Universidad Politécnica de Valencia
Facultad de Informática

Proyecto Fin de Carrera

Diseño e implementación de una aplicación P2P en una red PAN con tecnología Bluetooth

---

Realizado por :
José Cano Reyes

Dirigido por :
Juan Carlos Cano Escribá

Resumen

En el mundo actual, la comunicación entre dispositivos electrónicos es una necesidad tecnológica básica, en particular, los dispositivos que se encuentran a corta distancia y que normalmente se comunican entre si por medios alambrados. Estos, utilizan una amplia gama de cables y conectores que hacen la comunicación, en un momento dado, limitada, falible e incomoda en situaciones donde existen demasiados cables.

Debido a este escenario, la tendencia ahora es establecer enlaces inalámbricos entre dichos dispositivos. Tecnologías existentes como HomeRF™, IrDA, RFiD™ y el estándar IEEE 802.11 entre otras, ofrecen comunicación inalámbrica para diferentes situaciones y propósitos. Bluetooth (estándar 802.15.1), se encuentra dentro de este grupo de tecnologías emergentes y atractivas que ayudan a hacer la comunicación entre dispositivos, mucho más dinámica y sencilla, especializándose en comunicación entre dispositivos cercanos en las denominadas redes de área personal (PAN) .

Si a todo esto añadimos que el precio de estas tecnologías es cada vez mas asequible al ciudadano de a pié, son aún mucho más atractivas si cabe, por lo que cada día cuentan con un mayor número de simpatizantes. Algunos estudios indican ya que dentro de los próximos 10 años el 95% de las comunicaciones entre dispositivos electrónicos de todo tipo serán inalámbricas.

Con esta perspectiva, en este proyecto se ha querido hacer uso de una de estas tecnologías, una de las que está tomando mayor popularidad entre la gente por el amplísimo mercado de productos a los que se puede incorporar, y de hecho ya se está incorporando, Bluetooth.

El objetivo principal ha sido diseñar una aplicación que permita el intercambio de ficheros entre una serie de dispositivos, tomando el estándar Bluetooh como soporte de las comunicaciones. Más concretamente, se ha utilizado la pila de protocolos oficial para sistemas Linux que implementa el estándar, BlueZ. La aplicación se cimienta en los conceptos de red ubicua y de peer-to-peer (p2p) o comunicación de igual a igual, lo que implica que todos los dispositivos que intervienen en los intercambios de información son considerados a efectos de comunicación iguales. Como lenguaje de programación se ha utilizado Python, un lenguaje de muy alto nivel con cada día un mayor número de seguidores debido a su gran versatilidad, su naturaleza interpretada, su soporte para orientación a objetos y otra serie de características que lo hacen muy atractivo.

A continuación se describe brevemente como está organizado el documento indicando los aspectos a que se tratan en cada capítulo.

El primer capitulo ofrece introducciones a conceptos ya mencionados como redes ubicuas, aplicaciones peer-to-peer, redes PAN y el lenguaje de programación Python. Estos elementos son los pilares de la aplicación y conviene conocerlos para poder tener una mayor comprensión de la misma.

Para el segundo capítulo se ha dejado el estándar Bluetooth por requerir una mayor profundidad de contenidos. En el se detalla el origen de la tecnología , las partes que la componen, su especificación técnica, su robustez, los distintos entornos de software que proporcionan acceso a la misma , etc..

El capitulo tercero describe las tres fases que han sido necesarias para llevar a cabo el desarrollo del proyecto. Se centra principalmente en las dos primeras, que son, digamos los requerimientos previos de soporte de la tecnología Bluetooth antes de poder comenzar a diseñar la aplicación en sí.

En el cuarto capitulo se detalla en profundidad la aplicación, describiendo su arquitectura, el esquema de funcionamiento y la forma en que interactúan los distintos elementos que la componen.

En el quinto capítulo se describen todos los requerimientos software que necesita la aplicación para funcionar correctamente especificando en cada caso las versiones de los mismos. También se indica la forma de instalar los dos módulos previos que necesita la aplicación para poder ser ejecutada sin ningún tipo de error.

El capitulo seis describe las pruebas realizadas para comprobar el funcionamiento de la aplicación. En él, se detallan los escenarios escogidos para realizar las pruebas y se muestran algunas estadísticas de los resultados de las mismas.

Finalmente, en el capitulo siete se detallan las conclusiones obtenidas tras la realización del proyecto y se apuntan aspectos que se han podido quedar en el tintero, que en todo caso quedarían pendientes para posibles mejoras o ampliaciones.

Índice general

Índice de figuras

1. Introducción

1.1. Redes ubicuas

Hoy en día es perfectamente normal hablar de redes en las que conviven dispositivos que no están totalmente conectados a Internet, tales como teléfonos móviles, PDA's (Personal Digital Assistant), sistemas de navegación para vehículos, consolas de videojuegos, televisión digital, etc. En definitiva toda una clase de dispositivos heterogéneos cada día más comunes entre las personas.

Las tendencias actuales en investigación indican que todos estos dispositivos se conectarán en muy pocos años a redes con ancho de banda muy superior a los actuales. Al mismo tiempo estas redes tendrán acceso multi-modal (IPv6, xDSL, CATV, Wi-fi, fibra óptica, Bluetooth, etc ...), con lo cual la se multiplicará la conectividad de los dispositivos. El nuevo paradigma de la tecnología de la información bajo este entorno es lo que actualmente se ha dado en llamar ***redes ubicuas*.**

La palabra "Ubicuo" define la cualidad de "*existir en cualquier lugar simultáneamente*". Las redes ubicuas permitirán a los usuarios acceder a Internet desde cualquier sitio y en cualquier momento, esta característica introduce una nueva serie de problemas en el uso de las telecomunicaciones, la multidifusión y la efectividad de los modelos de negocio actuales. Y esto es solo el paso intermedio a las futuras redes en las que se englobarán una serie de productos que en muchos casos todavía están por definir, redes de habitaciones, de ropa inteligente, de periódicos electrónicos, e incluso elementos de bio-tecnología, todo al servicio de una mayor y más ordenada información.

Las redes ubicuas son por tanto el último eslabón en la secuencia de crecimiento de los entornos distribuidos. Desde la computación en redes de área local, en las que se seguía un modelo cliente/servidor, se ha pasado a la computación Web o computación en Internet, en las que se siguen modelos de n capas, y nuevas arquitecturas como B2B, C2B y P2P.

Los dispositivos actualmente mas usados en este tipo de redes son las PDA's, las Tablet PC's y los teléfonos móviles. Estos dispositivos suelen tener acceso a la red mediante Bluetooth, IEEE 802.11b y GPRS. Aunque hoy en día son dispositivos muy usados, aún no están completamente integrados en la red. En la practica, los servicios proporcionados son a menudo poco mas que acceso al correo electrónico y navegación Web limitada, aunque esta situación esta mejorando.

1.2. Peer-to-peer (P2P)

¿Quién no se ha descargado hoy en día una canción o un video desde Internet? Esta es una actividad cada vez más común entre los usuarios de la red. Pues bien, las aplicaciones que hacen posible este tipo de hechos son las aplicaciones *Peer-to-peer* o *P2P*.

El modelo clásico cliente-servidor está basado en una clara distinción de actuaciones a seguir entre los nodos denominados *cliente* y los denominados *servidor*. Los nodos servidor ofrecen servicios y recursos pero no pueden tomar la iniciativa en la comunicación. Por el contrario, los nodos cliente son los que toman la iniciativa, acceden y usan los servicios que ofrecen los servidores. Los clientes pueden comunicarse con los servidores pero no pueden comunicarse directamente con otros clientes, y los servidores no pueden comunicarse con los clientes hasta que estos establecen la comunicación. Por esta razón, los clientes deben conocer de la existencia de los servidores pero no necesitan saber nada de otros clientes.

