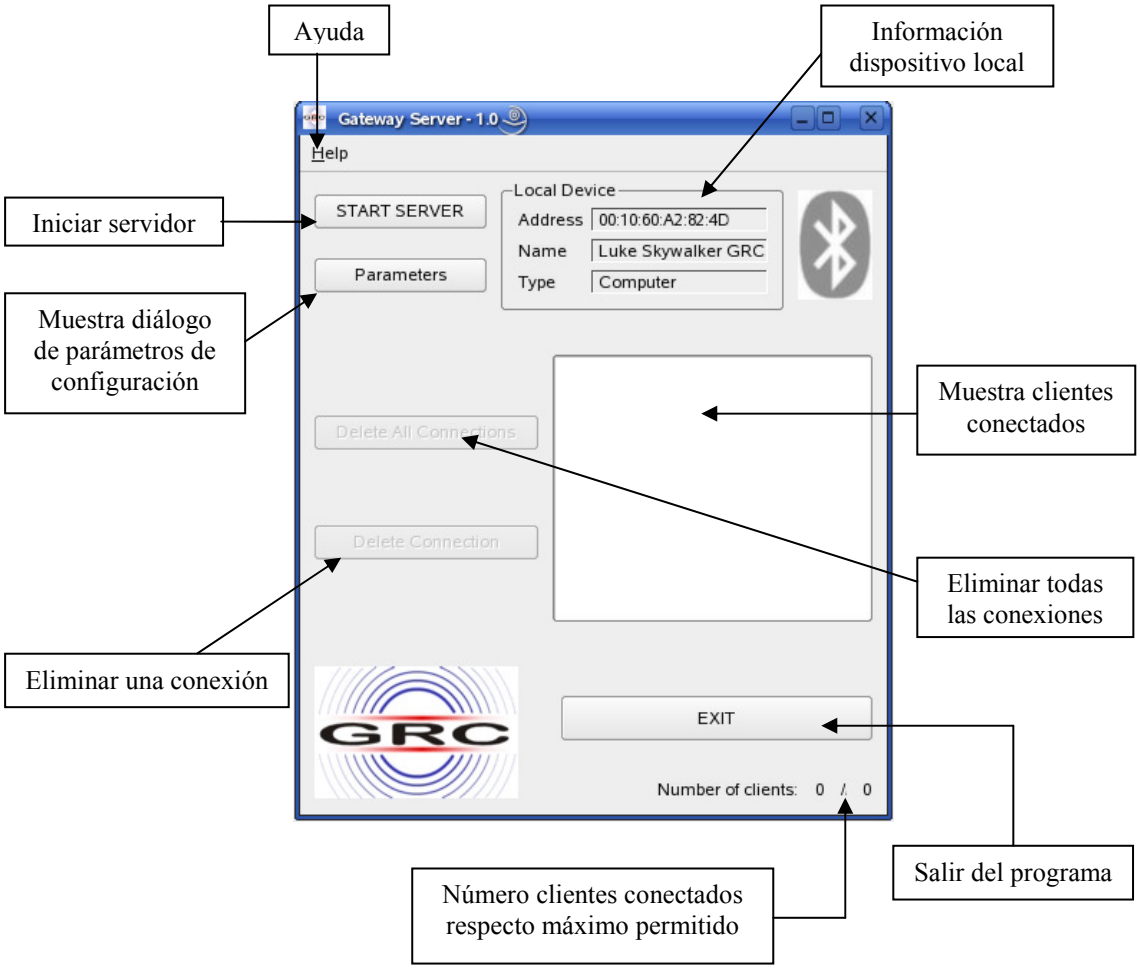


Figura 4-5: Interfaz gráfica del servidor

En la barra de menús sólo existe el menú "*Help*", que contiene dos entradas. La primera, "*About*", con información sobre el creador y la versión de la aplicación (ver figura 4-6). La segunda, "*About Qt*", con información sobre el programa Qt. Esta barra de menús se ha creado pensando en futuras ampliaciones y mejoras de la aplicación.

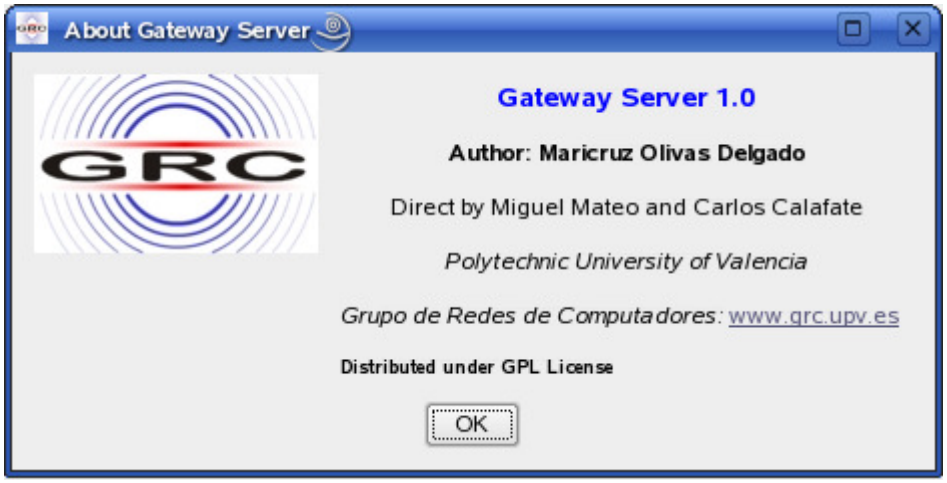


Figura 4-6: Diálogo "*About*" del servidor

Para poder arrancar el programa son necesarias las siguientes condiciones:

1) Quien ejecuta el programa es superusuario (*root*).

2) Existe un dispositivo Bluetooth local activo.

La condición de ser superusuario se establece porque se van a realizar acciones que requieren dicho privilegio, como el uso de `iptables`, de `ifconfig` o de la llamada `ioctl`. Por otra parte, que exista un dispositivo Bluetooth activo es indispensable para que nuestro programa funcione.

Nada más arrancar el servidor se realiza el registro del servicio MANET_Gateway.

En el cuadro "*Local Device*" se muestran algunas características del dispositivo Bluetooth local: la dirección BD_ADDR, el nombre amigable y el tipo.

Para iniciar el servidor se ha de pulsar sobre el botón "START SERVER", lo que hará que el servidor pase a escuchar conexiones entrantes y también que el nombre del botón cambien a "STOP SERVER", por lo que si lo pulsamos de nuevo se detiene la función de escucha y el servidor estará inactivo.

Mediante el botón "*Parameters*" se establecen los parámetros de configuración, como veremos en el apartado 4.4.1.4.

Una vez iniciado el servidor se observa en el cuadro de texto la lista de las direcciones Bluetooth de los clientes conectados con su correspondiente interfaz virtual BNEP asociada.

A la izquierda del cuadro de texto hay dos opciones: "*Delete All Connections*", que eliminara todas las conexiones existentes hasta el momento y "*Delete Connection*" que, previa selección del cliente requerido, elimina la conexión con el mismo.

En la esquina inferior derecha de la ventana, se muestra un contador de los clientes conectados frente al número máximo de clientes. Este valor máximo se ha establecido previamente. El contador se actualiza automáticamente con cada conexión o desconexión, así como el cuadro de texto que mostrará en tiempo real las conexiones establecidas en cada momento.

Por otra parte, el icono de Bluetooth que se muestra en la esquina superior derecha presentará color gris si no hay ningún cliente conectado y color azul en caso de que haya uno o más de uno conectado al servidor.

Por último, el botón de "*EXIT*" sale del programa si el usuario confirma que desea salir, con lo que se elimina el servicio del servidor SDP local. Al salir del programa también se eliminan las conexiones que se puedan encontrar activas en ese momento.

4.4.1.2 – Carga del módulo bnep y llamada ioctl

Como vimos en el apartado 4.2.3.6, para establecer una PAN es esencial cargar previamente el módulo BNEP. Esto se hace en la fase de inicialización del programa servidor, junto con el registro del servicio y la descripción del dispositivo local que se muestra en el cuadro "*Local Device*".

La función que lleva a cabo la carga de dicho módulo es `bnep_init()` y hace uso de la llamada al sistema `ioctl()` (abreviatura de *Input/Output Control*). Esta función permite enviar órdenes especiales y específicas a los dispositivos de entrada/salida y es de la forma:

```
int ioctl(int d, int request, void *arg)
```

El primer parámetro es un identificador de dispositivo. El segundo parámetro especifica la orden que se ha de enviar al dispositivo. Estas órdenes dependen del periférico con el que se esté tratando. El tercer parámetro es un puntero que apunta a la dirección inicial del parámetro o parámetros que se le han de pasar al controlador del dispositivo, o bien en donde ha de retornar éste los resultados.

La función `bnep_init()` obtiene primero un descriptor de *socket* de la siguiente forma:

```
ctl = socket(PF_BLUETOOTH, SOCK_RAW, BTPROTO_BNEP)
```

Donde `PF_BLUETOOTH` indica que el *socket* será de tipo Bluetooth, `SOCK_RAW` indica tipo de conector *raw* o directo y `BTPROTO_BNEP` establece el protocolo BNEP.

El descriptor `ctl` obtenido servirá de ahora en adelante como primer parámetro de llamadas `ioctl()` que se ejecutarán a lo largo del programa, por ejemplo cuando se muestren las conexiones establecidas mediante `bnep_show_connections()` o cuando queremos eliminar alguna.

Ahora pueden establecerse los parámetros BNEP haciendo uso de `ioctl()`:

```
ioctl(ctl, BNEPGETCONNLIST, &req)
```

Donde `req` es una estructura de tipo `bnep_connlist_req`, formada por diversos campos de información sobre dichos parámetros BNEP.

4.4.1.3 – Registro del servicio MANET_Gateway

Para registrar el servicio haremos uso de instrucciones del `sdpd` o demonio SDP de BlueZ. Registrar un servicio implica describir el servicio, conectar con `sdpd`, indicar a `sdpd` qué es lo que tiene que publicitar y desconectar.

Todo esto lo hacemos con la función `add_service()`, que comprende los siguientes pasos[1]:

4) Describir el servicio. En nuestro caso tendrá un `uuid` o identificador 0x1BFF, el nombre, MANET_Gateway, una breve descripción y el proveedor, en nuestro caso el Grupo de Redes de Computadores (GRC).

```
uint32_t service_uuid_int[] = { 0, 0, 0, 0x1BFF };
const char *service_name = "MANET_Gateway";
const char *service_dsc = "A Gateway for Bluetooth Devices in Ad
Hoc Network";
const char *service_prov = "GRC";
```

5) Asignar un *service record* (ver capítulo 3) donde se irá registrando toda la información del servicio. Esto se hace mediante la función `sdp_record_alloc()` que devuelve un puntero a estructura `sdp_record_t` (el *service record*).

```
sdp_record_t *record = sdp_record_alloc();
```

6) Establecer el identificador general del servicio.

```
sdp_uuid128_create( &svc_uuid, &service_uuid_int );
sdp_set_service_id( record, svc_uuid );
```

7) Publicar el *service record*, es decir, que se pueda encontrar públicamente. La función `sdp_list_append()` añade la información contenida por su segundo argumento al primer argumento de tipo `sdp_list_t` y devuelve asimismo una `sdp_list_t`.

```
sdp_uuid16_create(&root_uuid, PUBLIC_BROWSE_GROUP);
root_list = sdp_list_append(0, &root_uuid);
sdp_set_browse_groups( record, root_list );
```

8) Establecer la información L2CAP. La variable `psm` hace referencia al PSM (*Protocol Service Multiplexor*, ver capítulo 3) del protocolo BNEP, cuyo identificador es el 15.

```
sdp_uuid16_create(&l2cap_uuid, L2CAP_UUID);
proto_list[0] = sdp_list_append(0, &l2cap_uuid);
proto_list[0] = sdp_list_append(proto_list[0],
                        sdp_data_alloc(SDP_UINT16, &psm));
apseq_list = sdp_list_append(0, proto_list[0]);
```

9) Establecer la información BNEP. La variable `version` hace referencia a la versión del protocolo BNEP que se esté utilizando.

```
sdp_uuid16_create(&bnep_uuid, BNEP_UUID);
proto_list[1] = sdp_list_append(0, &bnep_uuid);
proto_list[1] = sdp_list_append(proto_list[1],
                        sdp_data_alloc(SDP_UINT16, &version));
```

10) Guardar en el *service record* la información L2CAP y BNEP almacenada en `proto_list`.

```
apseq_list   = sdp_list_append(apseq_list, proto_list[1]);
access_proto_list = sdp_list_append( 0, apseq_list);
sdp_set_access_protos( record, access_proto_list );
```

---

[1] Las variables de la forma `x_uuid` son de tipo `uuid_t` y las de la forma `x_list` pertenece al tipo `sdp_list_t`.

11) Establecer las características de NAP, que es el rol que ejercerá el servidor dentro de la red PAN.

```
sdp_uuid16_create(&pan_uuid, NAP_SVCLASS_ID);
svclass_list = sdp_list_append(0, &pan_uuid);
```

12) Guardar en el *service record* la información sobre NAP establecida. Como el perfil NAP es bien conocido, también guarda su `uuid` y la versión.

```
sdp_set_service_classes(record, svclass_list);
sdp_profile_desc_t profile;
sdp_uuid16_create(&profile.uuid, NAP_PROFILE_ID);
profile[0].version = 0x0100;
pfseq_list = sdp_list_append(NULL, &profile[0]);
sdp_set_profile_descs(record, pfseq_list);
```

13) Guardar en el *service record* el nombre, el proveedor y la descripción establecidos en el paso 1. Con esto se acaban de establecer todos los atributos necesarios.

```
sdp_set_info_attr(record, service_name, service_prov,
            service_dsc);
```

14) Conectar al servidor SDP local para registrar el *service record*. El argumento `BDADDR_LOCAL` provoca que la función `sdp_connect()` conecte con el servidor SDP local en vez de con uno remoto.

```
sdp_session_t session = sdp_connect( BDADDR_ANY, BDADDR_LOCAL,
SDP_RETRY_IF_BUSY );
sdp_record_register(session, record, SDP_RECORD_PERSIST);
```

15) Por último. desconectar del servidor SDP.

```
sdp_close(session);
```

4.4.1.4 – Establecimiento de los parámetros de configuración

Cuando se pulsa el botón "*Parameters*" nos encontramos con la ventana mostrada en la figura 4-7.

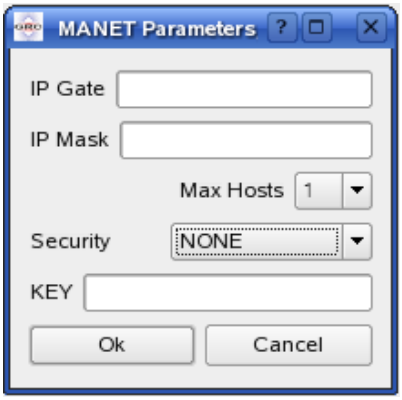


Figura 4-7: Ventana para la introducción de parámetros

En "*IP Gate*" se establece la dirección IP del servidor, es decir, es la dirección que posteriormente se configurará en la interfaz virtual de red PAN que crearemos, como se verá a continuación.

En "*IP Mask*" se ha de indicar la máscara de subred requerida para la red que vamos a crear.

El apartado "*Max Hosts*" es una lista desplegable en la que especificaremos el número máximo de clientes que atenderá el servidor. Como ya se dijo, este número no puede ser mayor que 7, que son los PANUs máximos que pueden haber conectados a un NAP o a un GN.

En el apartado "*Security*" nos aparece una lista desplegable con cuatro opciones de seguridad: "*NONE*", "*Allowed MACs*", "*PIN*" Y "*PIN+MAC*". Las tres últimas opciones se verán en la sección 4.5, "*NONE*" indica que no se implementa ningún mecanismo de seguridad.

El apartado "*KEY*" sólo se activará cuando se seleccione en el apartado anterior la opción "*PIN*" o la opción "*PIN+MAC*". Por seguridad, cuando se escriba en su campo aparecerán asteriscos y no los caracteres realmente escritos, para que nadie pueda ver la clave en pantalla.

Una vez introducidos los parámetros y después de haber pulsado el botón de "*Ok*" se guardan en un archivo de configuración llamado `conf.txt` que se almacenará en el mismo directorio donde se encuentre el ejecutable del programa. Este archivo no se destruye una vez finalizada la sesión, basta crearlo una vez (también se puede configurar abriendo con cualquier editor el propio archivo) si queremos utilizar siempre los mismos parámetros.