En cambio, en el modelo peer-to-peer (P2P), la principal característica es la ausencia de un servidor o punto central de control en sus versiones más radicales, los nodos son tratados de igual a igual y no asumen ningún tipo de rol en concreto, es decir, en un momento dado un nodo puede actuar como cliente o bien como servidor, esto implica la desaparición del concepto de jerarquía y control centralizado. Cada nodo puede iniciar la comunicación con otro nodo, pedir un servicio o ser requerido para un servicio. En el modelo P2P es necesario que los nodos tengan mecanismos para conocer la existencia de otros nodos y de los servicios que ofrecen. Estos mecanismos serán diferentes en función de la arquitectura concreta de P2P que se lleve a cabo.

Una ventaja de este modelo es la potencia de procesamiento que se puede conseguir mediante la interacción en proporciones muy pequeñas de un gran número de participantes, obteniendo de este modo grandes capacidades de cálculo.

También en el terreno de la cultura y del ocio, estas tecnologías adquieren gran importancia debido a que aportan facilidad en la obtención de archivos como son los archivos que contienen canciones (MP3s) o películas (AVIs) gracias a la colaboración entre multitud de usuarios y ofrecen la posibilidad de comunicación mediante mensajes instantáneos que ayudan al crecimiento de este submundo virtual que es Internet. Pero precisamente en este terreno es donde estas aplicaciones encuentran su mayor problema y es la constante situación en la frontera de la legalidad y moralidad por el hecho de poner en jaque a industrias como la musical o cinematográfica.

1.2.1. Arquitecturas

Se pueden distinguir cuatro arquitecturas según el diseño que se haya aplicado para construir la red peer-to-peer dependiendo de la presencia de servidores, la forma de búsqueda de contenidos y nodos activos, etc.

Modelo P2P puro

La principal característica de esta implementación es la ausencia de un servidor central y el tratamiento entre los nodos de igual a igual. De esta manera, los nodos pertenecientes a una red de este tipo pueden cumplir dos funciones: la de servidor, cuando otro nodo requiere información de éste y por lo tanto, se la ofrece, y la de cliente, cuando éste solicita información a otro nodo o peer. Este modelo permite el funcionamiento de la red a pesar de las caídas de sus nodos, gran problema en las aplicaciones pertenecientes a la arquitectura cliente-servidor, donde en el momento de la caída del servidor, la red aplicación queda inutilizada. Su esquema se muestra en la Figura 1-1.

Para realizar la búsqueda de nodos activos, se puede utilizar una lista de nodos conocidos o simplemente enviando un mensaje multicast o broadcast a la red.

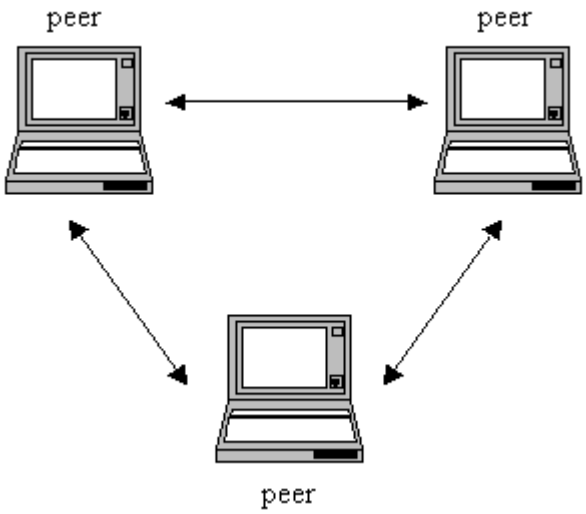


Figura 1-1 : Modelo peer-to-peer puro

P2P con servicio de consulta de nodos

Este modelo implementa una arquitectura con la presencia de un servidor central para indicar los nodos que se encuentran activos en la red. Por lo tanto, un nodo realiza la consulta al servidor para conocer los nodos que se encuentran activos en un momento dado. Posteriormente, el nodo podrá establecer una conexión directa con el peer elegido para compartir sus recursos.

La aplicación P2P en el nodo debe informar al servidor de su conexión y desconexión para mantener la integridad. Este servidor no se corresponde con el servidor de la arquitectura cliente-servidor sino que es un nodo que simplemente presta el servicio de consulta de nodos. La Figura 1-2 muestra el esquema de este modelo.

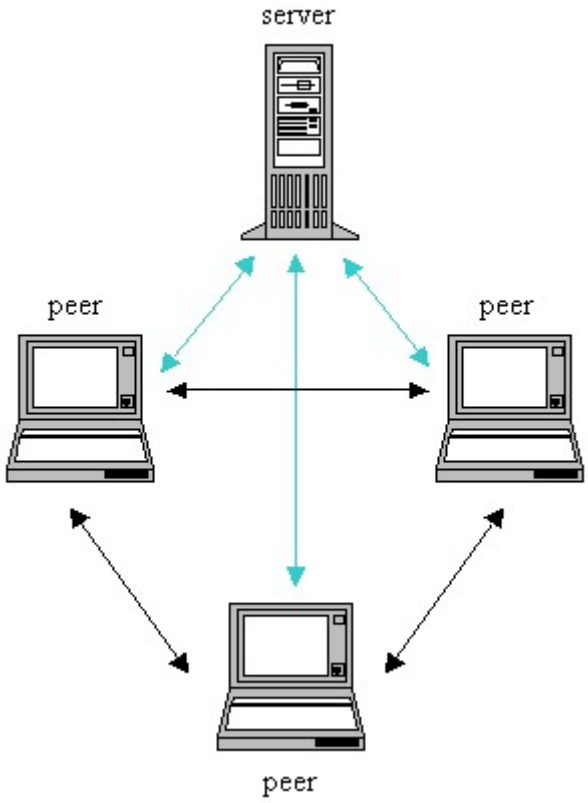


Figura 1-2 : Modelo peer-to-peer con servicio de consulta de nodos

P2P con servicio de consulta de nodos y recursos

La implementación de esta arquitectura es muy similar a la anterior, pero el servidor no sólo se encarga de almacenar la información de los nodos que permanecen activos sino de los contenidos que éstos presentan y comparten.

P2P con servicio de consulta de nodos, recursos y fuente de contenidos

Este modelo se basa en las dos implementaciones anteriores, es decir, un número de nodos que piden una cierta información al servidor. El servidor de este modelo cumple dos misiones: la primera es almacenar los nodos que se encuentran activos y la información que estos comparten. La segunda, almacenar los contenidos para compartir con los nodos conectados. Por tanto, ésta es la implementación de P2P menos pura y más parecida a la arquitectura tradicional de cliente-servidor. El esquema de este modelo se muestra en la Figura 1-3.