Si no se crea o no existe el archivo de configuración se establecen los siguientes parámetros por defecto:

- IP Gateway: 10.0.0.1
- IP Mask: 255.255.255.0
- Max. Num. Hosts: 5
- Security: NONE

4.4.1.5 – Escuchar conexiones entrantes

Cuando el usuario pulsa sobre el botón "*START SERVER*" se crea un proceso hijo que ejecutará la función `do_listen()` para atender las conexiones entrantes sin bloquear la interfaz gráfica del programa.

La función `do_listen()` crea un *socket* de tipo L2CAP, similar a un *socket* de red [9]. La estructura de dirección de *socket* L2CAP es la siguiente:

```
struct sockaddr_l2 {
    sa_family_t     l2_family;
    unsigned short  l2_psm;
    bdaddr_t        l2_bdaddr;
}
```

Veamos por pasos lo que hace la función `do_listen()`:

1) Crea un *socket* L2CAP y guarda su descriptor en la variable `soc_l2cap`.

```
int soc_l2cap = socket(AF_BLUETOOTH, SOCK_SEQPACKET,
                       BTPROTO_L2CAP);
```

2) Asocia con `bind()` el *socket* con el PSM correspondiente al protocolo BNEP, el BNEP_PSM (que, como ya se ha comentado, es el 15). El primer parámetro de esta llamada es el descriptor de *socket* abierto anteriormente, y el segundo contiene una variable de tipo `sockaddr_l2` con la información local. El puerto BNEP_PSM ha de establecerse con la ordenación adecuada mediante la función `htobs()`, que convierte datos *multibyte* al tipo de ordenación de *bytes* que utiliza Bluetooth (*Host to Bluetooth Short*).

```
struct sockaddr_l2 l2a;
l2a.l2_family = AF_BLUETOOTH;
l2a.l2_bdaddr=BDADDR_ANY;
l2a.l2_psm = htobs(BNEP_PSM);
bind(soc_l2cap, (struct sockaddr *) &l2a, sizeof(l2a));
```

3) Establece las opciones L2CAP de acuerdo con las especificaciones del protocolo BNEP.

```
struct l2cap_options l2o;
setsockopt(soc_l2cap, SOL_L2CAP, L2CAP_OPTIONS, &l2o,
            sizeof(l2o))
```

4) Establece el modo del enlace, en este caso *master*, puesto que estamos en el servidor que actuará como maestro mientras los clientes lo harán como esclavos.

```
int lm |= L2CAP_LM_MASTER;
setsockopt(soc_l2cap, SOL_L2CAP, L2CAP_LM, &lm, sizeof(lm))
```

5) Escucha las conexiones entrantes hasta un máximo de las especificadas por el parámetro "params.max_hosts" establecido en la configuración de parámetros.

```
listen(soc_l2cap,params.max_hosts)
```

6) Queda a la espera de aceptar conexiones entrantes. En la llamada `accept()` se obtiene un nuevo descriptor de fichero de *socket* `nsk` cuando se conecta un cliente, cuya información será volcada en la estructura de *socket* L2CAP `l2a`. Mediante este descriptor `nsk` se realizará la comunicación con el cliente (y para cada cliente se obtendrá un `nsk` distinto).

```
nsk = accept(soc_l2cap, (struct sockaddr *) &l2a, sizeof(l2a));
```

Cuando se acepta una conexión se crea un proceso hijo para atenderla, siguiendo el modelo clásico de programación de *sockets* [9], que es también el que utiliza `pand`. En el apartado 4.4.3 se explica lo que ocurre en este proceso hijo, encargado de establecer la conexión BNEP entre cliente y servidor.

4.4.1.6 – Eliminar y mostrar conexiones

Para eliminar conexiones tenemos dos posibilidades: eliminar una o eliminarlas todas. Para eliminar una conexión se utiliza `bnep_kill_connection()`, definida en BlueZ, a esta función le pasamos como argumento la dirección del dispositivo remoto cuya conexión queremos eliminar. Para eliminar todas las conexiones la función utilizada es `bnep_kill_all_connections()`, sin argumentos.

Las dos funciones anteriores utilizan la llamada al sistema `ioctl()`, pasándole como primer parámetro el descriptor de *socket* obtenido en la función `bnep_init()`. En `bnep_kill_connection()` la llamada es de esta forma:

```
ioctl(ctl, bnepconndel, &req)
```

En `bnep_kill_all_connections()` se utiliza esta misma llamada para eliminar cada interfaz BNEP de la lista encontrada al ejecutar:

```
ioctl(ctl, bnepgetconnlist, &req)
```

Esta última llamada es la misma que utiliza la función `bnep_show_connections()`, encargada de mostrar las conexiones que existen en cada momento. En su definición original muestra por pantalla la interfaz BNEP y la dirección Bluetooth asociada del dispositivo conectado, pero ha sido modificada ligeramente para que vuelque en una lista esta información para que sea mostrada en la interfaz gráfica, como se puede ver en la siguiente figura:

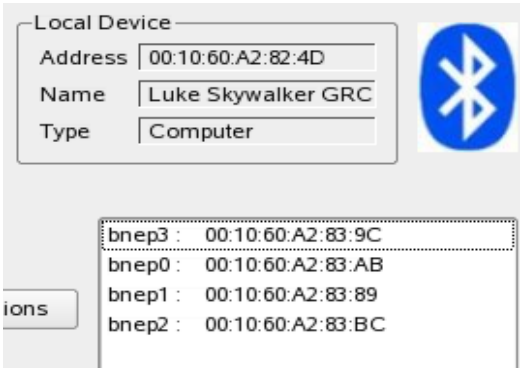


Figura 4-8: Clientes conectados al servidor MANET_Gateway

4.4.1.7 – Eliminar el servicio MANET_Gateway

Eliminar el servicio supone eliminarlo del registro del servidor SDP local, lo que se hace en el servidor cuando se sale del programa mediante la función `del_manet_service()`, que consta de los siguientes pasos:

1) Conectar con el servidor SDP local.

   ```
   sess = sdp_connect(BDADDR_ANY, BDADDR_LOCAL, SDP_RETRY_IF_BUSY);
   ```

2) Elimina el registro del servicio MANET_Gateway. El argumento `rec` corresponde al *service record* del servicio que se guardó cuando se efectuó el registro del mismo.

   ```
   sdp_record_unregister(sess, rec)
   ```

3) Cerrar la conexión con el servidor SDP local.

   ```
   sdp_close(sess)
   ```

4.4.2 – Cliente

Al igual que el servidor, el cliente también realiza la carga del módulo BNEP y puede eliminar la conexión, por lo que no se vuelven a explicar en este apartado dichas funciones.

4.4.2.1 – Interfaz gráfica

En la figura 4-9 se muestra la ventana principal del cliente “Gateway Client 1.0”, así como sus opciones.

La barra de menús es la misma que para el servidor, con las mismas opciones.

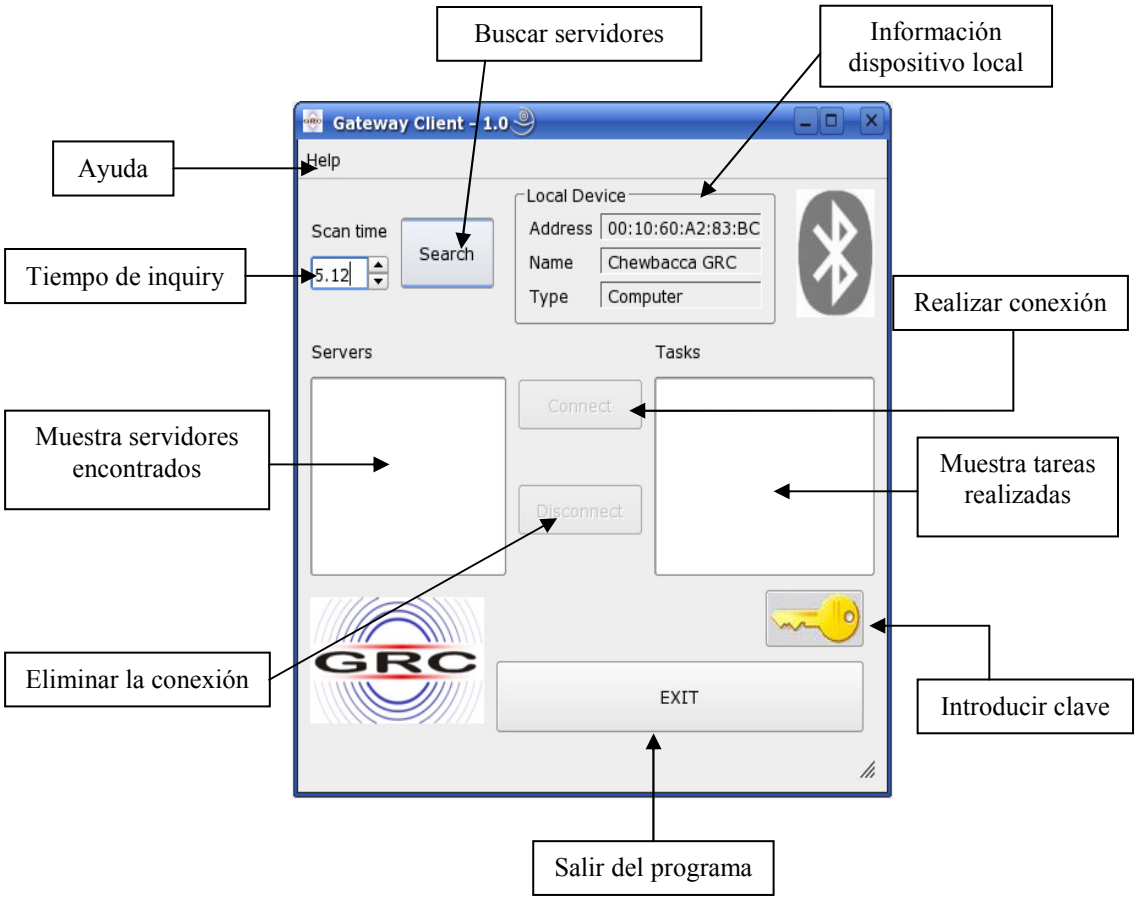


Figura 4-9: Interfaz gráfica del cliente

Mediante el botón "*Search*" el programa busca dispositivos Bluetooth cercanos que ofrezcan el servicio MANET_Gateway. En "*Scan time*" el usuario puede fijar el tiempo de realización del *inquiry*.

Los servidores encontrados se muestran en el cuadro "*Servers*" donde se indica su nombre amigable. En el caso de que no se encuentre ninguno se mostrará el mensaje "*No servers found*".

Si se ha encontrado algún servidor, el botón de "*Connect*" pasa a estado activo. Al presionarlo con el ratón, se procederá a la conexión con el servidor seleccionado por el usuario. En el cuadro "*Tasks*" se muestran las tareas realizadas de conexión y configuración. Si hay algún problema también se indicará aquí, por lo que el usuario puede saber si su conexión ha tenido éxito o no.

En caso que se establezca la conexión el botón "*Disconnect*" pasa a estado activo, pudiendo el usuario interrumpir la conexión cuando lo crea conveniente. Si la conexión se interrumpe por otra causa que no sea el accionamiento de este botón por parte del usuario (por ejemplo, el servidor ha desconectado o hay algún otro tipo de fallo en el enlace) esto también se indicará en el cuadro "*Tasks*" (ver figura 4-10).

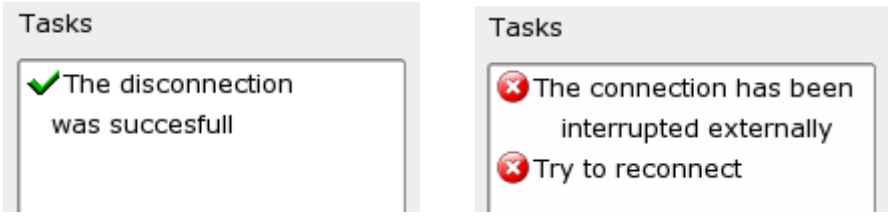


Figura 4-10: Detalle del cuadro de tareas cuando la conexión se interrumpe por el cliente (izquierda) y cuando es interrumpida externamente (derecha)

Por otra parte, el icono de Bluetooth que se muestra en la esquina superior derecha presentará color gris si el cliente no está conectado y pasará a azul cuando si lo esté.

Mediante el botón representado por una llave amarilla el usuario puede introducir la clave para poder conectarse a un servidor con seguridad. Esto lo veremos con más detalle en el apartado 4.5.

Por último, el botón de "*EXIT*" sale del programa si el usuario lo confirma. Esto no supondrá la pérdida de la conexión actual en caso que exista, por lo que el cliente permanecerá conectado.

4.4.2.2 – Descubrimiento del servicio MANET_Gateway

Para realizar el descubrimiento del servicio en realidad hemos de hacer dos operaciones: buscar dispositivos cercanos mediante el *inquiry* y buscar en cada dispositivo encontrado el servicio MANET_Gateway.

Estas dos operaciones se realizan mediante la función `manet_search()`. Los pasos que sigue esta función son los siguientes:

1) Establece un *socket* con el controlador del dispositivo Bluetooth local mediante la función `hci_open_dev()`. Esto se hace porque realizar operaciones Bluetooth a bajo nivel (como el *inquiry*) requiere enviar órdenes directamente al controlador a través de este *socket*. El argumento pasado a `hci_open_dev()` es el número del adaptador Bluetooth y se obtiene mediante `hci_get_route()`, a la que le pasamos el argumento NULL que hará que se obtenga el primer adaptador local Bluetooth disponible o, lo más normal, el único disponible.

```
int dev_id = hci_get_route(NULL);
int sock = hci_open_dev( dev_id );
```

2) Realiza un proceso de *inquiry* mediante `hci_inquiry()` y guarda en un puntero a la estructura `inquiry_info` la información relativa a cada dispositivo encontrado.

```
inquiry_info *ii;
hci_inquiry(dev_id, len, max_rsp, NULL, &ii, flags);
```

3) De la estructura `inquiry_info` obtenida nos interesa la dirección BD_ADDR de cada dispositivo Bluetooth, almacenada en el campo `bdaddr` de dicha estructura. Utilizando esta dirección se establece la conexión con el servidor SDP de dicho dispositivo mediante la función `sdp_connect()`.

```
for (int i=0; i < num_rsp; i++) {
      session = sdp_connect(BDADDR_ANY,&(ii+i)->bdaddr,
                            SDP_RETRY_IF_BUSY);
```

```
        //Los pasos 4 a 7 se sitúan dentro de este for
```

4) Una vez establecida la conexión con el servidor SDP remoto hay que buscar en su registro el servicio MANET_Gateway. De éste sabemos su `uuid`, que es lo que necesitamos para, mediante `sdp_service_search_req()`, realizar la búsqueda del servicio.

```
uint32_t svc_uuid_int[] = { 0x0, 0x0, 0x0, 0x1BFF };
uuid_t svc_uuid;
sdp_uuid128_create( &svc_uuid, &svc_uuid_int );
sdp_list_t *response_list = NULL, *search_list;
search_list = sdp_list_append( NULL, &svc_uuid );
int er = sdp_service_search_req(session,search_list,
                  SDP_ATTR_REQ_RANGE,&response_list);
```

5) La función `sdp_service_search_ req()` devuelve 0 si no ha habido ningún error, lo que no quiere decir que haya encontrado el servicio. Para saber esto también hemos de comprobar que la estructura de tipo `sdp_list_t` no esté vacía, es decir, asegurarnos de que haya encontrado el servicio.

```
if(!err && sdp_list_len(response_list)){
      //El servicio existe en el dispositivo dado
      //Los pasos 6 y 7 están dentro de este if
```

6) Ahora que ya hemos encontrado el dispositivo que nos ofrece el servicio, leemos su nombre amigable con `hci_read_remote_name()` y lo guardamos en un vector de caracteres.

```
char name_shown[248]
hci_read_remote_name(sock, &(ii+i)->bdaddr,
            sizeof(name_shown),name_shown, 0)
```

7) Guardamos tanto el nombre recién obtenido como la dirección del dispositivo en una lista, con el objetivo de mostrarla posteriormente en el cuadro "*Servers*" de la interfaz gráfica. Para añadir en la lista utilizamos la función `auto_insertar()`. Para convertir la dirección Bluetooth de formato `bdaddr_t` a cadena se utiliza la función `ba2str()`.

```
ba2str(&(ii+i)->bdaddr, addr);
auto_insertar(&lista, name_shown, addr);
```

8) Cerramos la conexión con el servidor sdp.

```
sdp_close(session);
```

9) Repetimos desde el paso 3 para cada dispositivo, hasta completar el bucle `for`.

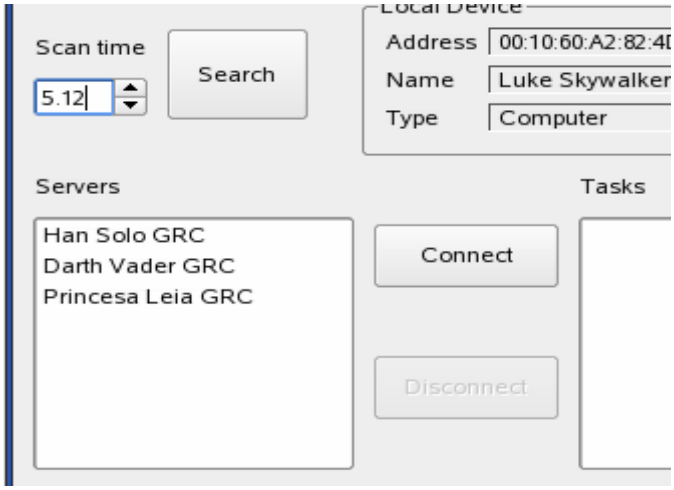


Figura 4-11: Detalle de los servidores MANET_Gateway encontrados

4.4.2.3 – Conectando con el *socket* L2CAP del servidor

Al pulsar el botón de "*Connect*" se ejecuta la función `do_connect()`, a la que se le pasa como parámetro la dirección del servidor seleccionado.

A continuación se comenta cómo se realiza la conexión con el *socket* L2CAP del servidor, a través del cual se intercambiará posteriormente la información necesaria para establecer la conexión PAN. Esto último, como ya se dijo, se explica con detalle en el siguiente apartado.

Estos son los pasos seguidos para establecer conexión vía *socket* L2CAP:

1) Crea un *socket* L2CAP y guarda su descriptor en la variable `sk`.

```
sk = socket(AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP)
```

2) Establece las opciones L2CAP de acuerdo con las especificaciones del protocolo BNEP.

```
struct l2cap_options l2o;
setsockopt(soc_l2cap, SOL_L2CAP, L2CAP_OPTIONS, &l2o,
            sizeof(l2o))
```

3) Conecta con el servidor a través del *socket*. Esto se realiza mediante la llamada al sistema `connect()`, cuyo primer argumento es el descriptor del *socket* creado anteriormente. En el segundo se especifican los parámetros de conexión al servidor. La dirección está contenida en la variable `baddr` (recordar que la habíamos guardado en una lista al realizar la búsqueda del servicio) que al ser de tipo puntero a cadena se "traduce" a `bdaddr_t` mediante la función `str2ba()`. El puerto es, por supuesto, el BNEP_PSM y la familia es de tipo Bluetooth.

```
struct sockaddr_l2 l2a;
l2a.l2_family = AF_BLUETOOTH;
l2a.l2_psm = htobs(BNEP_PSM);
str2ba( baddr, &l2a.l2_bdaddr );
connect(sk, (struct sockaddr *) &l2a, sizeof(l2a))
```

En el apartado siguiente vemos lo que ocurre después de establecido el *socket* L2CAP con el servidor.

4.4.3 – Comunicación entre cliente y servidor

En este apartado no se contemplan las opciones de seguridad, puesto que se explicarán en el apartado 4.5.

Una vez establecida la conexión mediante el *socket* L2CAP, tal y como se ha comentado en puntos anteriores, se procede a la realización de la conexión BNEP y la adhesión del cliente a la red PAN. A grandes rasgos, esto se produce de la siguiente manera:

1) El servidor envía a través del *socket* los parámetros de configuración del cliente: la puerta de enlace (`ip_gate`), la dirección IP (`ip_address`) y la máscara de subred (`ip_mask`).
2) Se establece la conexión BNEP.
3) Se crea la interfaz BNEP correspondiente en ambos dispositivos.
4) El cliente configura la nueva interfaz creada con los parámetros obtenidos.
5) El servidor realiza las acciones correspondientes para establecer el *gateway*.

Los dos últimos pasos se producen de forma simultánea en el cliente y en el servidor respectivamente.

El envío de parámetros se realiza mediante la transmisión de una única cadena utilizando la llamada `send()`, y se lee en el cliente mediante la llamada `recv()`.

Los pasos 2 y 3 se realizan mediante funciones ya implementadas en BlueZ y requieren una mayor explicación, así como la configuración del cliente y del servidor. Todo esto se detalla en los siguientes apartados.

4.4.3.1 – Establecimiento de la conexión y de la interfaz BNEP

Para establecer la conexión se utiliza la función `bnep_create_connection()` en el cliente y `bnep_accept_connection()` en el servidor. Sus prototipos son:

```
int bnep_create_connection(int sk,uint16_t role,uint16_t svc,char *dev)
```

Donde `sk` es el descriptor del socket L2CAP, `role` es el rol del dispositivo local (en este caso PANU), `svc` es el servicio remoto al que se quiere conectar (NAP, en este caso) y `dev` es el nombre del dispositivo de red que se obtendrá si la función ha tenido éxito (`bnep0`).

```
int bnep_accept_connection(int sk, uint16_t role, char *dev)
```

Donde `sk` vuelve a ser el descriptor de socket L2CAP, `role` es el rol del dispositivo local (NAP) y `dev`, como antes, es el nombre del dispositivo de red que será `bnep#`, siendo `#` el número de conexión.

Lo que hacen estas funciones es comunicarse a través del *socket* L2CAP, enviando y recibiendo una serie de parámetros en los que tanto el servidor como el cliente han de ponerse de acuerdo. Si este intercambio ha tenido éxito ambas funciones llaman a la función `bnep_connadd()`, que realiza a su vez una llamada `ioctl` para añadir la nueva conexión BNEP creada.

4.4.3.2 – Configuración de la interfaz BNEP en el cliente

Si todo ha ido bien, se crea una interfaz de red en el cliente denominada `bnep0`. Esta interfaz se configura como cualquier otra interfaz de red utilizando la orden `ifconfig`.

Para establecer la dirección IP:

```
ifconfig bnep0 ip_address
```

Para establecer la máscara de red:

```
ifconfig bnep0 netmask ip_mask
```

Ahora sólo nos queda establecer la puerta de enlace por defecto, que será la dirección IP del *bridge* creado por el servidor. Esto equivale a actualizar correctamente la tabla de encaminamiento, y se realiza mediante la herramienta `route` de Linux de la siguiente manera:

```
route add default gw ip_gate dev bnep0
```

Esta orden supone que por defecto se encaminen los paquetes con cualquier dirección destino utilizando la interfaz `bnep0` y a través de la puerta de enlace *ip_gate*.

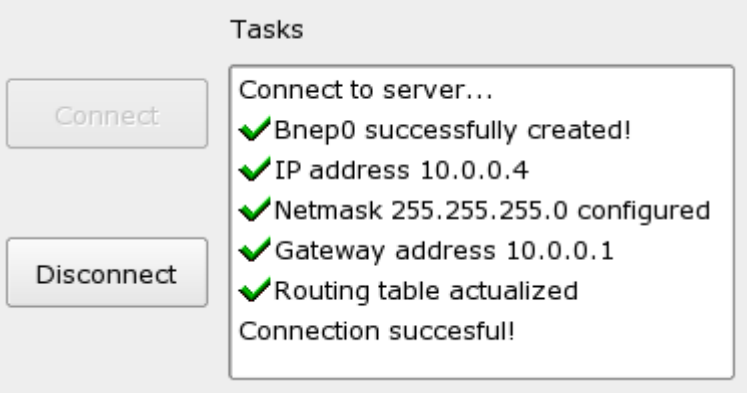


Figura 4-12: Cliente después de conectarse con éxito

4.4.3.3 – Establecimiento del *gateway*

Llegados a este punto nos encontramos con dos problemas:

1) Como se comentó en el apartado 4.2.3.6, por cada cliente que se añade al servidor se crea una interfaz de red virtual distinta que hay que configurar con su propia dirección IP y máscara de red (ver figura 4-2), por lo que se pueden llegar a crear hasta 7 subredes distintas sin que ninguno de los clientes pueda comunicarse entre ellos.
2) Ofrecer a los clientes acceso a la red a la que está conectado el servidor MANET_Gateway, ya sea una MANET o cualquier otro tipo de red.

Por tanto, para establecer las opciones propias de un *gateway*, hay que "aglutinar" todas las interfaces de red en una sola y dar salida a la red externa.

Para el primer problema la solución es la creación de un *bridge* o puente, que posibilita asociar varias interfaces a un mismo puente lógico. Básicamente la función que realiza un *bridge* es retransmitir de manera transparente el tráfico entre diversas interfaces de red. La utilidad `brctl` se encarga de esta tarea en los sistemas Linux. En nuestro caso,

crearemos un *bridge* denominado "pan0" y le asociaremos las interfaces `bnep#` que se vayan conectando.

Para el segundo problema la solución es hacer NAT (*Network Address Translation*) o traducción de direcciones IP. El NAT se suele usar para traducir direcciones IP privadas de una red interna a una única dirección IP pública al enviar, realizando el proceso inverso al recibir. En este proceso quedan "enmascaradas" las direcciones IP privadas, técnica denominada *IP Masquerading*. En nuestro caso la dirección IP que queremos enmascarar es la del *bridge*, dirección que se traducirá a aquella que posea el interfaz de red de nuestro servidor conectado a la red externa a la PAN.

Linux dispone de la herramienta `iptables`, que permite configurar las reglas del sistema de filtrado de paquetes IP (*NetFilter*), para poder conformar las opciones de NAT, así como otras opciones de seguridad que veremos posteriormente. Se supone que se trabaja con un núcleo Linux de versión 2.4 o superior, puesto que para la versión 2.2 la herramienta que realizaba estas funciones es `ipchains`.

La topología de red resultante se puede observar en la figura 4-13, donde "RED" es la red externa genérica e "[Interfaz_red]" la interfaz de red del servidor que tiene conexión con esta red externa. Con letras azules se resaltan las soluciones adoptadas para la resolución de los problemas comentados. En esta topología los clientes pueden comunicarse entre ellos y acceder a la red externa a través del servidor.

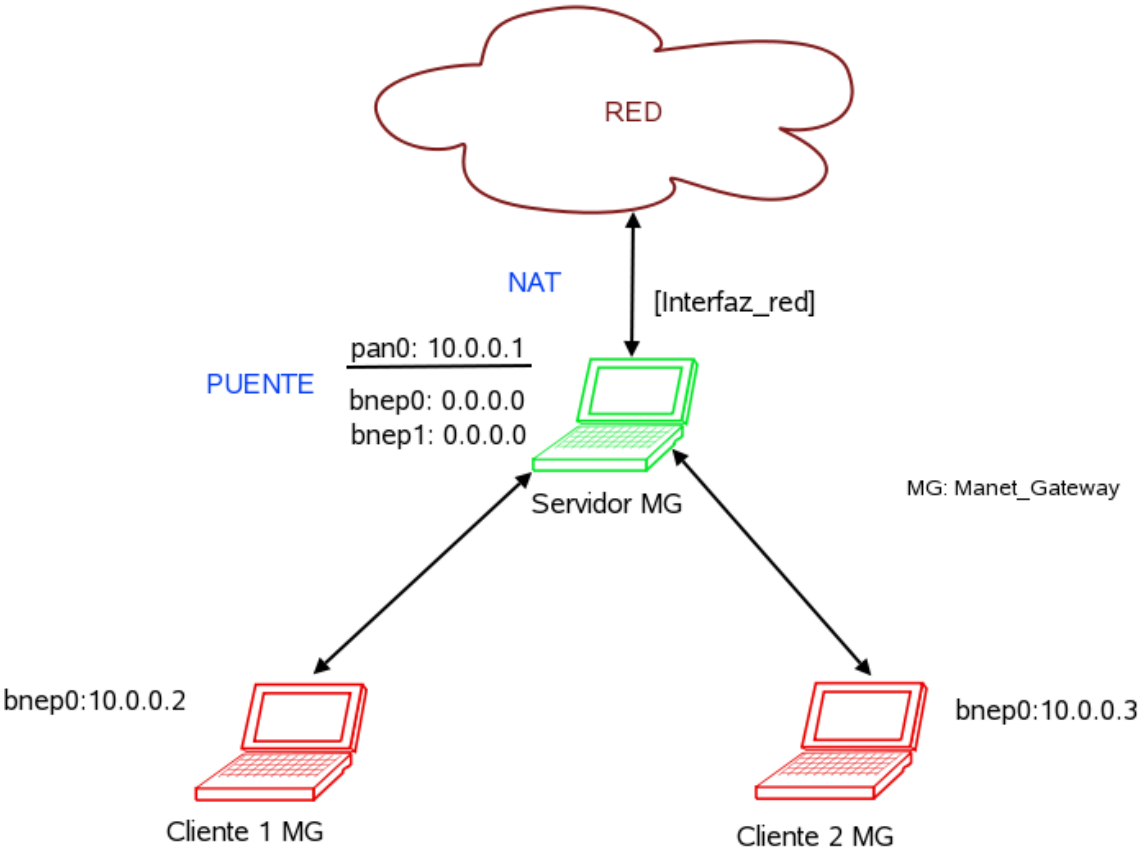


Figura 4-13: Topología de red con puente y NAT

Las acciones llevadas a cabo en nuestro programa para el establecimiento del *bridge* son las siguientes:

1) Crear el puente, que será una instancia lógica que llamaremos `pan0`. Esta instancia es el contenedor de las interfaces que tomarán parte en el *bridging*.

   ```
   brctl addbr pan0
   ```

2) Configurar el puente. La instancia `pan0` aparecerá como una más de las interfaces de red que posea el dispositivo. Hay que darle la dirección IP y la máscara de subred que se ha establecido en los parámetros (apartado 4.4.1.4), mediante la herramienta `ifconfig`:

   ```
   ifconfig pan0 ip_gate
   ifconfig pan0 netmask ip_mask
   ```

3) Dar de alta a la interfaz, puesto que de lo contrario aparecerá como inactiva:

   ```
   ifconfig pan0 up
   ```

4) Añadir al *bridge* la interfaz BNEP que se ha creado al establecer la conexión con el cliente, siendo # el número de la interfaz dada:

   ```
   brctl addif pan0 bnep#
   ```

5) Configurar dicha interfaz con la dirección IP nula o por defecto. A partir de ahora cualquier paquete que llegue a través de esta interfaz se retransmitirá de manera transparente a la instancia lógica `pan0`.

   ```
   ifconfig bnep# 0.0.0.0
   ```

Por su parte, para establecer las opciones de NAT las acciones realizadas son:

1) Primero, habilitar el *forwarding* o reenvío de paquetes TCP/IP. Si no lo hacemos, los paquetes que lleguen a `pan0` y se dirijan a la red externa, serán descartados. Para hacer esto hay que darle el valor 1 a la variable del sistema "net.ipv4.ip_forward":

   ```
   echo ""1"" > /proc/sys/net/ipv4/ip_forward
   ```

2) Activar el NAT propiamente dicho, mediante una regla que indica a NetFilter que debe hacer *IP Masquerading* (enmascaramiento) con los paquetes que se envíen hacia la red externa.

   ```
   iptables –t nat –A POSTROUTING –o interface –j MASQUERADE
   ```

   Aquí *interface* indica la interfaz de red que encamina por defecto, es decir, aquella que tiene acceso a la red externa. Esto se puede ver con la utilidad `route` de Linux, que muestra la tabla de encaminamiento de una máquina. Por ejemplo:

   ```
   # route –e
   ```

```
Destination  Gateway      Genmask        Flags  MSS  Window  irtt  Iface
192.168.1.0  *            255.255.255.0  U        0  0          0  ra0
loopback     *            255.0.0.0      U        0  0          0  lo
default      192.168.1.1  0.0.0.0        UG       0  0          0  ra0
```

   En este caso *interface* es `ra0`.

Por último, para que la configuración de *forwarding* que acabamos de realizar no se convierta en un potencial agujero de seguridad, resulta conveniente asegurar mediante reglas dicho *forwarding*. De nuevo usamos `iptables` para ello:

```
iptables –A FORWARD –i pan0 –j ACCEPT
iptables –A FORWARD –m state --state ESTABLISHED,RELATED –j ACCEPT
iptables –A FORWARD –j DROP
```

La primera regla acepta cualquier conexión que proceda de la red PAN, la segunda acepta todo el tráfico de conexiones ya establecidas (ESTABLISHED) relacionado con otras conexiones (RELATED) y la tercera evita cualquier otro tipo de conexiones.