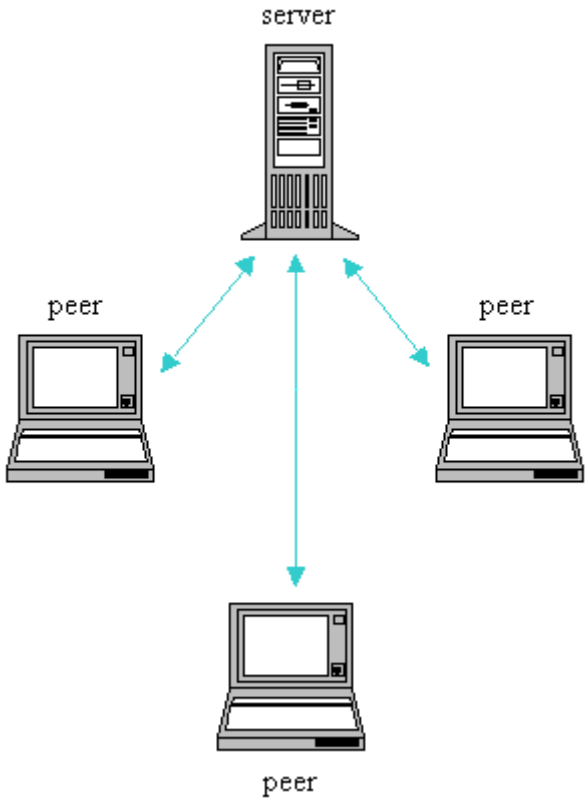


Figura 1-3 : Modelo peer-to-peer con servicio de consulta de nodos, recursos y fuentes de contenidos

1.2.2 Tipos de aplicaciones

P2P para archivos compartidos

La aplicación pionera para compartir archivos fue probablemente Napster, que permitía al usuario acceder a canciones que estaban almacenadas en los discos duros de otros usuarios. Napster mantenía una base de datos centralizada que contenía los títulos y la ubicación de las canciones, pero no las almacenaba físicamente. Cuando un usuario realizaba una búsqueda de un título en especial, el sistema mostraba una lista de opciones de donde podría, en base a su criterio personal, obtener la melodía requerida. Pero Napster desapareció por problemas legales con las compañías propietarias de los derechos de las canciones. En la actualidad programas como Kazaa o eMule (entre otros) son sus sucesores.

Otros sistemas P2P no tienen la función centralizadora como los anteriores, sino que sólo tienen el sitio web donde el usuario puede obtener el software P2P y registrase. Este es el caso de *Gnutella* y *free.net*, donde el usuario hace los papeles de cliente y servidor almacenando tanto los archivos a compartir como una lista de nodos según se hagan necesarios.

P2P para computación distribuida

Esta modalidad de P2P se basa en la premisa de que la mayoría de las computadoras de los miembros están gran parte del tiempo en espera de comandos o acciones por tomar para proceder a procesar. Durante este tiempo la capacidad de procesamiento está ociosa. Estos ciclos de computación ociosos pueden ser captados con un fin central de procesamiento distribuido. Las capacidades que se obtienen a través de esta modalidad de P2P son asombrosas.

Como ejemplo, la Universidad de Berkeley creó un proyecto de esta índole (llamado SETI@home) para aliviar las capacidades de procesamiento del Instituto SETI (Search for Extraterrestrial Intelligence). Para participar en este proyecto, el usuario descarga de Internet un protector de pantalla que simplemente indica el momento cuando la computadora esta ociosa. Una vez que se activa este protector, el SETI le asigna a esa máquina el procesamiento de un trozo de las bitácoras de datos telemétricos de los radio-telescopios. Con esto el SETI ha logrado procesar 14 años de datos almacenados en sólo dos años con la ayuda de 450.000 usuarios del P2P que administra Berkeley.

P2P para colaboración

Esta modalidad busca organizar espacios de trabajo en línea y compartir proyectos para grupos de trabajo. *Groove Networks* comenzó el desarrollo de este tipo de aplicaciones en los noventa y su intención actual son los clientes inalámbricos. El adaptar P2P a esta tecnología requiere que los fabricantes de tecnología inalámbrica como Bluetooth desarrollen técnicas para la detección de los otros clientes o servidores.

P2P para voz

Una de las últimas aplicaciones en las que ha hecho aparición y de una manera muy fuerte la tecnología ha sido en la voz. Ejemplo de ello es el software Skype desarrollado por el equipo que desarrolló Kazaa. Éste permite el uso del ordenador para realizar llamadas de voz a nuestros contactos que también posean el software, con una calidad que incluso supera al de las líneas tradicionales y que está muy por encima del que ofrecen softwares gratuitos que también permiten la utilización de la tecnología IP para el uso de la voz.

Otros

Finalmente, también la comunidad universitaria está apostando en cierta medida por esta tecnología para resolver diversos retos que requieren una solución tecnológica. Proyectos como Edutella están orientados al ámbito de la tele-enseñanza y pretende crear una infraestructura para gestionar datos sobre redes P2P. Edutella es una iniciativa que se lleva a cabo entre las universidades alemanas de Hannover, Braunschweig y Karlsruhe, las universidades suecas de Estocolmo y Uppsala y la universidad de Stanford entre otras.

1.3. Redes WPAN

1.3.1. Justificación

Las tecnologías de redes inalámbricas locales y personales requieren un extenso trabajo de investigación dado que son tecnologías en desarrollo y en constante cambio. Las primeras publicaciones en estándares oficiales IEEE se llevaron a cabo en abril del 2003 y están sujetas a revisión dentro los próximos cinco años. En pocos años este tipo de redes tendrán repercusiones tecnológicas y económicas muy importantes debido a su alta eficiencia y a los objetos hacia los que están orientadas. Muchas de sus aplicaciones aun no se vislumbran, por lo que el estudio profundo de las tecnologías es fundamental para el desarrollo de las mismas.

1.3.2. Definición de WPAN

Los aparatos electrónicos personales cada día se vuelven más inteligentes e interactivos. Muchos de ellos han incrementado sus capacidades de datos, por ejemplo, las computadoras tipo notebook, teléfonos celulares, agendas electrónicas (PDAs), reproductores de música personales, cámaras digitales, etc.

Dichas capacidades les permiten retener, usar, procesar y comunicarse con varios tipos de información. Por ejemplo, muchos de estos aparatos tienen una base de datos de información personal (PIM) con la que manejan agendas personales, libros de direcciones y realizan todo tipo de listas personales. Las bases de datos PIM en un dispositivo personal deben de estar sincronizadas con otras bases de datos PIM de otros dispositivos. La solución obvia para la sincronización de dichos elementos es interconectarlos y sincronizarlos.

Tradicionalmente se han utilizado cables de propósito específico para interconectar aparatos personales. Sin embargo muchos usuarios encuentran que todos estos cables resultan en una tarea un tanto frustrante e improductiva. Además los cables se pueden perder, dañar e incrementan innecesariamente el peso y volumen de los aparatos. Por lo que se vuelve indispensable el desarrollo de soluciones para la interconexión de aparatos de forma inalámbrica. Es importante que estas no tengan un impacto importante en cuanto a la forma original, peso, requerimientos de energía, costes, facilidad de uso, etc.

De aquí nace la necesidad de crear una forma eficiente, rápida y confiable de hacer transiciones de información de forma inalámbrica. Una solución se conoce como redes inalámbricas de área local o WPAN por sus siglas en ingles (Wireless Personal Area Network). Una WPAN es "**una red inlámbrica de área personal**", sin embargo esta definición no nos dice mucho, para profundizar más en este aspecto es importante tener en cuenta los siguientes aspectos.

La característica principal de este tipo de redes es que enfocan sus sistemas de comunicaciones a un área típica de 10 metros a la redonda que envuelve a una persona o a

algún dispositivo esté en movimiento o no. "A diferencia de las redes de área local (WLAN), una conexión hecha a través de una WPAN involucra a muy poca o nula infraestructura o conexiones directas hacia el mundo exterior". Este tipo de tecnología también procura hacer un uso eficiente de recursos, por lo que se han diseñado protocolos simples y lo más óptimos para cada necesidad de comunicación y aplicación.

Interconectar dispositivos personales es diferente a conectar dispositivos computacionales. Las soluciones típicas de conexión para dispositivos computacionales (por ejemplo, una conexión WLAN para una computadora notebook) asocian al usuario del dispositivo con servicios de datos disponibles. Dicha situación contrasta con la naturaleza personal e íntima de una conexión inalámbrica para los dispositivos asociados a usuarios particulares. El usuario es relacionado con los dispositivos electrónicos de su posesión, o en su vecindad en vez de a un lugar geométrico en particular o en alguna localidad de la red.

El termino red de área personal (PAN) se concibió para describir estos diferentes tipos de conexión en red. La versión desconectada o desatada de dicho concepto es el concepto de WPAN. Una WPAN puede verse como una burbuja personal de comunicación alrededor de una persona. Dentro de dicha burbuja, que se mueve en la misma forma en que lo hace una persona, los dispositivos personales se pueden conectar entre ellos.

Para satisfacer las diferentes necesidades de comunicación dentro de un área personal la IEEE ha dividido sus esfuerzos en grupos de trabajo, que se encargan de desarrollar estándares.

1.3.2.1. Grupos de trabajo

Existen principalmente cuatro grupos de trabajo para la tecnología WPAN, cada uno de ellos con características e intereses específicos que generan estándares que satisfacen necesidades específicas de comunicación.

- El grupo de trabajo **802.15.1** ha realizado un estándar basado en las especificaciones de la fundación Bluetooth. Este grupo de trabajo publicó el estándar IEEE 802.15.1 el 14 junio de 2002.

- El grupo de trabajo **802.15.2** está desarrollando un modelo de coexistencia entre las WLAN y WPAN, así como de los aparatos que las envuelven.

- El grupo de trabajo **802.15.3** está trabajando para establecer los estatus y publicar un estándar nuevo (publicado en junio de 2003) de alta velocidad (20 Mbits/s o mayores) para WPANs. Además de ofrecer una alta velocidad de transmisión, este nuevo estándar se está diseñando para consumir poca energía y ofrecer soluciones a bajos costos así como aplicaciones multimedia.

- El grupo de trabajo T4 para el desarrollo IEEE **802.15.4**, investiga y desarrolla soluciones que requieren una baja transmisión de datos y con ello una duración en las baterías de meses e incluso de años así como una complejidad relativamente

baja. Dicho grupo de trabajo ha publicado el estándar que lleva su nombre; IEEE 802.15.4.

1.3.2.2. Diferencia entre WLANs y WPANs

A simple vista la operación así como los objetivos de una WPAN parecen ser los de una WLAN, definidos en el estándar IEEE 802.11. Ambas tecnologías permiten a un dispositivo electrónico conectarse con el ambiente que lo rodea e intercambiar datos sobre canales inalámbricos libres o frecuencias que no necesitan licencia de uso. Sin embargo, WLANs se han diseñado y optimizado para dispositivos transportables de comunicación, por ejemplo las computadoras tipo notebook. Las WPAN fueron diseñadas para dispositivos con mayor movilidad.

Las dos tecnologías difieren principalmente en tres puntos fundamentales:

- Niveles de energía y cobertura.
- Control de medios.
- Configuraciones de red.

1.3.2.3. Aplicaciones

El IEEE 802.15 se diseña para ser ocupado en una amplia gama de aplicaciones, incluyendo el control y monitoreo industrial; seguridad pública, como la detección y determinación de la localización de personas en lugares de desastres; medición en automóviles, como el monitoreo de la presión neumática en las llantas; tarjetas o placas inteligentes; y agricultura de precisión, como medición del nivel de humedad en el suelo, pesticida, herbicida, niveles de pH. Sin embargo las mayores oportunidades de desarrollo del IEEE 802.15 están en la automatización del hogar.

En una casa, se pueden considerar varias posibilidades de mercado: periféricos para el PC, tales como ratones inalámbricos, teclados, joysticks, agendas electrónicas (PDAs) y juegos; aparatos electrónicos, como radios, televisiones, VCRs, CDs, DVDs, controles remotos, y demás, y un control universal para controlar todos los anteriores; automatización del hogar, como calefacción, ventilación, aire acondicionado, iluminación, seguridad y el control de objetos como ventanas, cortinas, puertas, y cerraduras; monitoreo de salud, incluyendo censores, monitores y diagnósticos; así como juguetes y juegos, juegos interactivos entre personas o grupos.

Se espera que los requerimientos máximos de transmisión de datos para aplicaciones con periféricos de PC estén en el rango de los 115.2 kb/s a menos de 10kb/s para automatización de tareas del hogar y para algunos dispositivos electrónicos. De la misma manera se espera que los periféricos de PC acepte un rango de aproximado de 15 ms y de más de 100 m para aplicaciones de automatización del hogar.

1.4 Python

Fue en el Stichting Mathematisch Centrum de Holanda, en 1991, cuando nació la primera versión de Python como sucesor del lenguaje ABC. El creador del lenguaje es Guido Van Rossum y actualmente sigue ocupándose de él junto a un buen número de colaboradores de todo el mundo.

Python es un lenguaje de programación muy potente y de fácil aprendizaje. Tiene eficaces estructuras de datos de muy alto nivel y una solución de programación orientada a objetos simple pero eficaz. Su elegante sintaxis , su gestión dinámica de tipos y su naturaleza interpretada hacen de él un lenguaje de scripts ideal para el desarrollo rápido de aplicaciones en muchas áreas.

El lenguaje comenzó a desarrollarse para ordenadores Mac aunque actualmente es en sistemas Linux donde se lleva todo el peso de la programación. El uso en Linux de Python está mucho más extendido que en otras plataformas aunque es compatible con la mayoría de ellas, incluyendo Windows.

Además, permite mantener de forma sencilla interacción con el sistema operativo, y resulta muy adecuado para manipular archivos de texto. Esta característica hace que muchas distribuciones de GNU/Linux utilicen Python para sus herramientas de configuración (valga como ejemplo el programa instalador de Red Hat *anaconda*). También es ampliamente utilizado en la web.

El intérprete de Python y la extensa biblioteca estándar están disponible libremente en forma de fuentes o ejecutables, para las plataformas más importantes en la web oficial de Python (http://www.python.org/), y se pueden distribuir libremente. La misma sede contiene también distribuciones y direcciones de muchos módulos, programas y herramientas Python de terceras partes, además de documentación adicional.
Existe una web más informal en http://starship.python.net/ que contiene unas cuantas páginas personales relativas a Python. Mucha gente tiene software descargable en estas páginas.

El intérprete funciona como el intérprete de órdenes de UNIX: cuando se le llama con la entrada estándar conectada a un dispositivo tty, lee y ejecuta las órdenes interactivamente; cuando se le da un nombre de fichero como argumento o se le da un fichero como entrada estándar, lee y ejecuta un script desde ese fichero.

La distribución estándar de Python incluye mucho código en C y en Python. Existen módulos para leer buzones UNIX, recuperar documentos por HTTP, generar números aleatorios, extraer opciones de una línea de órdenes, escribir programas CGI, comprimir datos, crear sockets, etc ...

Es fácil ampliar el lenguaje con nuevas funciones y tipos de datos implementados en C y C++ (u otros lenguajes a los que se pueda acceder desde C). Python es también adecuado como lenguaje de extensión para aplicaciones adaptables al usuario.

Todas las versiones de Python son Open Source (http://www.opensource.org/) . Históricamente, la mayoría de las versiones de Python han sido a su vez compatibles con GPL (The GNU General Public License), lo que significa que se puede modificar y distribuir libremente su código.

Todas las razones mencionadas hacen de Python un lenguaje muy atractivo para empezar a usarlo, por eso se ha escogido para desarrollar la aplicación.