Al conectarse el primer cliente con el servidor, se realizan las acciones pertinentes para la creación del puente y el establecimiento de las reglas en `iptables`. Para los siguientes clientes conectados estas acciones ya no se vuelven a realizar, por lo que únicamente se añade la nueva interfaz al puente y se configura con dirección IP nula.

4.5 – Seguridad

4.5.1 – Estudio previo

Al plantearse la opción de establecer algún mecanismo de seguridad, los objetivos que se perseguían eran la integración en el programa y su fácil configuración y funcionamiento.

Puesto que el protocolo de comunicaciones utilizado es Bluetooth, primero se estudiaron los distintos mecanismos de seguridad establecidos en dicho estándar, ya descritos en el apartado 3.4, donde se consignaron los 3 modos de seguridad en Bluetooth: sin seguridad, a nivel L2CAP y a nivel LM.

El siguiente paso fue determinar cómo se implementaban estos modos en BlueZ. A priori el modo más seguro es el Modo 3, dado que no permite ningún tipo de conexión hasta que no se establezca el intercambio de número PIN y todo el tráfico posterior es cifrado.

Configurar un dispositivo para que adopte este modo de seguridad requiere modificar el archivo `hcid.conf` de la siguiente manera:

```
# Security Manager mode
      security user; # o security auto;
 # Authentication and Encryption (Security Mode 3)
      auth enable;
      encrypt enable;
```

Si el campo `security` se asigna a `user` siempre se le preguntará al usuario por el número PIN y, si se pone `auto`, siempre se usará el PIN establecido en un archivo local (normalmente en `/etc/bluetooth/pin`).

Para el intercambio del PIN, BlueZ utiliza un programa que denomina "*PIN helper*", es decir, administrador de PIN. De hecho, no hay un único programa administrador de PIN, por lo que se puede tener instalado el kbluepin, el bluepincat, el bluepin, el pin_helper… El administrador de PIN también se indica en el fichero `hcid.conf`:

```
# PIN helper
      pin_helper /bin/bluepincat;
```

Así pues, para establecer una conexión segura con otro dispositivo Bluetooth, sólo hay que configurar el archivo `hcid.conf`. Durante el emparejamiento de los dispositivos se realizará el proceso de autenticación con ayuda del "*PIN helper*" correspondiente y se generará la clave de cifrado a nivel de enlace. Si todo ha ido bien, el dispositivo autenticado pasa a ser de confianza, por lo que no se le vuelve a demandar el código PIN en sucesivas conexiones, utilizándose para todas ellas la clave generada a nivel de enlace. Los inconvenientes encontrados para este Modo 3 son los siguientes:

- La configuración de seguridad obliga a que todas las conexiones tengan que ser seguras. Esto obliga al dispositivo remoto a ser también seguro y provoca una situación un tanto anómala: si el dispositivo al que queremos conectarnos es no seguro (es decir, a priori no opone ninguna "resistencia" a que utilicemos sus servicios) nuestro dispositivo local exigirá que la conexión sea segura y, al no darse esta condición, suspenderá la comunicación.
- No se encontró ningún mecanismo de autorización adicional, por lo que un dispositivo remoto autenticado tiene acceso a todos los servicios, sin que podamos indicar restricciones adicionales a cada uno de ellos.
- El hecho de que sólo se realice la autenticación en una ocasión puede ser un fallo importante de seguridad si alguien con malas intenciones se hace con el dispositivo autenticado.
- Establecer el Modo 3 es manual, es decir, requiere que el usuario realice una serie de acciones, con lo que contraviene el propósito de facilidad de uso que se había establecido.
- Intentar integrar las acciones que tendría que realizar el usuario en nuestro código resultó enormemente complejo por la funciones implicadas y porque BlueZ, como ya se ha comentado, hace uso de un administrador de PIN. Escoger el adecuado es imposible puesto que en cada dispositivo servidor puede haber uno distinto instalado, con lo que el código podría fallar.

Por todos estos inconvenientes se pasó a estudiar la posibilidad de implementar el Modo 2 de Bluetooth, que posibilita seguridad a nivel L2CAP y la autorización a los servicios. Si se observan las opciones del demonio PAN comentadas en 4.2.3.6 se ve que hay tres opciones de seguridad, que se pueden implementar en el NAP o en el GN:

```
--auth -A                    Enable authentication
--encrypt -E                 Enable encryption
--secure -S                  Secure connection
```

Cualquiera de las tres opciones implica que el dispositivo remoto que quiera conectarse al NAP o al GN se tenga que autenticar. Esto supone tener habilitadas las opciones de seguridad en el archivo `hcid.conf`. Una vez el dispositivo se autentica queda autorizado para utilizar el servicio en ocasiones futuras. Esto no quiere decir que pueda utilizar el resto de servicios, puesto que queda autorizado exclusivamente para el servicio de NAP (GN).

A nivel de código fuente, para establecer estas opciones de seguridad se hace lo siguiente:

```
lm |= L2CAP_LM_ENCRYPT; //si se ha escogido la opción -E
lm |= L2CAP_LM_AUTH; //si se ha escogido la opción -A
```

```
lm |= L2CAP_LM_SECURE; //si se ha escogido la opción –S
```

Una vez indicada la opción o las opciones se llama a la función `setsockopt()` (ver el paso 4 del apartado 4.4.1.5)

Así pues no parecía muy complejo realizar los cambios adecuados para establecer el Modo 2 de seguridad en nuestro servidor. Sin embargo, configurar alguna de estas opciones de seguridad en el cliente no estaba contemplado en el demonio `pand` y tampoco se pudo conseguir implementarlas en nuestro código.

En resumen, el principal inconveniente en este caso es que, aún pudiendo establecer el Modo 2 para nuestro servidor, el cliente tiene que configurarse en Modo 3, con lo que se vuelven a tener los problemas comentados para dicho modo. Además, el mecanismo de autorización sigue siendo automático y no permite interacción con el usuario.

Exploradas estas posibilidades quedó clara la dificultad de establecer un modo de seguridad Bluetooth "puro", pero también se determinaron las características que queríamos para nuestro sistema de seguridad:

- Establecer algún mecanismo de autenticación "no-Bluetooth", más concretamente, establecer una clave a nivel de servicio.
- Esta autenticación es pues a nivel de nuestro servicio, por lo que no implica que se puedan utilizar otros servicios ofertados.
- Realizar un "filtrado de MACs", es decir, mantener una lista de direcciones Bluetooth de dispositivos autorizados a utilizar el servicio.
- Poder establecer varios niveles de seguridad en nuestro servidor: sin seguridad, sólo filtrado de MACs, sólo autenticación o ambas.
- En el cliente, posibilidad de introducir la clave de conexión con el servicio y modificar ésta de manera dinámica cuando se desee.

Obsérvese que, si no se produce la autenticación y, por tanto, el emparejamiento de dispositivos tal y como se realiza en Bluetooth, tampoco se genera la clave de enlace necesaria para realizar el cifrado de los paquetes que se intercambien entre los dispositivos. Nuestro mecanismo no contempla el cifrado porque cifrar supone ralentizar el proceso de transmisión de datos, lo que disminuye la eficiencia. Esto no interesa en el escenario en el que se sitúa nuestro servicio, donde se supone que se generará gran cantidad de tráfico tanto de los nodos clientes hacia la MANET como en sentido contrario y lo importante es que la transmisión de dichos datos sea lo más rápida posible.

En los siguientes apartados se describe detalladamente la solución desarrollada y cómo se consiguieron los objetivos propuestos.

4.5.2 – Solución desarrollada: "Pseudo Modo 2"

Como solución para el problema de autenticación se propone la utilización del algoritmo de cifrado RSA para el intercambio de la clave entre el cliente y el servidor. La autorización se realiza comprobando la dirección Bluetooth de los clientes en un archivo de MACs autorizadas. Se ha dado en llamar a esta solución "Pseudo Modo 2"

puesto que posee semejanzas y diferencias con el Modo 2 "puro" de Bluetooth. Las semejanzas son:

- Establece seguridad a nivel L2CAP, una vez constituida la comunicación a este nivel.
- Ofrece seguridad en el servicio en concreto, no en todo el dispositivo como el Modo 3.
- Mecanismo de autorización.

Las diferencias son las siguientes:

- No se cifra el tráfico Bluetooth punto-a-punto, sólo el de intercambio de la clave.
- No basta con que un dispositivo esté autorizado, además se establece un mecanismo de autenticación mediante validación de clave.
- El "gestor de seguridad" es el propio usuario del servidor MANET_Gateway, no ningún otro mecanismo ajeno o automatizado.

Además de estas diferencias hay que señalar el hecho de que este "Pseudo Modo 2" sólo afectará, naturalmente, a nuestro servicio y no al resto.

Más importante aún es el hecho de que el "Pseudo Modo 2" es totalmente compatible con el Modo 3 de seguridad Bluetooth, que ya se ha explicado que ha de ser configurado manualmente por el usuario pero que posibilitaría un mayor nivel de seguridad. Es decir, podemos tener habilitado el Modo 3 y seguir usando el esquema de seguridad propuesto en nuestro servicio.

Ciñéndonos a nuestra implementación, se establecen tres niveles de seguridad, siendo el último el más seguro:

- Nivel 0: no hay seguridad.
- Nivel 1: filtrado de MACs (autorización).
- Nivel 2: validación de la clave (autenticación).
- Nivel 3: filtrado de MACs y validación de la clave (autenticación y autorización).

Antes de enviar los parámetros de configuración al cliente (es decir, una vez establecido el *socket* L2CAP entre servidor y cliente) se realizan las funciones correspondientes de seguridad. Si alguna de las comprobaciones falla, se cerrará la conexión L2CAP y el cliente no podrá conectarse.

Como ya se ha comentado, el mecanismo de autorización es gestionado por el propio usuario del servidor, que establece en el archivo denominado `autorized_macs.txt` las direcciones Bluetooth que quiere autorizar. Una vez conectado el cliente al *socket* L2CAP, el servidor comprobará mediante la función `check_MAC()` que la dirección del cliente es una de las autorizadas en el fichero. Si no es así, se cierra la conexión y el cliente no podrá formar parte de la red Bluetooth.

El mecanismo de autenticación es más complejo, al introducir el concepto de RSA y requerir que se ejecuten más funciones tanto en el servidor como en el cliente. En el apartado siguiente se explica con detalle.

4.5.2.1 – Algoritmo de cifrado RSA y funciones

El algoritmo de cifrado RSA [10] es el más extendido de los algoritmos de clave pública o asimétrica [11], considerándose un estándar *de facto*.

El algoritmo RSA se basa en el intercambio de dos claves: una privada y una pública. La clave pública es conocida por todo el mundo pero la clave privada sólo pertenece a su propietario. De esta forma, un emisor puede cifrar un mensaje con la clave pública pero únicamente el receptor puede descifrarlo mediante la clave privada que sólo posee él.

Sin entrar en fundamentos matemáticos, diremos que las claves son pares de números formados por un exponente "e" y un módulo "n" que es el producto de dos grandes números primos (mayores que 10^{100}) elegidos al azar. La seguridad del algoritmo radica en el problema conocido en matemáticas como el de factorización de números grandes por sus números primos. Con las técnicas de computación actuales es imposible romper el algoritmo (se prevé que las computadoras cuánticas puedan hacerlo, pero de momento está por demostrar y no afectan para nada al desarrollo de este proyecto).

Volviendo a nuestro sistema, supongamos que el usuario ha configurado el servidor MANET_Gateway con un nivel de seguridad 2 ó 3 y, por lo tanto, ha dispuesto una clave que ha de ser la misma que envíe el receptor con objeto de autenticarse. Si el cliente enviara la clave "en claro" (sin cifrar) cualquier dispositivo malicioso no autorizado podría interceptar mediante un rastreador de paquetes Bluetooth dicha clave y conectarse al servicio. Para que esto no ocurra entra en juego el algoritmo RSA.

Las claves pública y privada se generan al configurar los parámetros en el MANET_Gateway Server, acción que se realiza cada vez que iniciamos una nueva sesión en el servidor o cuando se establecen los parámetros en el cuadro de diálogo correspondiente. La función `RSA_write_files()` es la encargada de generar y escribir las claves, siguiendo estos pasos:

1) Genera una clave RSA, con un tamaño de módulo de 2048 bits (por debajo de 1024 el algoritmo se considera inseguro) y de exponente 65537; con lo que nos aseguramos una clave de gran longitud. Hay que recordar que, cuanto mayor sea la longitud de la clave, tanto más seguro es el algoritmo.

```
RSA *key = RSA_generate_key(2048, 65537, NULL, NULL);
```

2) Genera el archivo `pub.key` (que se guardará en el mismo directorio que el ejecutable) con la clave pública mediante la función `PEM_write_RSAPublicKey()`, a la que se le pasan como parámetros el fichero donde queremos escribir la clave pública y la clave RSA anterior.

```
pub_keyfile = fopen("pub.key", "wb")
PEM_write_RSAPublicKey(pub_keyfile, key)
```

3) Análogamente, se genera el archivo `priv.key` con la clave privada.

```
PEM_write_RSAPrivateKey(priv_keyfile, key, NULL, NULL, 0, NULL,
NULL);
```

El cliente, por su parte, tiene la opción de introducir la clave adecuada antes de conectarse con el servidor, pinchando sobre el botón que tiene dibujada una llave, como ya vimos, con lo que aparece un cuadro de diálogo que se muestra en la figura 4-14. Lo

que aquí se escriba aparecerá en asteriscos, para evitar que alguna persona ajena pueda leer la clave.

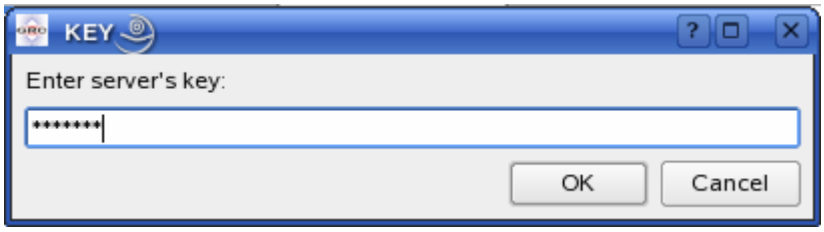


Figura 4-14: Diálogo para introducir la clave en el cliente

Una vez se ha establecido la conexión por el *socket* L2CAP se siguen los siguientes pasos hasta que el cliente logra autenticarse:

1) El servidor envía a través del *socket* la clave pública, que había guardado en el archivo `pub.key`.
2) El cliente recibe este archivo, lo guarda y extrae la clave pública mediante la función `PEM_read_RSAPublicKey()`.
3) Utilizando esta clave pública, el cliente cifra la clave "*server's key*" que había introducido previamente, juntamente con una serie de bits generados de forma aleatoria, mediante la función `RSA_public_encrypt()`
4) El cliente envía la clave cifrada de esta manera al servidor.
5) El servidor recibe esta clave y la descifra mediante `RSA_private_decrypt()`.
6) Por último, el servidor comprobará que esta clave descifrada, que le ha enviado el cliente, es la misma que se había establecido como clave del servicio. Si coinciden, el cliente queda autenticado y puede conectarse. En caso contrario, se cierra la conexión.

4.6 – Sumario

El servicio MANET_Gateway integra de forma sencilla y automática nodos Bluetooth en una red MANET o cualquier otro tipo de red.

El punto de partida del desarrollo ha sido el demonio `pand` de BlueZ, el encargado de implementar el perfil PAN en sistemas Linux. Partiendo de esta base, se han integrado todas las acciones necesarias para conseguir el objetivo, implementándose tanto el cliente como el servidor mediante una intuitiva interfaz gráfica.

De esta forma, el usuario no ha de realizar ninguna acción exterior al programa. En el caso de "Gateway Client", el usuario dará la orden de buscar dispositivos que le puedan ofrecer el servicio MANET_Gateway y elegirá a cuál quiere conectarse. Será informado de si la conexión ha tenido éxito o no, y la interfaz virtual de red de tipo BNEP creada será configurada adecuadamente por el programa. El usuario puede interrumpir en cualquier momento esta conexión mediante un simple *click*, o puede saber si ha existido una desconexión por causa ajena.

En el caso de "Gateway Server", el usuario puede decidir sobre una serie de parámetros básicos de configuración, como el número de clientes máximo que quiere atender (hasta un máximo de siete), qué dirección IP tendrá el *gateway*, qué máscara de red establece para la *piconet* Bluetooth, así como la seguridad. De todas formas, se han establecido unos parámetros por defecto, por si el usuario no desea personalizarlos. El programa se encarga de realizar todas las funciones propias para establecer el *gateway* y dar salida a los clientes conectados a la red externa. El usuario podrá ver en todo momento qué clientes hay conectados y puede eliminar las conexiones que desee. También puede interrumpir el servidor, es decir, parar la función que atiende conexiones entrantes.