2. El estándar Bluetooth (802.15.1)

2.1. Antecedentes

Hablar de Bluetooth es hablar de unificación. Traducido al castellano, Bluetooth significa "Diente Azul", el apodo de un rey danés llamado *Harald Blaatand* que, en el siglo X, logró unir políticamente bajo su reinado a Dinamarca y Noruega, ambas separadas por el mar. Al igual que este rey, la tecnología inalámbrica Bluetooth une y conecta distintos dispositivos entre sí mediante enlaces de radio universal de corto alcance, con capacidad de crear pequeñas radio *LANs*, solventando uno de los problemas que la vida digital provoca : la masiva proliferación de cables.

Bluetooth se creó en los laboratorios Ericsson para aplicaciones tales como sustituir cables en equipos de comunicación. Se decidió que cualquier fabricante interesado debería poder acceder libremente a las especificaciones de Bluetooth, y se fueron sumando otras compañías. En Febrero de 1998, se fundó el *Bluetooth* ***Special Interest Group*** *(SIG)*, creado con el fin de ofrecer soporte para la nueva tecnología. Actualmente, más de mil compañías lo integran y trabajan conjuntamente por un estándar abierto para el concepto *Bluetooth* .

Bluetooth no sólo está pensado para conectar periféricos a los ordenadores. Sus posibilidades son amplias en todos aquellos campos en los que las distancias sean cortas. Por ejemplo ya se está aplicando a la telefonía móvil, el sector automovilístico, etc ...

En la actualidad se calcula que existen cerca de 200 millones de dispositivos Bluetooth en el mundo, con un alto ritmo de crecimiento. Según Gartner , una firma de investigación de mercados, para finales de 2004 serán más de 350 millones.

2.2. Especificación técnica

Bluetooth es un sistema de radio que opera en la banda de frecuencia libre de *2.45 GHz* (*Figura 2-1*), esta banda de frecuencia está disponible en la mayor parte del mundo.

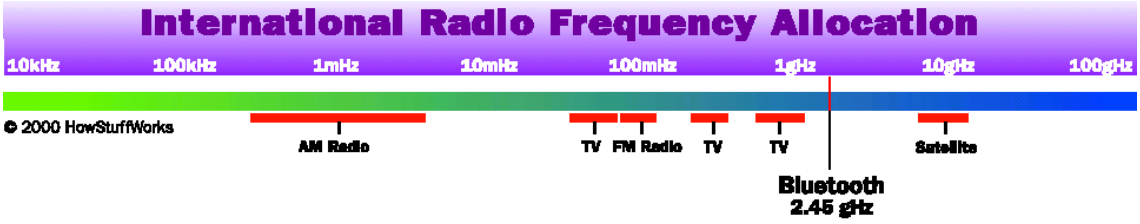


Figura 2-1 : Ubicación de la banda dentro del espectro electromagnético

Se utilizan 79 canales (aunque en España, Francia y Japón se usan 23) de radio frecuencia con un ancho de banda de 1 MHz cada uno y un ratio máximo de símbolos de 1 MSímbolo/s. Después de que cada paquete es enviado en una determinada frecuencia de transmisión, ésta cambia a otra de las 79 (en su caso 23) frecuencias. El rango típico de operación de *Bluetooth* es de unos 10 m (distancia más que suficiente para interconectar dispositivos domésticos entre sí), sin embargo se pueden alcanzar distancias de hasta 100 m con el uso de amplificadores.

Como se puede observar en la *Figura 2-2* la comunicación sobre *Bluetooth* se divide en varias capas. Los recuadros sombreados representan las capas definidas en la especificación. La capa *RFCOMM* se trata como caso aparte, ya que, aunque no es estrictamente necesaria, se ha introducido para añadir la funcionalidad de emulación de puertos serie. Está basada en la norma ETSI TS 07.10.

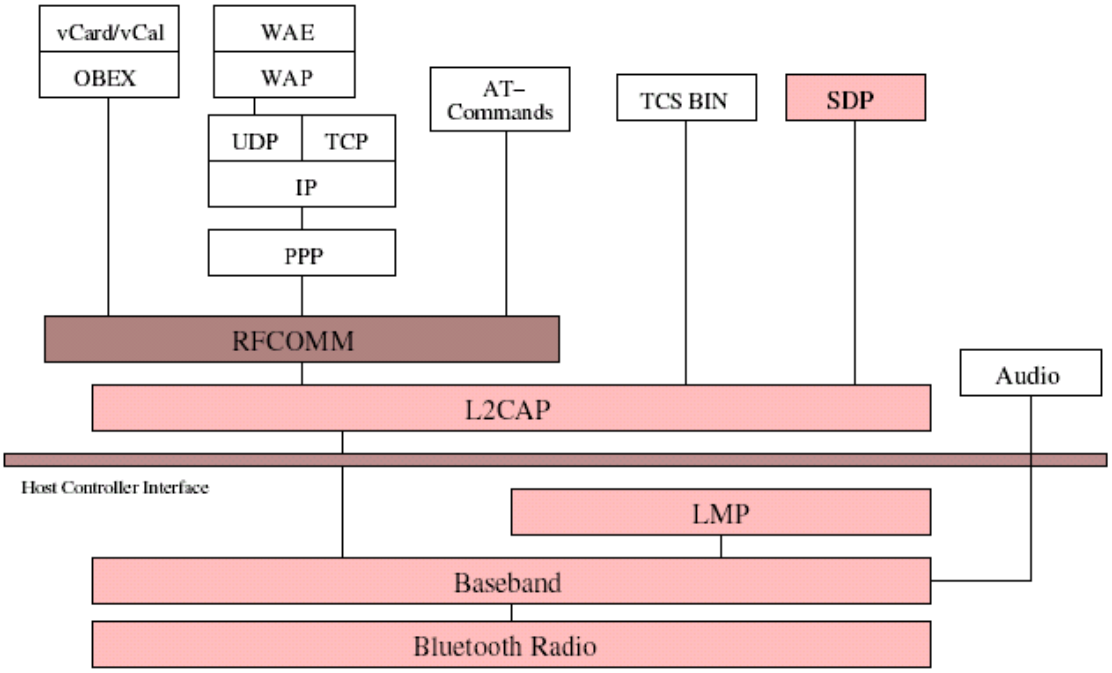


Figura 2-2 : Pila de protocolos Bluetooth

A continuación se presenta una breve descripción de algunas de ellas. Se entrará en mayor detalle en sub-capítulos posteriores.

La capa de comunicación más baja es llamada **Banda Base**. Esta capa implementa el canal físico real. Emplea una secuencia aleatoria de saltos a través de 79 frecuencias de radio diferentes. Los paquetes son enviados sobre el canal físico, donde cada uno es enviado en una frecuencia de salto diferente. La *Banda Base* 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. Soporta dos tipos de enlace: *síncrono orientado a la conexión* (**SCO**), para datos, y *asíncrono no orientado a la conexión* (**ACL**), principalmente para audio. Los dos pueden ser multiplexados para usar el mismo enlace de radiofrecuencia (*RF)*. Usando ancho de banda reservado, los enlaces *SCO* soportan tráfico de voz en tiempo real.

El **Link Manager Protocol (LMP)** o *Protocolo de Gestión de Enlace* es el responsable de la autenticación, encriptación, control y configuración del enlace. El *LMP* también se encarga del manejo de los modos y consumo de potencia, además soporta los procedimientos necesarios para establecer un enlace *SCO*.

El **Host Controller Interface (HCI)** o *Interfaz del Controlador del Host* brinda un método de interfaz uniforme para acceder a los recursos de hardware de *Bluetooth*. Éste contiene una interfaz de comando para el *controlador banda base* y la *gestión de enlace*, además de para acceder al hardware.

El Logical Link Control and Adaptation Protocol (L2CAP) o *Protocolo de Control y Adaptación de Enlace Lógico*, corresponde a la capa de enlace de datos. Ésta brinda servicios de datos orientados y no orientados a la conexión a capas superiores. *L2CAP* multiplexa los protocolos de capas superiores con el fin de enviar varios protocolos sobre un canal banda base. Con el fin de manipular paquetes de capas superiores más grandes que el máximo tamaño del paquete banda base, *L2CAP* los segmenta en varios paquetes banda base. La capa *L2CAP* del receptor reensambla los paquetes banda base en paquetes más grandes para la capa superior. La conexión L2CAP permite el intercambio de información referente a la calidad de la conexión, además maneja grupos, de tal manera que varios dispositivos pueden comunicarse entre sí.

El **Sevice Discovery Protocol (SDP)** o *Protocolo de Descubrimiento de Servicio* define cómo actúa una aplicación de un cliente *Bluetooth* para descubrir servicios disponibles de servidores *Bluetooth*, además de proporcionar un método para determinar las características de dichos servicios.

El protocolo **RFCOMM** ofrece emulación de puertos serie sobre el protocolo *L2CAP*. *RFCOMM* emula señales de control y datos *RS-232* sobre la banda base *Bluetooth.* Éste ofrece capacidades de transporte a servicios de capas superiores (por ejemplo *OBEX*) que usan un canal serie como mecanismo de transporte. *RFCOMM* soporta dos tipos de comunicación, directa entre dispositivos actuando como *endpoints,* y *dispositivo-modem-dispositivo*. Además tiene un esquema para emulación de *null modem***.**

El **control de telefonía binario (TCS binario)** es un protocolo que define la señalización de control de llamadas, para el establecimiento y liberación de una conversación o una llamada de datos entre unidades *Bluetooth.* Además, éste ofrece funcionalidad para intercambiar información de señalización no relacionada con el progreso de llamadas.

La capa de **Audio** es una capa especial, usada sólo para enviar audio sobre *Bluetooth.* Las transmisiones de audio pueden ser ejecutadas entre una o más unidades usando muchos modelos diferentes. Los datos de audio no pasan a través de la capa *L2CAP,* sino

directamente a la *Banda Base*, después de abrir un enlace y realizar un establecimiento entre dos unidades *Bluetooth.*

Protocolos Específicos

- **Control de telefonía – Comandos AT.** *Bluetooth* soporta un número de comandos *AT* para el control de telefonía a través de emulación de puerto serie (*RFCOMM*).

- **Protocolo Punto-a-Punto (PPP).** El *PPP* es un protocolo orientado a paquetes y por lo tanto debe usar su mecanismo serie para convertir un torrente de paquetes de datos en una corriente de datos serie. Este protocolo corre sobre RFCOMM para lograr las conexiones punto-a-punto.

- **Protocolos UDP/TCP-IP.** Los estándares *UDP/TCP-IP* permiten a las unidades *Bluetooth* conectarse, por ejemplo a *Internet*, a través de otras unidades conectadas. Por lo tanto, la unidad puede actuar como un puente para *Internet*. La configuración *TCP/IP/PPP* está disponible como un transporte para *WAP*.

- **Wireless Aplication Protocol (WAP)** o *Protocolo de Aplicación Inalámbrica. E*s una especificación de protocolo inalámbrico que trabaja con una amplia variedad de tecnologías de red inalámbricas conectando dispositivos móviles a *Internet*. *Bluetooth* puede ser usado como portador para ofrecer el transporte de datos entre el cliente *WAP* y su servidor de *WAP* adyacente. Además, las capacidades de red de *Bluetooth* dan a un cliente posibilidades únicas en cuanto a movilidad comparado con otros portadores *WAP*. Un ejemplo de *WAP sobre Bluetooth* sería un almacén que transmite ofertas especiales a un cliente cuando éste entra en el rango de cobertura.

- **Protocolo OBEX.** Es un protocolo opcional de nivel de aplicación diseñado para permitir a las unidades *Bluetooth* soportar comunicación infrarroja para intercambiar una gran variedad de datos y comandos. Éste usa un modelo *cliente-servidor* y es independiente del mecanismo de transporte y del *API (Application Program Interface)* de transporte. *OBEX* usa *RFCOMM* como principal capa de transporte.

La *Figura 2-3* muestra una comparación de la pila de protocolos *Bluetooth* con el modelo de referencia estándar **OSI**. A pesar de que ambos modelos no concuerdan exactamente, la comparación entre ellos es muy útil para relacionar las diferentes partes de cada uno. 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.*

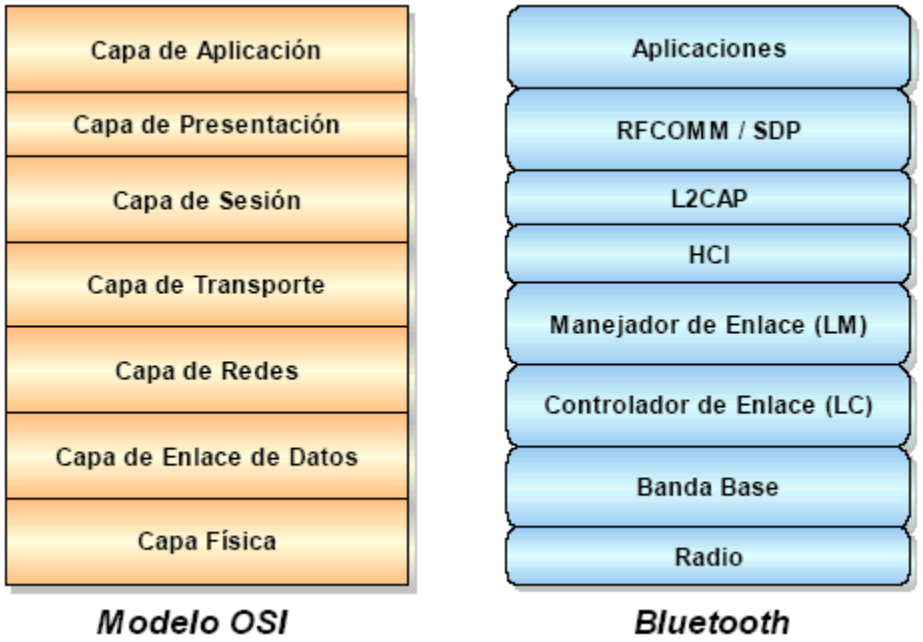


Figura 2-3 : Modelo de referencia OSI y Bluetooth

2.2.1. Banda Base

2.2.1.1. Descripción general

Bluetooth emplea una combinación de conmutación de paquetes y conmutación de circuitos. Soporta un canal de datos asíncrono de hasta tres canales de voz simultáneos. El canal asíncrono soporta comunicación simétrica y asimétrica. En la comunicación asimétrica pueden ser enviados 723.3 kb/s desde el servidor y 57.6 kb/s hacia el servidor, mientras que en la comunicación simétrica pueden ser enviados 433 kb/s en ambas direcciones.

Bluetooth brinda conexión punto-a-punto (*Figura 2-4.a*) o conexión punto-a-multipunto (*Figura 2-4.b*). Dos o más unidades compartiendo el mismo canal forman una **piconet**. Cada piconet debe tener un maestro y puede tener hasta siete esclavos activos. Además pueden haber muchos más esclavos en estado parked (hasta 256). Estos esclavos no están activos en el canal sin embargo están sincronizados con el maestro con el fin de asegurar una rápida iniciación de comunicación.

La interconexión de varias piconets forma una **scatternet** (*Figura 2-4.c*). Para ello, al menos una unidad actúa como maestro en una piconet y como esclavo en otra u otras. Información más detallada puede encontarse en la especificación Bluetooth .