Un punto muy importante del desarrollo ha sido la implementación de seguridad en el servicio, estableciéndose en varios niveles, según se elija. Así, podemos realizar un filtrado de dispositivos Bluetooth permitidos comprobando su dirección y también podemos asociar una clave al servicio, que se intercambiará mediante el algoritmo RSA. Para mayor seguridad, se recomienda utilizar ambos mecanismos a la vez.

Referencias de este capítulo:

[1] Jasmin Blanchette y Mark Summerfield, "C++ GUI Programming with Qt". Prentice Hall in association with Trolltech Press.

[2] Página oficial del proyecto BlueZ: http://www.bluez.org.

[3] Versión 2 de *GNU General Public License*, junio de 1991, por la *Free Software Fundation, Inc.* disponible en: http://www.gnu.org/copyleft/gpl.html.

[4] "An Introduction to Bluetooth programming in GNU/Linux", Albert Huang. Recurso *online* disponible en: http://people.csail.mit.edu/albert/bluez-intro/.

[5] "BlueZ Documentation", de Doxygen, recurso online disponible en: http://www-md.e-technik.uni-rostock.de/ma/hm27/uebung_hnp_2004/documentation/bluez-2.10-html/main.html.

[6] Página web de Marcel Holtmann: http://www.holtmann.org/linux/kernel.

[7] Descargas desde la página oficial del proyecto BlueZ: http://www.bluez.org/download.html.

[8] Documento detallado para establecer redes PAN en Linux con BlueZ, de Michael Schmidt, *University of Siegen*, Alemania: http://bluez.sourceforge.net/contrib/HOWTO-PAN.

[9] Douglas E. Comer y David L.Stevens, "Internet working with TCP/IP. Volume III. Client Server Programming and Applications. BSD Socket Version". Prentice Hall International Editions, 1993.

[10] Definición de RSA en Wikipedia: http://es.wikipedia.org/wiki/RSA.

[11] Criptografía asimétrica en Wikipedia: http://es.wikipedia.org/wiki/Criptograf%C3%ADa_asim%C3%A9trica.

Capítulo 5

Resultados experimentales

5.1 – Introducción

En los experimentos realizados los objetivos son dos: determinar el tiempo de conexión y estudiar la calidad del enlace variando diversos parámetros. En el tiempo de conexión se incluye tanto el tiempo de *inquiry* y búsqueda del servicio como el de conexión a la MANET una vez escogido el servidor. Asimismo se estudia el retardo introducido por la implementación de seguridad.

La medida de calidad del enlace utilizada será el *throughput* (velocidad de transmisión) que se puede conseguir con distintos tipos de paquetes (ver apartado 3.2.3.2). Para ello primero se realizó una prueba en un entorno casi libre de interferencias (el tercer sótano de un aparcamiento situado en la Universidad Politécnica de Valencia) para analizar el *throughput* en función de la distancia. Posteriormente se realizaron las pruebas en un entorno más real y susceptible a interferencias, como es el propio laboratorio del Grupo de Redes de Computadores donde se ha desarrollado todo el trabajo de este proyecto. Se analiza si estas interferencias influyen negativamente en el *throughput* y en qué casos, comparando los resultados obtenidos en el aparcamiento y en el laboratorio.

Figura 5-1: Adaptador Bluetooh de Conceptronic

Otro punto importante es la gestión del enlace cuando hay más de un dispositivo Bluetooth conectado al servidor MANET_Gateway. Primero se configuraron todos los enlaces con los respectivos clientes con el mismo tipo de paquete; en un segundo tipo de pruebas se asignaron distintos tipos de paquetes a cada cliente y se observaron interesantes resultados. Todas estas pruebas se realizaron para el enlace de bajada (de servidor a clientes), por lo que se hizo una prueba para el enlace de subida (de clientes a servidor). Como se ha comprobado que los resultados obtenidos son prácticamente los mismos (cosa lógica si se tiene en cuenta que sólo cambia el sentido de la transmisión) no tiene sentido reflejarlo en este documento.

Las pruebas se realizaron con dispositivos Bluetooth de Conceptronic, concretamente el adaptador Conceptronic CBTU, con interfaz USB (ver figura 5-1), de versión 1.1 y clase 2 (alcance de 10 m).

5.2 – Tiempo de conexión

Nos interesa averiguar cuánto tarda un dispositivo Bluetooth cliente en conectarse a una MANET mediante nuestro servicio. En este tiempo de conexión se incluye tanto el tiempo que emplea el cliente en encontrar el servicio, como el que emplea para conectarse al servidor escogido. En los siguientes dos apartados se muestra el estudio de estos tiempos por separado y, en un tercer apartado, el reparto de tiempo entre estos procesos.

5.2.1 – Descubrimiento del servicio

Lo primero que ha de hacer un cliente para conectarse a la MANET es encontrar los servidores Bluetooth que pueden ofrecerle el servicio MANET_Gateway. Para ello, como se explicó en el capítulo 4, se hacen en realidad dos operaciones:

1) Búsqueda de dispositivos cercanos mediante el *inquiry*.
2) Búsqueda del servicio en cada dispositivo encontrado.

A estos dos tiempos juntos los vamos a denominar tiempo de descubrimiento del servicio. El tiempo de *inquiry* es siempre fijo y, en nuestro caso, configurable por el cliente en múltiplos de 1.28 segundos. En [1] se recomienda un tiempo de *inquiry* de 10.24 segundos, lo que garantiza que se sondeen todas las frecuencias. Sin embargo, en [2] se demuestra que reduciendo este tiempo a la mitad, 5.12 segundos, es posible localizar el 99% de los dispositivos en el rango de transmisión.

Por su parte, la búsqueda del servicio aumentará, como es lógico, en tanto en cuanto lo haga el número de dispositivos encontrados. Esto queda demostrado en [3], donde se recomienda además que los dispositivos Bluetooth que actúen como clientes del servicio se configuren en modo no visible para que no puedan ser encontrados cuando otro dispositivo realice un *inquiry*. De esta forma se intenta paliar el problema, pero la existencia de otros dispositivos Bluetooth cercanos en modo visible es imposible de controlar. Por esta razón se ha supuesto un caso que servirá para determinar un tiempo de referencia. Este supuesto es el caso mejor que se puede dar, y es que sólo haya un

dispositivo en modo visible dentro del rango de transmisión y es justamente el que ofrece el servicio. En estas condiciones, y para mediciones realizas a una distancia de 2 metros, el tiempo de descubrimiento de dispositivo y servicio es de 5.675 segundos.

5.2.2 – Tiempo de conexión al servidor

Una vez encontrado uno o varios servidores, el siguiente paso es conectarse a uno de ellos. En la figura 5-2 se observa la variación del tiempo de conexión con la distancia a partir del instante en que el usuario selecciona uno de los servidores y elige la opción de conectarse. Se muestra el tiempo de conexión en caso de que el servidor no implemente seguridad y en caso de que sí lo haga. Para la configuración con seguridad se ha elegido el nivel 3, es decir, comprobación de MACs permitidas y asociación de clave.

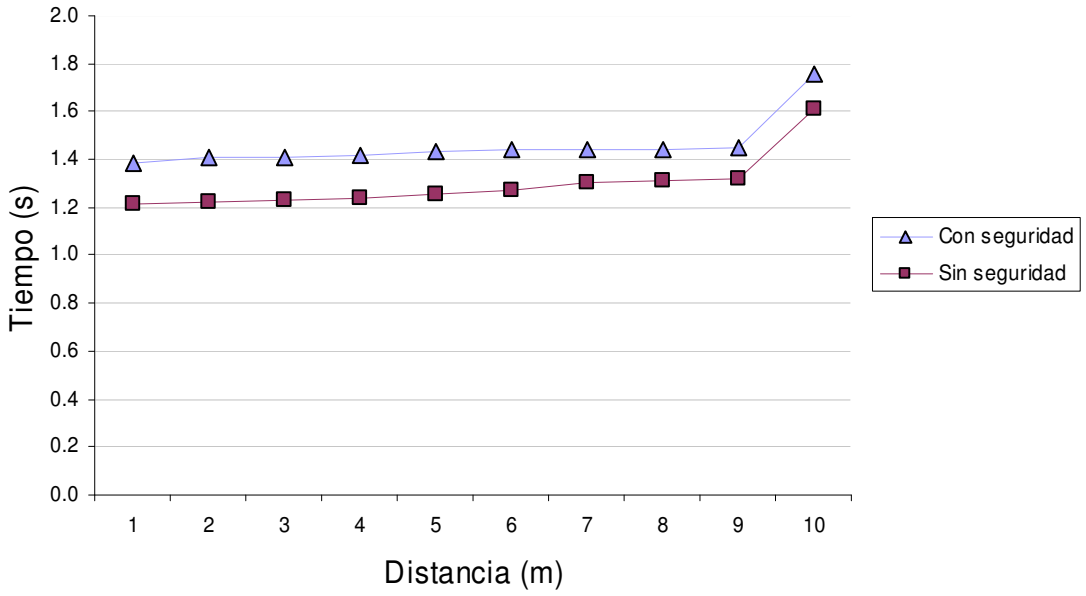


Figura 5-2: Tiempo de conexión en función de la distancia

Como se puede ver en la figura, al llegar a 10 metros (límite teórico de cobertura Bluetooth) el tiempo de conexión aumenta. Para distancias hasta los 9 metros, el tiempo de conexión se mantiene casi constante. El tiempo de conexión medio para la implementación con seguridad es un 11 % superior al de sin conexión. Este último tiene un valor medio de 1.26 segundos, siempre refiriéndonos a distancias inferiores al límite teórico Bluetooth.

5.2.3 – Reparto de tiempos de conexión

En la figura 5-3 puede verse el reparto de tiempo de conexión total, incluyendo el *inquiry*, la búsqueda del servicio y la conexión con el servidor en sí. El *inquiry* se ha fijado en 5.12 segundos y ocupa casi tres cuartas partes del tiempo de conexión, concretamente un 72%. Para la búsqueda del servicio se dedica tan sólo un 7,82% del reparto total, pero este valor es sólo para el caso en que se descubre un dispositivo que

oferta el servicio, el denominado caso mejor comentado anteriormente. Si se encontraran más dispositivos, este porcentaje aumentaría.

Por su parte, el tiempo de conexión una vez encontrado el servidor representa un 17.81% del tiempo total. En caso de implementarse seguridad, aumentaría el tiempo de conexión en 2.19%, una cifra casi despreciable.

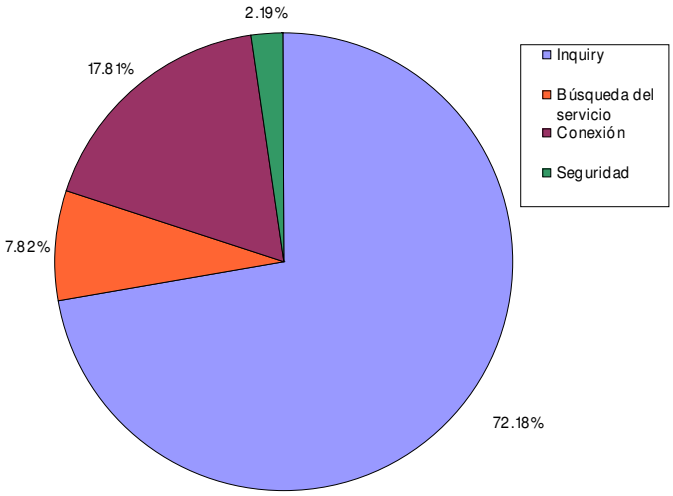


Figura 5-3: Reparto del tiempo de conexión

5.3 – *Throughput* en función de la distancia

La realización de esta prueba ha tenido lugar en el sótano tercero de un garaje situado en la Universidad Politécnica de Valencia, dado que es el sitio más cercano al laboratorio con un nivel de interferencias más bajo. El objeto de estudio es el cálculo del *throughput* en función de la distancia y para los diversos tipos de paquetes: con y sin corrección de errores (DM y DH, respectivamente) y de 1, 3 y 5 ranuras. El experimento consiste en calcular el *throughput* del enlace cuando el cliente se descarga del servidor un archivo de 1 Mbyte vía FTP (*File Transfer Protocol*), variando el tipo de paquete en el enlace de bajada (es decir, servidor-cliente). Los resultados presentados son media de 10 valores obtenidos en experimentos distintos. El enlace de subida (cliente-servidor) se mantuvo en todos los casos con tipo de paquete DH1, pero la influencia es mínima dado que lo único que se envía del cliente al servidor son reconocimientos TCP (ACKs). Como referencia de distancias se ha tomado la anchura de las plazas de garaje, por lo que se sitúan las medidas a 2.7 m (anchura de una plaza), a 5.4 m (anchura de dos plazas) y a 10.4 m (anchura de cuatro plazas, tres de 2.7 m y una de 2.3 m).

En la figura 5-4 se pueden observar los valores de *throughput* para paquetes de tipo DH. Se verifica que con DH3 se obtiene un aumento proporcional al número de ranuras, pero con DH5 este aumento no es proporcional dado que existe un equilibrio entre el *throughput* que puede llegar a obtenerse y los errores producidos debido a la mayor longitud del paquete. Por lo tanto, DH5 no ofrece tantas ventajas respecto a DH3 como

DH3 respecto a DH1. Así, DH3 mejora respecto al DH1 en un 305% (es decir, el triple) y DH5 mejora respecto DH3 en tan sólo un 119% y respecto a DH1 en un 364%.

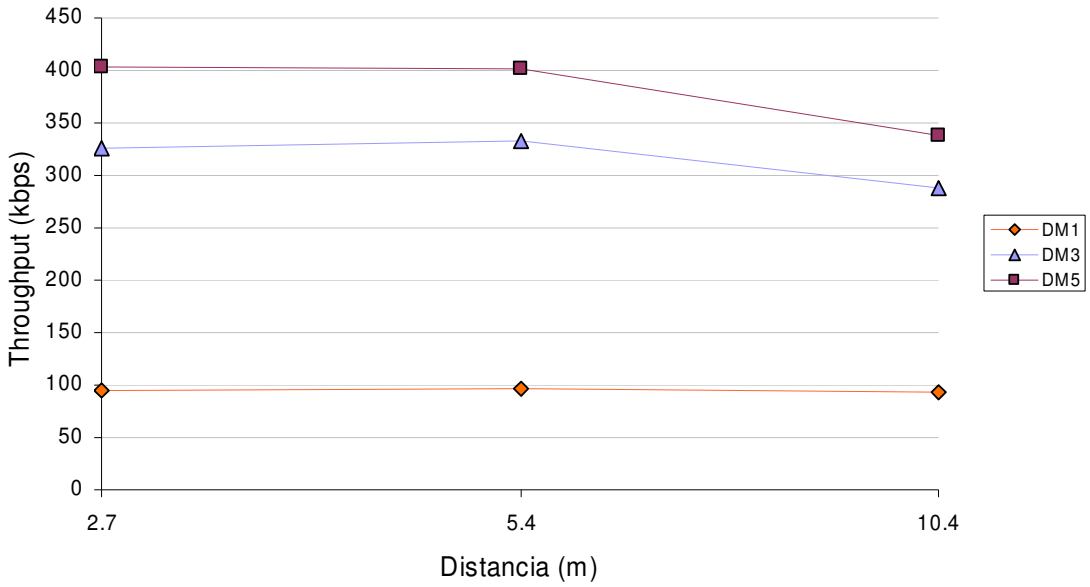


Figura 5-4: *Throughput* en función de la distancia para paquetes DH

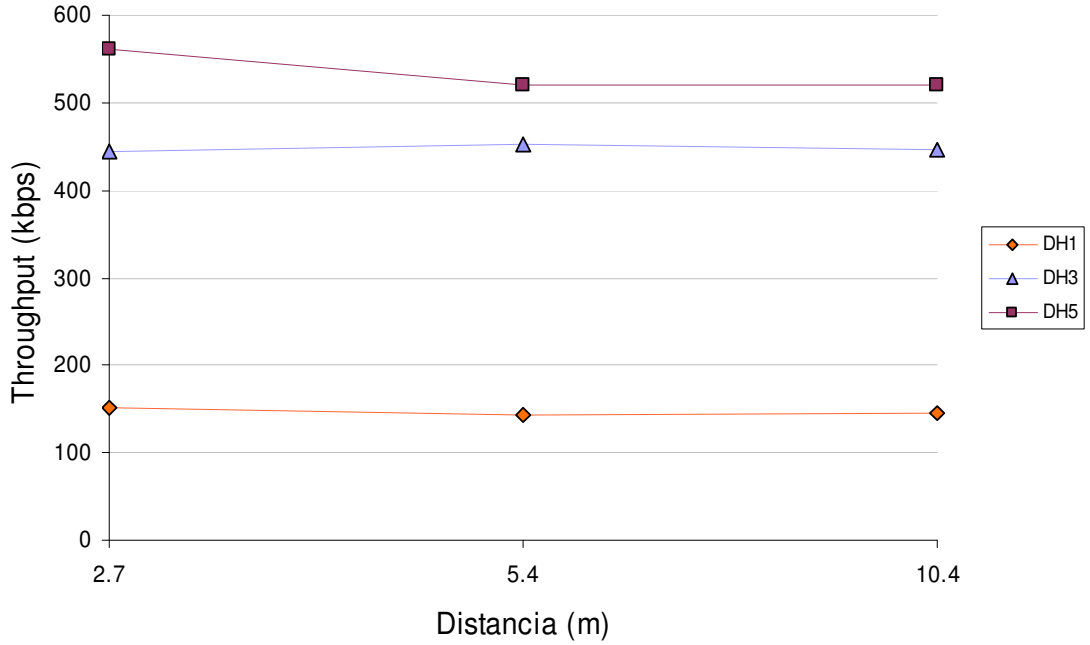


Figura 5-5: *Throughput* en función de la distancia para paquetes DM

En la figura 5-5 se presentan los resultados de medidas similares, pero en este caso con paquetes de tipo DM. Vemos que el *throughput* disminuye respecto a los paquetes de