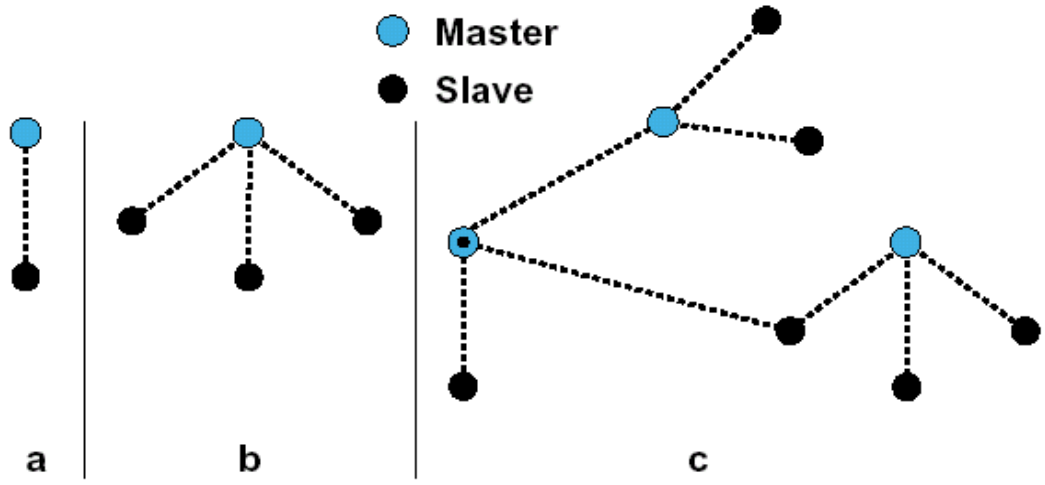


Figura 2-4 : piconets (punto-a-punto (a), punto-multipunto (b)) y *scatternet* (c)

2.2.1.2. Canal físico

El canal físico contiene *79 frecuencias de radio diferentes*, las cuales son accedidas de acuerdo a una secuencia de saltos aleatoria. La rata de saltos estándar es de *1600 saltos/s.* El canal está dividido en **timeslots** (ranuras de tiempo), cada **slot** (ranura) corresponde a una frecuencia de salto y tiene una longitud de 625 µs. Cada secuencia de salto en una *piconet* está determinada por el maestro. Todos los dispositivos conectados a la *piconet* están sincronizados con el canal en salto y tiempo. En una transmisión, cada paquete debe estar alineado con el inicio de un slot y puede tener una duración de hasta cinco timeslots**.** Durante la transmisión de un paquete la frecuencia es fija. Para evitar fallos en la transmisión, el maestro inicia enviando en los timeslots pares y los esclavos en los timeslots impares. En la *Figura 2-5* se puede observar este esquema de transmisión.

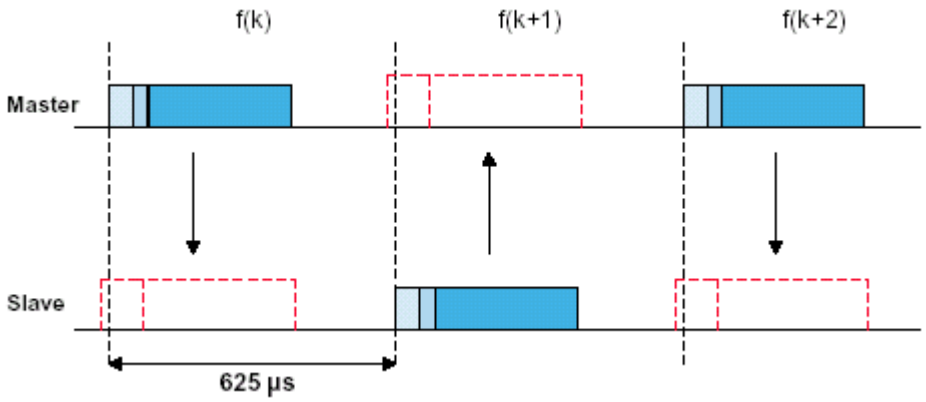


Figura 2-5 : Transmisión en una piconet

Un paquete ocupa normalmente una única ranura (Figura 2-6), pero puede ocupar hasta cinco consecutivas (en la figura 2-7 se muestra el caso de un paquete transmitido en tres ranuras consecutivas).