tipo DH. Esto es debido a que DM utiliza un código de corrección de errores y, por lo tanto, destina parte de los bytes del paquete a este código, con lo que transporta menos bytes de carga útil que los paquetes DH. También se observa que el *throughput* para los paquetes DM1 se mantiene prácticamente constante sea cual sea la distancia, pero que en los casos de los paquetes DM3 y DM5 el *throughput* disminuye a partir de los 5.4 metros, viéndose más afectados por la distancia. En el caso de los paquetes DH, tanto DH1 como DH3 mantenían constante el *throughput* con la distancia, y en el caso de los paquetes DH5 el *throughput* se ve disminuido pero no a partir de los 5.4 metros sino de los 2.7, manteniéndose constante a partir de los 5.4 metros.

Por otra parte, vemos que en ninguno de los distintos casos evaluados se llega a alcanzar la velocidad asimétrica máxima que se establece en el estándar Bluetooth, a pesar de trabajar en un entorno casi libre de interferencias. Esto se debe a que estamos trabajando a nivel de aplicación, no a nivel físico, por lo que se añaden más cabeceras al paquete básico Bluetooth que reducen el débito útil. En la tabla 5.1 puede verse el *throughput* ideal establecido en el estándar, el obtenido mediante experimentación (haciendo la media de todos los valores a las distintas distancias) y el porcentaje de empeoramiento de las medidas experimentales respecto al ideal.

Tipo de paquete	*Throughput* ideal(kbps)	*Throughput* experimental (kbps)	Porcentaje empeoramiento (%)
DM1	108.8	94.5	13.14
DM3	387.2	315.9	18.41
DM5	477.8	381.5	20.15
DH1	172.8	146.8	15.05
DH3	585.6	447.4	23.60
DH5	723.2	534.5	26.09

Tabla 5-1: Comparación entre el *throughput* experimental y el teórico

En el caso de los paquetes DM el empeoramiento es menor, lo que puede ser debido de nuevo a que implementan FEC. Dentro de un mismo tipo de paquetes, conforme se aumenta el número de ranuras el *throughput* se aleja del umbral máximo teórico.

5.4 - Interferencias

En el entorno donde se han realizado las pruebas del laboratorio se captan las siguientes redes de tecnología IEEE 802.11 susceptibles de causar interferencias en nuestro servicio:

SSID	Canal	Frecuencia	Fuerza de la señal
CASTADIVA	3	2.422 GHz	100%
eduroam	8	2.447 GHz	60%
UPVNET	8	2.447 GHz	70%
borles	6	2,437 GHz	80%
eduroam	5	2,432 GHz	40%
UPVNET2G	5	2,432 GHz	40%
DISCA_CAN	11	2,462 GHz	70%
camelot	10	2,457 GHz	90%

Tabla 5-2: Redes IEEE 802.11 captadas en el laboratorio

El entorno del laboratorio es, pues, un entorno más realista si se tiene en cuenta que, en nuestro escenario de aplicación, una red *ad hoc*, existirán nodos con interfaz IEEE 802.11.

En la figura 5-6 se muestra una comparativa entre las mediciones efectuadas en el aparcamiento y en el laboratorio, para un mismo cliente situado a 2.7 metros (recordemos, la anchura de una plaza de garaje) del servidor, y para todos los tipos posibles de paquetes ACL.

Tal como sería de esperar, el *throughput* cuando utilizamos paquetes de tipo DM prácticamente no se ve afectado por las interferencias, porque incluye FEC. Para los paquetes tipo DH que no incluyen este tipo de protección, si se verifican diferencias más significativas.

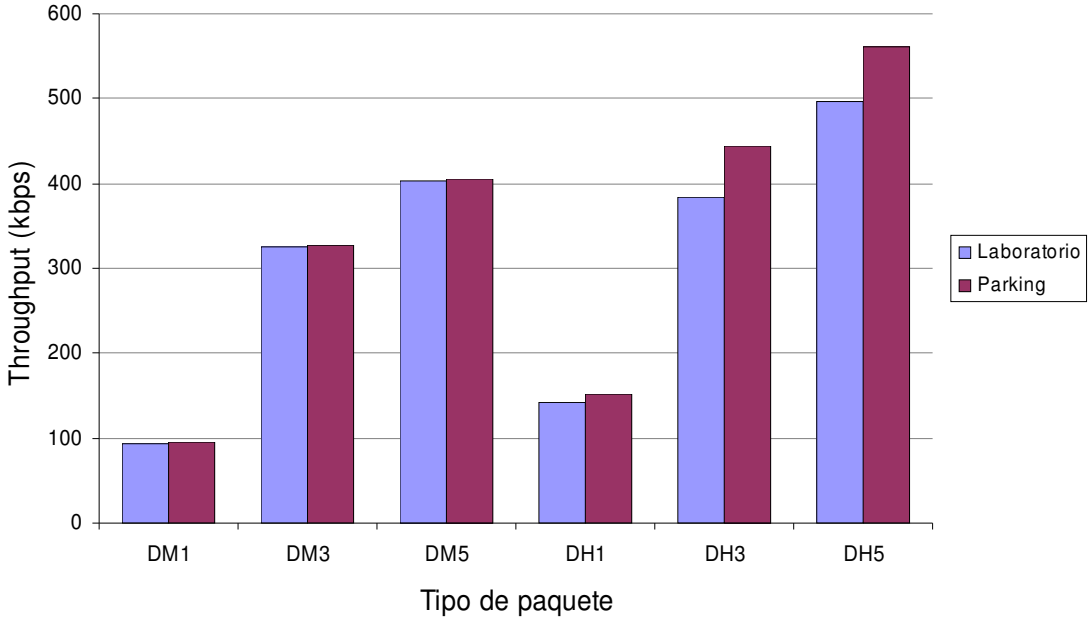


Figura 5-6: Comparativa del *throughput* obtenido en el aparcamiento y en el laboratorio

El hecho de que el *throughput* no se vea afectado si utilizamos paquetes DM puede explicarse si observamos la probabilidad de pérdida de paquetes, *P*. Los paquetes DM están codificados con un código FEC de 2/3, así que, por ejemplo, en cada bloque, 15 bits son usados para codificar 10 bits de datos y este bloque codificado puede corregir un bit en cada bloque de 15 bits. Por tanto, la probabilidad de pérdida en paquetes DM puede aproximarse por la siguiente relación [4]:

$$P = 1 - ((1-b)^{15} + 15 \cdot b \cdot (1-b)^{14})^{n/15}$$

Donde *b* representa la tasa de error de bit (BER, *Bit Error Rate*) y *n* el número de bits. Para paquetes DH esta fórmula es:

$$P = 1 - (1-b)^{n}$$

Si tomamos, por ejemplo, un valor de BER de 0.002 y un valor de n de 40, en el primer caso se obtiene una probabilidad de pérdida de 0.0011 y en el segundo de 0.0769. Por tanto, los paquetes DM son más robustos, es decir, su probabilidad de pérdida de paquete es menor, lo que los hace más inmunes frente a las interferencias.

Por otra parte, el *throughput* se ve más afectado en los paquetes de tipo DH3 y DH5, empeorando en el laboratorio en un 13.44% y un 11.3% respectivamente. En cambio, en los paquetes DH1 el porcentaje de empeoramiento es de sólo un 6%. Obviamente, a mayor número de bits (es decir, a mayor número de ranuras), la probabilidad de pérdida en los paquetes es mayor, por lo que en un entorno con interferencias habrá más pérdida de paquetes si el enlace está configurado para un tamaño de paquetes superior.

5.5– Gestión del enlace Bluetooth

Las pruebas de este apartado se han realizado en el laboratorio, estando los dispositivos clientes a una distancia entre 1 y 3 metros del servidor. El objetivo es cuantificar cómo se realiza la gestión del enlace al haber más de un dispositivo conectado al servidor y recibiendo datos del mismo simultáneamente. La prueba consistía, al igual que la realizada en el aparcamiento, en la descarga simultánea por parte de los clientes conectados de un archivo de 1 Mbyte situado en el servidor. En el enlace de subida se configuró en todos los casos el tipo DH1, aunque ya hemos dicho que esto se hace por cuestiones de uniformidad, no influyendo en las mediciones de forma significativa.

En un primer banco de pruebas todos los enlaces de bajada se configuran con el mismo tipo de paquete, por lo que todos los clientes están en igualdad de condiciones. Los resultados se muestran en el apartado 5.5.1.

En el segundo banco de pruebas no todos los enlaces se configuran con el mismo tipo de paquete. Esto da lugar a un reparto distinto del ancho de banda disponible, como veremos en el apartado 5.5.2.

5.5.1 – Mismo tipo de paquete en todos los enlaces

En la figura 5-7 se representa el *throughput* en función del número de clientes en el caso de paquetes DH; en la figura 5-8 se representa lo mismo, pero utilizando paquetes DM. De nuevo se observa, para el mismo número de ranuras, un mayor *throughput* en el caso de paquetes DH, que es también mayor cuando aumenta el número de ranuras.

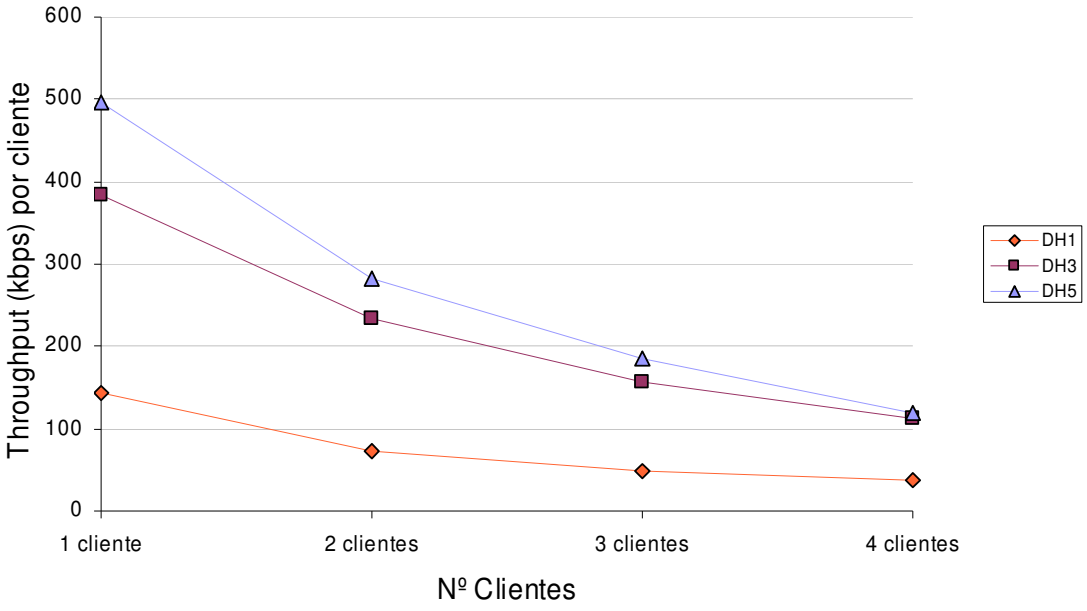


Figura 5-7: *Throughput* en función del número de clientes para paquetes DH

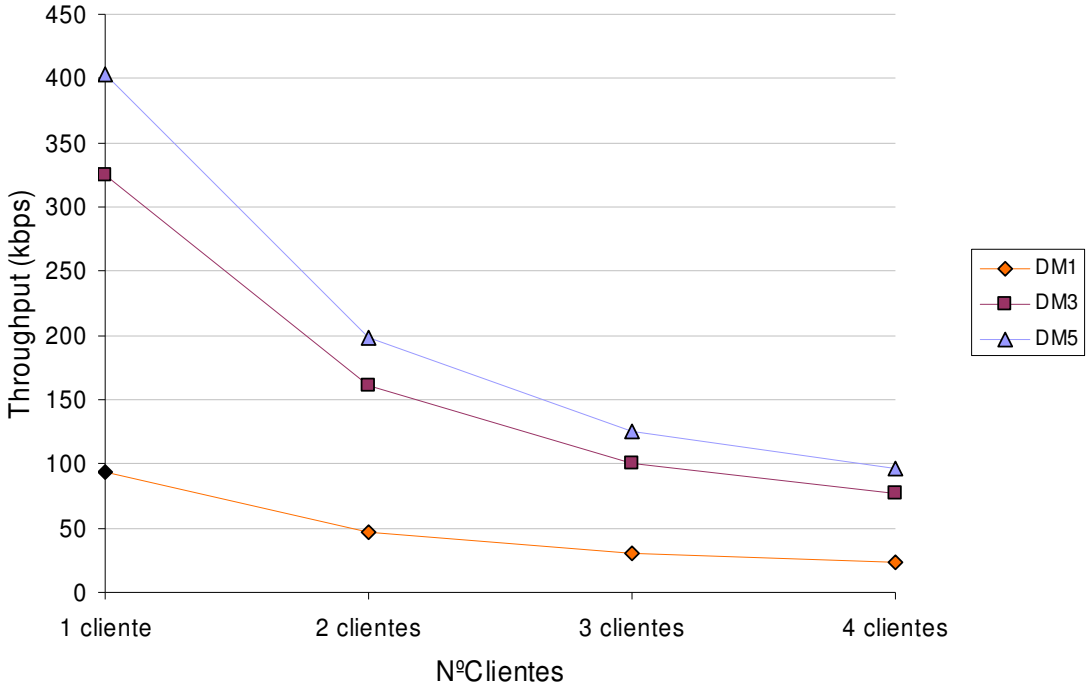


Figura 5-8: *Throughput* en función del número de clientes para paquetes DM

Al aumentar el número de clientes, el *throughput* por cliente disminuye considerablemente, siendo esta disminución proporcional al número de clientes conectados. Para el caso de dos clientes, el *throughput* por cliente disminuye, alcanzando un valor alrededor de la mitad (un 50.5% en paquetes DM y un 43.7% en paquetes DH). Si hay tres clientes el *throughput* por cliente se divide aproximadamente entre tres, disminuyendo un 68.8% en paquetes DM y un 62.6% en paquetes DH. Por último, cuando existen cuatro clientes, el *throughput* que obtiene cada cliente es un cuarto respecto al del caso en que sólo hay un cliente activo, disminuyendo un 76.03% y un 73.63% en el caso de paquetes DH y DM respectivamente. La disminución es más pronunciada en paquetes DM, pero en general se puede concluir que el *throughput* agregado se mantiene aproximadamente constante, lo que refleja un reparto adecuado de los recursos a medida que aumentamos el número de clientes.

5.5.2 – Variando el tipo de paquete en los diversos enlaces

El objetivo de este último conjunto de pruebas es determinar cómo se gestiona el ancho de banda disponible en Bluetooth cuando los enlaces con distintos clientes se configuran con tipos de paquete distintos. En concreto, se quiere determinar qué ocurre asignando distinto número de ranuras y qué ocurre cuando, fijado el número de ranuras, se varía el tipo de paquete DM o DH. Esta información es útil si deseamos, por ejemplo, asignar distinta prioridad a los clientes.

Para empezar se realizaron las pruebas en el caso de dos clientes conectados, Cliente A y Cliente B. Primero se efectuaron todas las posibles combinaciones con tipo de paquete

DH y variando el número de ranuras. El resultado se puede observar en la figura 5-9. En ella se puede ver el *throughput* obtenido en los 3 casos: Cliente A con tipo de enlace DH1 y Cliente B con tipo DH5, A con DH1 y B con DH3, y por último A con DH3 y B con DH5.

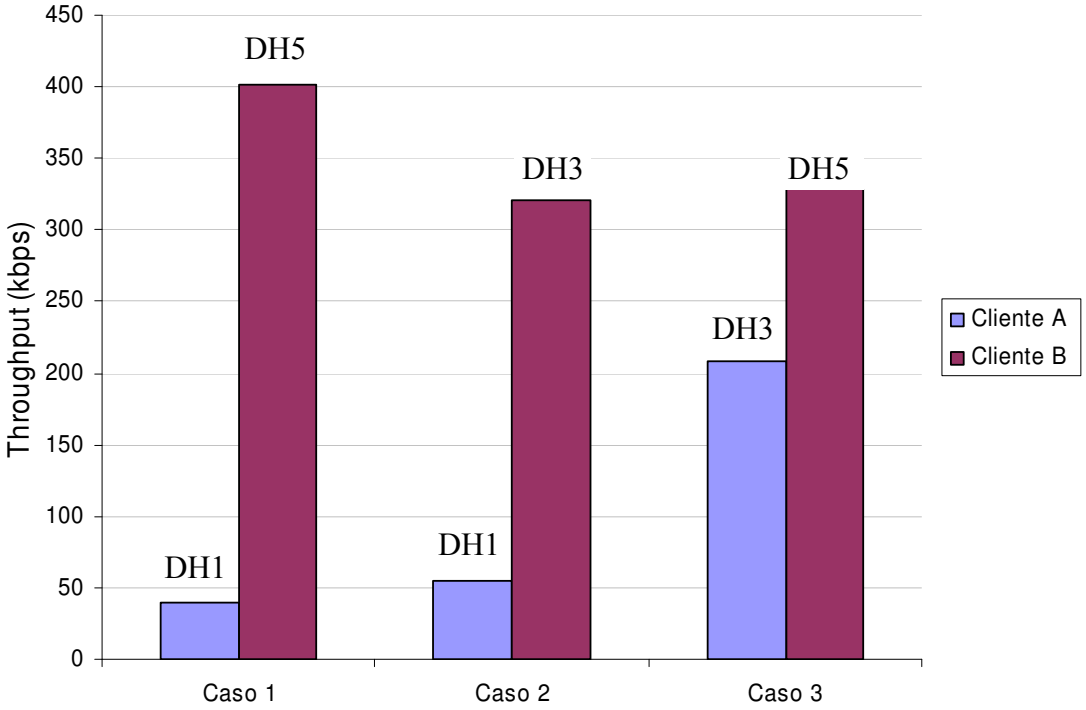


Figura 5-9: *Throughput* variando el número de ranuras para dos clientes

A mayor número de ranuras mayor se favorece dicho enlace frente al de menor número de ranuras. Esto es más notable si uno de los enlaces se configura con tipo de paquete DH1. Por ejemplo, en el Caso 1, el Cliente B con tipo DH5 alcanza un *throughput* de 402 kbps, un 30% más que en el caso de dos clientes configurados ambos con DH5 (ver figura 5-7). En cambio, el *throughput* en el Cliente A cuyo enlace es DH1, ha disminuido en un 47% respecto el caso análogo con DH1. El Caso 2 arroja resultados similares: el enlace con DH3 aumenta el *throughput* en un 27% y el de DH1 disminuye el suyo en un 26% y, por lo tanto, no disminuye tanto como cuando el otro enlace es DH5.

En el Caso 3, existe un enlace con DH3 y otro con DH5. El *throughput* en el enlace con DH3 disminuye en un 11%, y el correspondiente al enlace con DH5 aumenta un 14%. En este caso las diferencias no son tan notables pero se sigue favoreciendo al enlace con más ranuras.

Los resultados para las pruebas realizadas manteniendo el número de ranuras, pero cambiando el tipo de paquete, se muestran en la figura 5-10. Para el Caso 1, el enlace DH5 mejora el *throughput* en un 2% respecto al caso de dos clientes con el mismo tipo de enlace y DM5 disminuye en un 3%. Para el Caso 2, el *throughput* aumenta un 3% para DH3 y disminuye un 1.6% para DM3. Estas diferencias son tan mínimas que se pueden considerar despreciables al caer dentro del margen de error de las medidas.

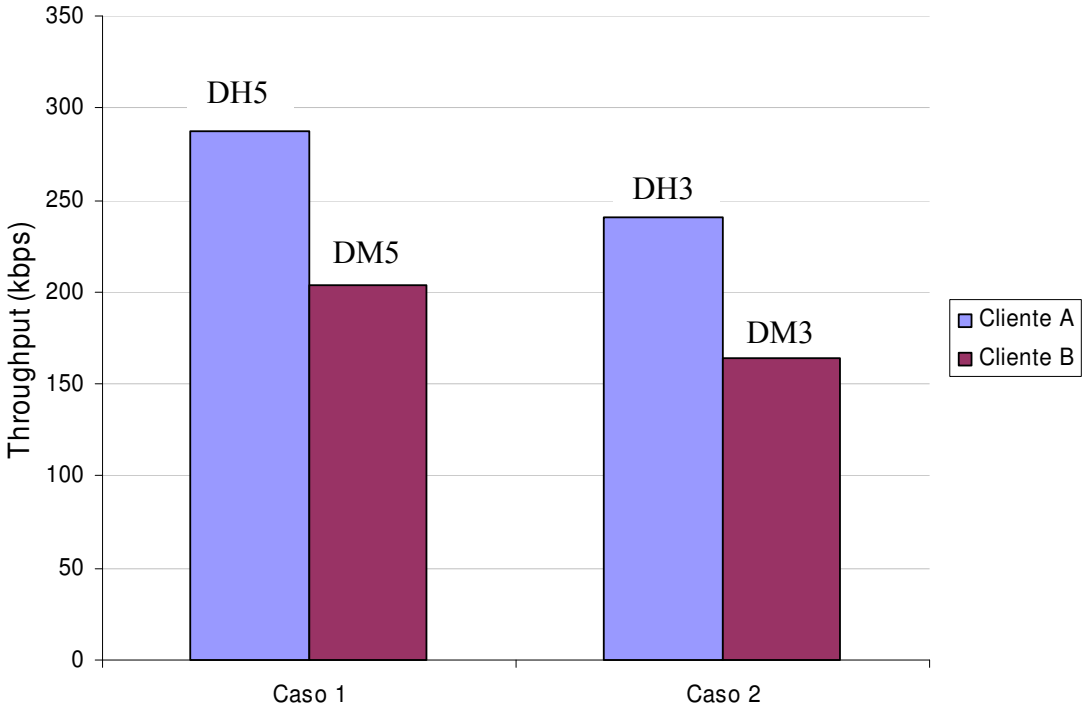


Figura 5-10: *Throughput* variando el tipo de paquete para dos clientes

Posteriormente se realizaron pruebas análogas pero con tres y cuatro clientes, para distintas combinaciones. En las figuras 5-11, 5-12, 5-13 y 5-14 se encuentran las gráficas correspondientes. Los resultados obtenidos han sido análogos que para el caso de dos clientes, es decir, el enlace configurado con más ranuras aumenta su *throughput* en detrimento de los demás, siendo esto más notable en el caso de DH3 y DH5 frente a DH1. Esto es lógico si tenemos en cuenta que utilizar un mayor número de ranuras significa ocupar durante más tiempo los recursos físicos disponibles en Bluetooth. Asimismo, para un mismo número de ranuras pero distinto tipo de paquete, el tráfico no varía, dado que la asignación del número de ranuras es la misma.

Los resultados presentados muestran que la tecnología Bluetooth hace posible la discriminación entre distintos clientes mediante una asignación inteligente de tipos de enlace a los distintos clientes. Con el fin de compaginar la asignación de prioridades a los clientes con una utilización elevada de los recursos se recomienda combinar simplemente conexiones del tipo DH3 y DH5, obteniéndose así dos prioridades y ancho de banda relativamente elevado para todos los clientes.

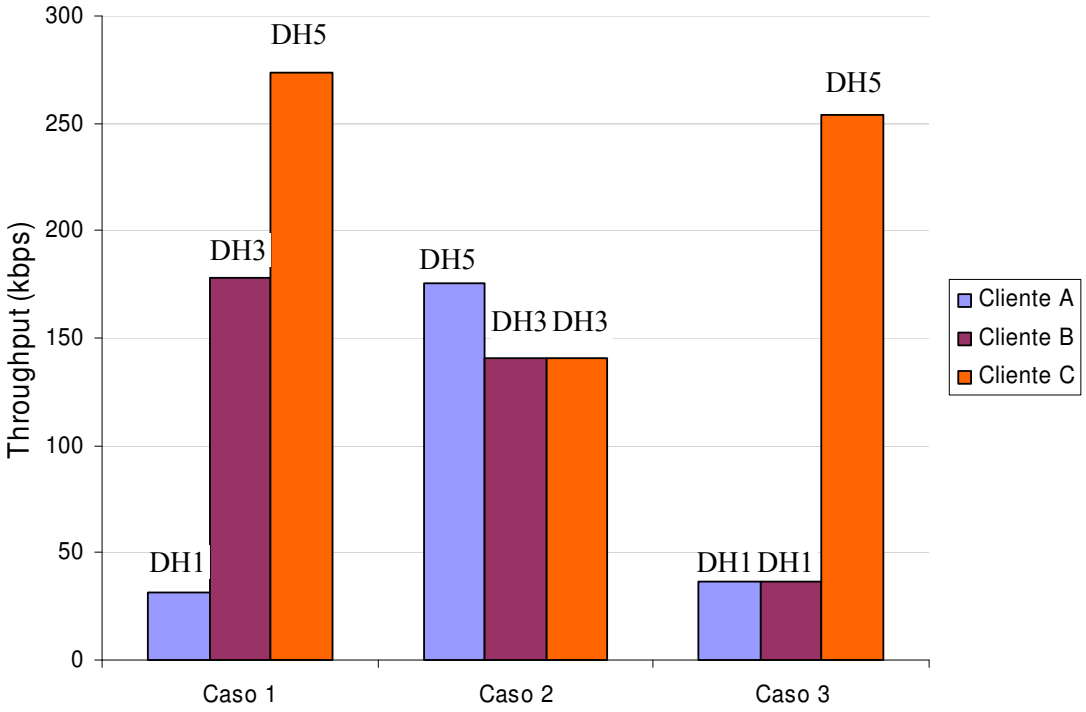


Figura 5-11: *Throughput* variando el número de ranuras para tres clientes

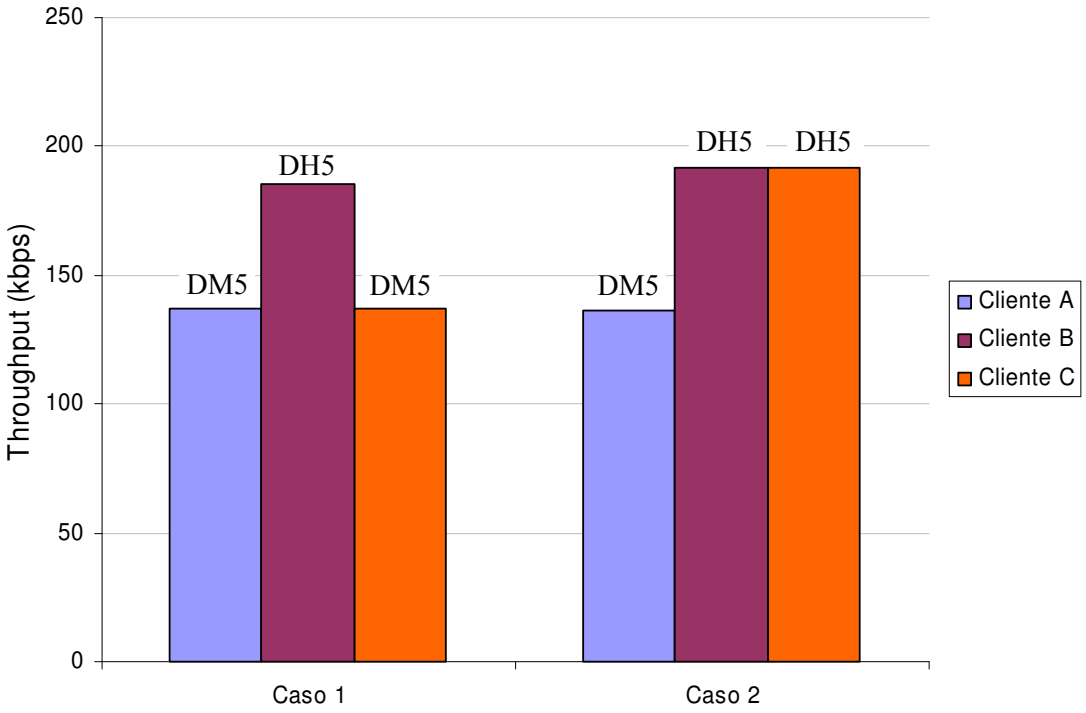


Figura 5-12: *Throughput* variando el tipo de paquete para tres clientes

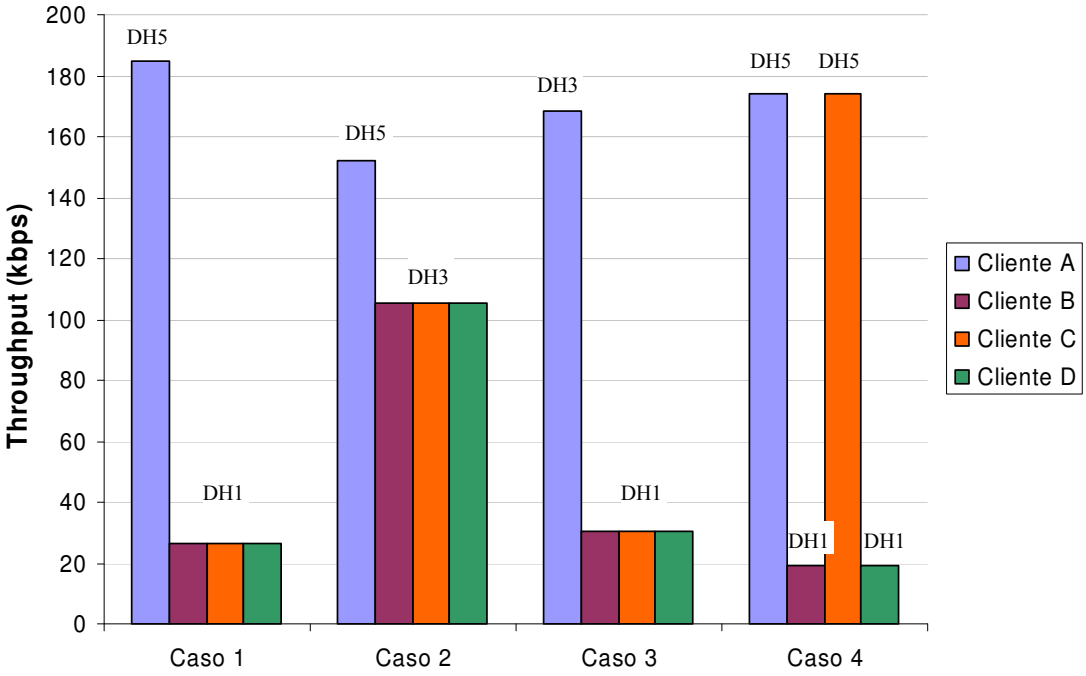


Figura 5-13: *Throughput* variando el número de ranuras para cuatro clientes

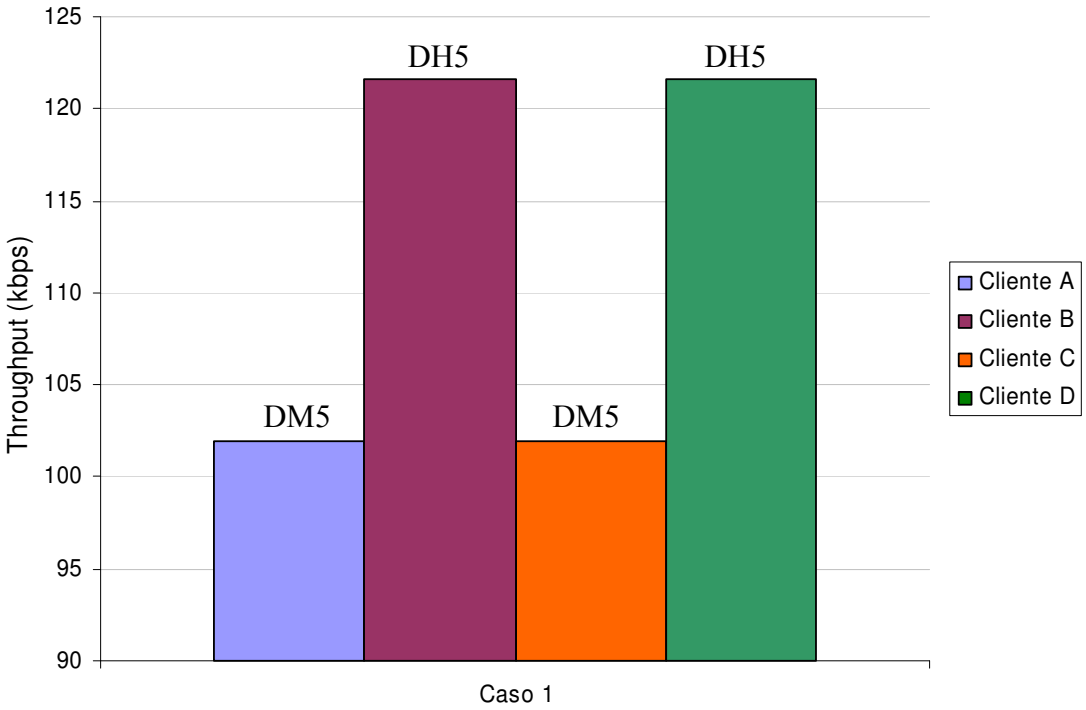


Figura 5-14: *Throughput* variando el tipo de paquete para cuatro clientes

Referencias de este capítulo:

[1] “Specification of the Bluetooth System, Core Version 1.1”. Bluetooth SIG, 2001. Disponible en www.bluetooth.com.

[2] B. Peterson, R.Baldwin y J.Kharoufeh, “Bluetooth Inquiry Time Characterization and Selection”. Presentado en *IEEE Transactions on Mobile Computing*, septiembre de 2006.

[3] José Cano Reyes, Eduardo Burgoa, Carlos T.Calafate, Juan Carlos Cano y Pietro Manzoni, “An autoconfiguration method for IEEE 802.11 based MANETs using Bluetooth”. XVII Jornadas de Paralelismo, Albacete, septiembre 2006.

[4] Ling-Jyh Chen, Rohit Kapoor, M.Y.Sanadidi y Mario Gerla, *Department of Computer Science*, *University of California*, Los Angeles, “*Enhancing Bluetooth TCP Throughput via Link Layer Packet Adaptation*”. Presentado en *IEEE International Conference on Communications*, junio 2004.

Capítulo 6

Conclusiones

En este proyecto se ha desarrollado el servicio Bluetooth **MANET_Gateway**. Este servicio permite a nodos con tecnología Bluetooth integrarse en una red inalámbrica *ad hoc* de una forma sencilla, rápida y transparente para el usuario. Como parte de la memoria, se ha justificado la importancia de este tipo de redes inalámbricas, denominadas MANET, y sus aplicaciones, que constituyen todo un campo de desarrollo presente y futuro. El interés de integrar la tecnología Bluetooth ha quedado patente por su amplia aceptación y generalización en dispositivos inalámbricos de todo tipo, junto con su bajo consumo de energía y su facilidad de uso, que incluye la posibilidad de descubrir dispositivos cercanos, conectarse a sus servicios y formar pequeñas redes con soporte para TCP.

Aunque inicialmente el servicio MANET_Gateway se concibió para un escenario de uso concreto, como es su utilización en redes MANET, su utilización no se restringe a este caso. De hecho, el servicio desarrollado constituye un verdadero *gateway* o puerta de enlace Bluetooth hacia cualquier tipo de red, permitiendo incluso que los dispositivos Bluetooth puedan conectarse a Internet. Por tanto, el abanico de posibles escenarios y ámbitos de aplicación se amplía considerablemente.

Por su parte, la implementación de seguridad, no constituyendo un objetivo principal, ha ido adquiriendo importancia conforme se desarrollaba el trabajo. Primero fue necesario realizar un estudio previo, con el fin de conocer las soluciones que presenta Bluetooth en el campo de la seguridad y cómo se implementan dichas soluciones en BlueZ, el API de desarrollo utilizado. Finalmente se ha optado por la realización de un desarrollo propio, denominado “Pseudo Modo 2”, que proporciona unos niveles de seguridad más flexibles y acorde con los propósitos definidos.

Este sistema de seguridad “Pseudo Modo 2”, se puede basar en el filtrado de direcciones Bluetooth y en autenticación mediante criptografía asimétrica a nivel de servicio. Todo esto sin tener que renunciar a una autenticación criptográfica a nivel de enlace, lo que se consigue configurando el Modo 3 de seguridad Bluetooth, totalmente compatible con nuestro sistema. La implementación de seguridad se ha integrado plenamente en el programa, consecuente con la filosofía de facilidad de uso que se ha seguido a lo largo

de todo el desarrollo. Si bien no puede asegurarse nunca la invulnerabilidad de cualquier solución software de seguridad, esta implementación de seguridad es muy robusta, sobretodo cuando se configura el nivel de máxima seguridad, y más aún si se combina con el Modo 3 ya contemplado por el estándar Bluetooth.

Una vez finalizada la fase de desarrollo de la aplicación, se han realizado una serie de pruebas con el fin de analizar el funcionamiento. Por una parte, se ha perseguido determinar el tiempo de conexión, entendiéndose éste como el tiempo destinado por un cliente para conectarse a la red correspondiente empleando nuestro servicio. Por otra parte, se ha estudiado la calidad del enlace Bluetooth establecido así como la gestión del mismo. También se ha realizado un breve estudio sobre la posible influencia de las interferencias en el enlace.

Por tanto, en un primer banco de pruebas, se ha medido el tiempo de conexión, comprobándose que en pocos segundos cualquier nodo Bluetooth puede conectarse a la MANET (o a la red correspondiente) mediante el servicio MANET_Gateway. Aproximadamente tres cuartas partes de este tiempo se emplea en el descubrimiento del servicio, que comprende tanto la búsqueda de dispositivos Bluetooth cercanos como la búsqueda del servicio en cada dispositivo encontrado. Este tiempo de descubrimiento es mayor cuanto mayor es el número de dispositivos Bluetooth en modo visible que se encuentren dentro del rango de transmisión. Asimismo, se ha comprobado que el retardo en el tiempo de conexión introducido cuando se utilizan los mecanismos de seguridad es mínimo, por lo que a efectos de usuario no será notable.

A modo de validación, se han efectuado diversas pruebas con el objetivo de observar la calidad del enlace Bluetooth conseguido y la gestión del mismo, mediante la medida del *throughput* conseguido configurando el enlace con los distintos tipos de paquetes que existen en Bluetooth: DH y DM, cada uno de una, tres y cinco ranuras. Como era de prever, el mayor *throughput* se consigue utilizando paquetes DH frente a paquetes DM, dado que estos últimos realizan corrección de errores FEC, lo que supone dedicar cierto número de bits a esta tarea, con la consiguiente disminución de la eficiencia. De igual manera, a mayor número de ranuras se alcanza un mayor *throughput*, siendo más notable la mejora en el caso del paquete de tres ranuras respecto a los de una sola ranura, que en el caso de los paquetes de cinco ranuras frente a los de tres. Esto se debe al equilibrio existente entre el *throughput* que puede llegar a obtenerse y los errores producidos debido a la mayor longitud del paquete.

Midiendo el *throughput* en función de la distancia en un entorno casi libre de interferencias (el sótano tercero de un aparcamiento), se constata que dicho *throughput*, en todos los casos contemplados para los distintos tipos de paquetes, permanece casi constante hasta el límite teórico de alcance radio para Bluetooth (diez metros). Comparando estos resultados con los obtenidos en un entorno sujeto a múltiples interferencias de redes IEEE 802.11 cercanas, el laboratorio del Grupo de Redes de Computadores, se ha puesto de manifiesto que el *throughput* en el caso de los paquetes DM se mantiene prácticamente igual en ambos entornos. Esto es debido precisamente a que la implementación de FEC hace a los paquetes DM más robustos frente a interferencias. Los paquetes DH, sin embargo, sí se ven afectados por las interferencias; con lo que su *throughput* disminuye en el entorno del laboratorio, siendo más notable esta disminución en el caso de paquetes de más de una ranura (los de tres y los de cinco) puesto que poseen mayor longitud.

En cuanto a la gestión del enlace, se han realizado pruebas para comprobar la equidad del reparto de los recursos en función de los clientes conectados, asignando a cada uno su correspondiente ancho de banda en proporción. Esto se corresponde al caso en que todos los enlaces a los diversos clientes se configuran con el mismo tipo de paquete. Haciendo variar el tipo de paquete para cada enlace, se realizaron diversas pruebas con configuraciones distintas, de los que se concluye los enlaces configurados con paquetes de mayor número de ranuras se ven favorecidos frente a los de menos número de ranuras.

Como análisis final de los resultados obtenidos en los experimentos de validación del enlace Bluetooth, se recomienda combinar conexiones del tipo DH3 y DH5, obteniéndose así dos prioridades y ancho de banda relativamente elevado para todos los clientes. De esta manera, se compagina la asignación de prioridades a los clientes con una utilización elevada de los recursos. En cambio, si se trabaja en un entorno con muchas interferencias, es más recomendable trabajar con paquetes DM puesto que son más robustos frente a estas interferencias.

Por último, hay que resaltar el uso de Linux como plataforma avanzada para el establecimiento de comunicaciones Bluetooth; dado que ofrece una completa pila de protocolos (BlueZ) que es muy sencilla de manejar, con multitud de herramientas a disposición del usuario y con un potente entorno de desarrollo.

Como líneas de trabajo futuras se propone estudiar el efecto de protocolos de encaminamiento adecuados para MANETs y la posibilidad de integrar alguno de ellos en el servicio MANET_Gateway. Actualmente el servidor realiza funciones de NAT, por lo que enmascara a los clientes conectados al mismo. Con la puesta en marcha de un protocolo de encaminamiento, los nodos Bluetooth podrían ser visibles desde la MANET, aunque evitando tener que realizar funciones de encaminamiento como los demás nodos pertenecientes a la red *ad hoc*.

Dado que todas las pruebas experimentales han sido realizadas con dispositivos Bluetooth de versión 1.1, se propone un estudio futuro empleando dispositivos de versión 2.0, que teóricamente posibilitan el aumento de la velocidad al triple. Esto debería proporcionar un *throughput* mayor para el enlace Bluetooth y podría cumplir de forma más solvente los requisitos de ancho de banda de las aplicaciones actuales.

Otra posible ampliación futura puede ser la implementación de esta misma herramienta de forma no gráfica, es decir, en modo consola. Esto podría permitir su uso en dispositivos que no disponen de interfaz gráfica, por ejemplo en *routers* o puntos de acceso inalámbricos. De esta manera, se podría ejecutar el programa en *background* y automatizar ciertas tareas, como la búsqueda del servicio y la conexión a un servidor determinado en el caso del cliente o el registro del servicio y la escucha de conexiones entrantes en el caso del servidor.

Anexo

Requerimientos de la aplicación

Con el fin de asegurar un correcto funcionamiento de la aplicación, hay que tener en cuenta una serie de cuestiones referentes a los distintos elementos que intervienen en el sistema y la instalación de los mismos. De esta manera evitaremos que la aplicación no funcione o deje de funcionar de la forma adecuada por no tener instalada la versión adecuada de cierto paquete.

Como la aplicación ha sido realizada para sistemas Linux, funciona en cualquiera de las distribuciones (Mandriva, Debian, Fedora, Suse...). La distribución Linux utilizada para el desarrollo de este proyecto ha sido la Suse 10.0 y la versión del *kernel* 2.6.13.

Por supuesto, el primer requisito necesario es tener instalado el software de BlueZ. Como se dijo en el capítulo 4, BlueZ se distribuye en un conjunto de paquetes y su núcleo viene instalado en el *kernel* de Linux a partir de la versión 2.4 del mismo. Además del núcleo de BlueZ es necesario tener instalado el paquete bluez-utils-XX, que proporciona el conjunto de librerías BlueZ indispensables para la aplicación. Además, puede ser útil tener instalado el paquete bluez-utils-XX que, aunque no es necesario para que funcione nuestra aplicación, dispone de varias herramientas interesantes (ver Capítulo 4). Las versiones de estos paquetes usadas para el desarrollo de este proyecto han sido:

3) bluez-libs-2.19-2: librerías Bluetooth.
4) bluez-utils-2.19-5: demonios y herramientas Bluetooth.

Para la implementación gráfica se ha utilizado la librería Qt de Trolltech, de licencia GPL, que puede descargarse desde la página web oficial: http://www.trolltech.com/qt. En concreto, para el desarrollo de este proyecto hemos utilizado la versión 3.3.4 del programa, aunque ya se encuentra disponible la versión 4.

Qt dispone de la herramienta `qmake` que permite generar un archivo de tipo `makefile` a partir de un archivo menos complejo con extensión `.pro`. Como ejemplo, el archivo GUIServer.pro es el siguiente:

```
TEMPLATE        = app
LANGUAGE        = C++

unix:CONFIG     += qt warn_on release thread

unix:LIBS       += -lssl -lbluetooth

unix:INCLUDEPATH        += /usr/include/

HEADERS += gatewayserver.h \
        mainwindow.h \
        listsc.h \
        bnepc.h \
        thread.h \
        parameterdialog.h

SOURCES += gatewayserver.c \
        main.cpp \
        mainwindow.cpp \
        listsc.c \
        bnepc.c \
        thread.cpp \
        parameterdialog.cpp

#############################################################################
# Automatically generated by qmake (1.07a) Mon Feb 26 12:35:55 2007
#############################################################################

# Input
```

Para generar el archivo Makefile hay que ejecutar la siguiente orden en el directorio correspondiente (no es necesario ser superusuario):

```
#qmake -o Makefile GUIServer.pro
```

Una vez hecho esto, para generar el ejecutable `GUIServer` basta con efectuar:

```
#make
```

Análogamente se actúa para el caso del cliente.

Por tanto, para compilar el programa es necesario tener los paquetes Qt correspondientes. En concreto, las versiones de los paquetes Qt utilizadas para este proyecto han sido:

- qt3-3.3.4-28.3: biblioteca para el desarrollo de aplicaciones con GUI.
- qt3-devel-3.3.4-28: bibliotecas y archivos *include* necesarios para el desarrollo.
- qt3-devel-tools-3.3.4-28: constructor de interfaces de usuario y otras herramientas (asistente,

El último paquete no es necesario para la compilación, puesto que contiene los programas específicos de desarrollo.

Para la implementación de seguridad, concretamente para la comprobación de la clave, se hace uso de funciones del algoritmo RSA que se encuentran en la librería `openssl`, en nuestro caso:

- openssl-0.9.7g2-2: *Secure Sockets* y *Transport Layer Security*.
- openssl-devel-0.9.7g2: bibliotecas y archivos include necesarios para el desarrollo.

La página web del proyecto *OpenSSL* es http://www.openssl.org, donde puede encontrarse toda clase de documentación y las últimas versiones disponibles.

Como también se comentó en el capítulo 4, la herramienta `iptables` forma parte del *kernel* de Linux a partir de la versión 2.4. La página web con información acerca del proyecto Netfilter, del que forma parte esta herramienta, es http://www.iptables.org.

Tanto `iptables` como la librería `openssl` son necesarios para el cliente y para el servidor. En cambio, la herramienta `brctl` de Linux, indispensable para la creación del *bridge*, es requerida únicamente por el servidor, siendo el paquete utilizado en nuestro caso:

- bridge-utils-1.0.6-2: utilidades para configurar el *Linux Ethernet Bridge*.

Toda la información y paquetes disponibles de esta herramienta se pueden encontrar en http://bridge.sourceforge.net.

El código de *MANET_Gateway* se encuentra disponible en la página web del Grupo de Redes de Computadores, en el apartado de software:

http://www.grc.upv.es/6software.html