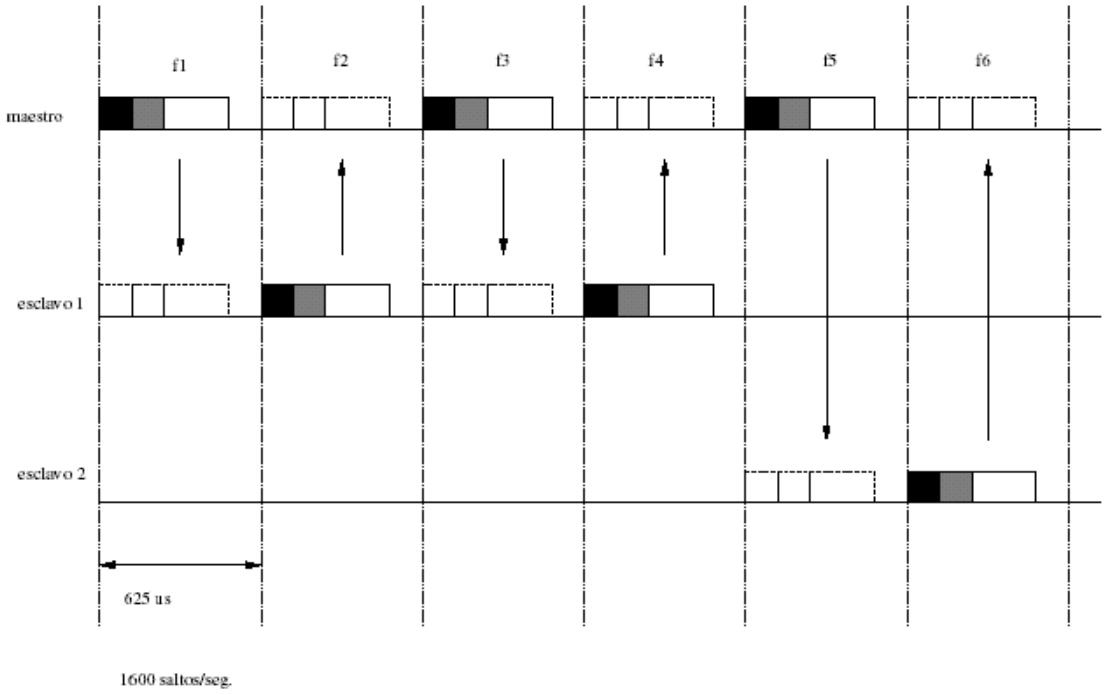


Figura 2-6 : paquetes de una ranura

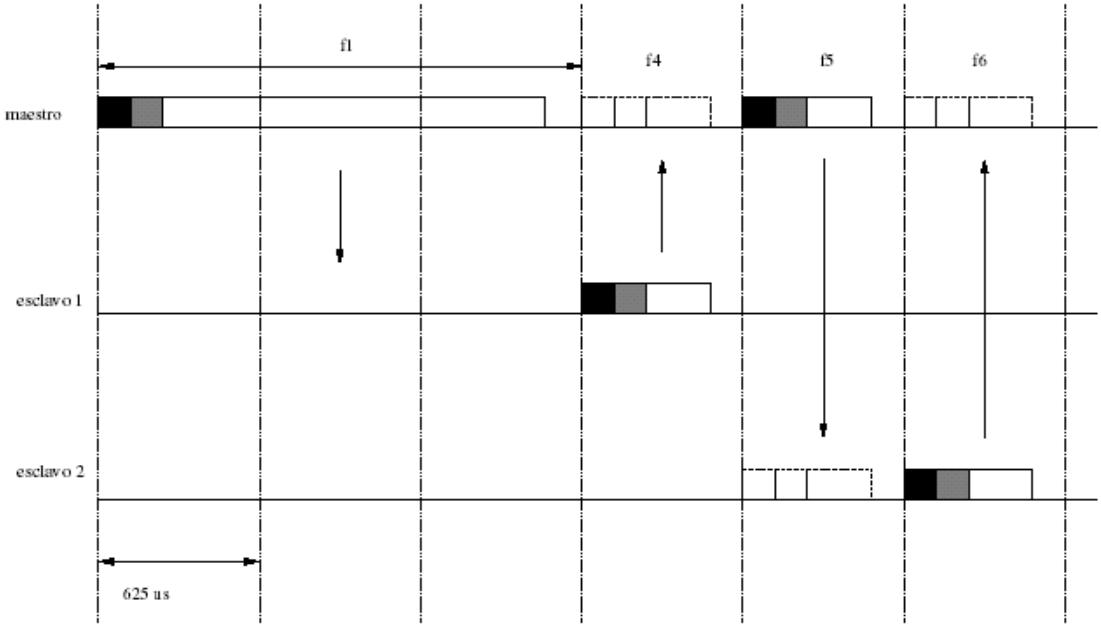


Figura 2-7 : paquetes multi-ranura

2.2.1.3. Enlace físico

La comunicación sobre *Bluetooth* es perfecta para enlaces *SCO* o enlaces *ACL*. El enlace *SCO* es una conexión simétrica *punto-a-punto* entre el maestro y un esclavo específico. Para lograr la comunicación, el enlace *SCO* reserva slots en intervalos regulares en la iniciación, es por esto que el enlace puede ser considerado como una conexión de conmutación de circuitos.

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 del paquete, así todos los esclavos leerán el paquete.

2.2.1.4. Paquetes

Los datos enviados sobre el canal de la *piconet* son convertidos en paquetes, éstos son enviados y el receptor los recibe comenzandopor el *bit* menos significativo. Como se observa en la Figura 2-8, el formato de paquete general consta de tres campos: código de acceso, cabecera y carga útil.

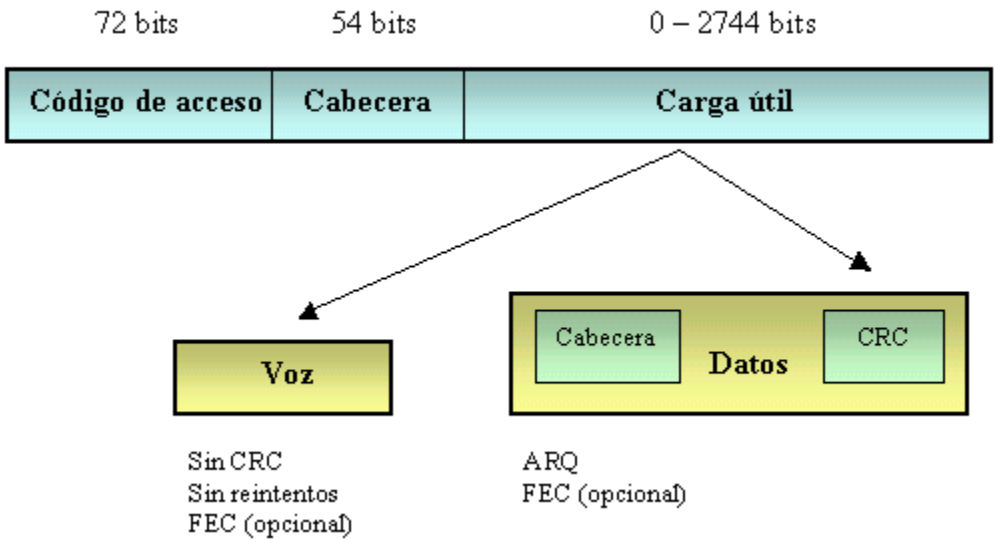


Figura 2-8 : Formato de paquete general

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** - Para identificar los paquetes sobre el canal de la *piconet.*

- **Código de acceso de dispositivo** – 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 rango, o *IAC dedicado* cuando se desea descubrir unidades de un tipo específico.

Cabecera del paquete. Como se observa en la Figura 2-9, la cabecera del paquete consta de seis campos:

- **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 2-9 : 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 código *CRC.*

2.2.1.5. Corrección de errores

En una comunicación *Bluetooth* existen distintos mecanismos de corrección de errores :

- En la cabecera, cada *bit* es repetido tres veces.
- En la carga útil se usa un esquema de *código Hamming*. Los *bits* de información son agrupados en secuencias de 10 *bits*, éstos son enviados como 15 *bits* y el algoritmo corrige todos los errores de un *bit* y detecta los errores de dos *bits*.
- Para garantizar una recepción correcta, todos los paquetes de datos son retransmitidos hasta que el emisor reciba una confirmación. La confirmación es enviada en la cabecera de los paquetes retornados.
- Los paquetes **broadcast** son paquetes transmitidos desde el maestro a todos los esclavos. No hay posibilidad de usar confirmación para esta comunicación, sin embargo, para incrementar la posibilidad de recibir correctamente un paquete, cada *bit* en el paquete es repetido un número fijo de veces.
- El chequeo de redundancia cíclica (*CRC*) se usa para detectar errores en la cabecera. La suma de comprobación *CRC* está contenida en el campo *HEC* de la cabecera de paquete. Los chequeos de redundancia cíclica también se aplican sobre la carga útil en la mayoría de los paquetes.
- Para asegurar que no desaparezcan paquetes completos, *Bluetooth* usa números de secuencia. Actualmente sólo se usa un número de secuencia de un *bit*.

2.2.1.6. Transmisión / Recepción

Como se mencionó antes, el maestro de la *piconet* empieza enviando en timeslots pares y el esclavo en los impares. Solamente el último esclavo direccionado está autorizado para enviar en el timeslot de los esclavos. Esto no causa problemas ya que el maestro siempre está inicializando todas las conexiones y transmisiones nuevas. Cada esclavo espera las oportunidades de conexión dadas por el maestro.

Los paquetes pueden ser más grandes que un timeslot**,** debido a esto el maestro puede continuar enviando en los timeslots impares y viceversa. El sistema de reloj del maestro sincroniza a toda la *piconet.* El maestro nunca ajusta su sistema de reloj durante la existencia de una *piconet,* son los esclavos quienes adaptan sus relojes con un **offset** de tiempo con el fin de igualarse con el reloj del maestro. Este offset es actualizado cada vez que es recibido un paquete desde el maestro.

2.2.1.7. Control de Canal

El control de canal describe cómo se establece el canal de una *piconet* y cómo las unidades pueden ser agregadas o liberadas en la *piconet.* La dirección del maestro determina la secuencia de saltos y el código de acceso al canal. La *fase* de la *piconet* está determinada por el sistema de reloj del maestro. Por definición, la unidad *Bluetooth* que inicia la conexión representa al maestro.

En *Bluetooth*, la capa de control de enlace se divide en dos estados principales: **standby** y **conexión**. Además existen siete sub-estados: **page, page scan, inquiry** (búsqueda)**, inquiry scan,** respuesta de maestro, respuesta de esclavo y respuesta a **inquiry**. Los sub-estados son usados para agregar nuevos esclavos a una *piconet.* Para moverse de un estado a otro se usan comandos de capas más altas o señales internas. La *figura 2-10* muestra la representación de los estados y la interacción entre ellos.

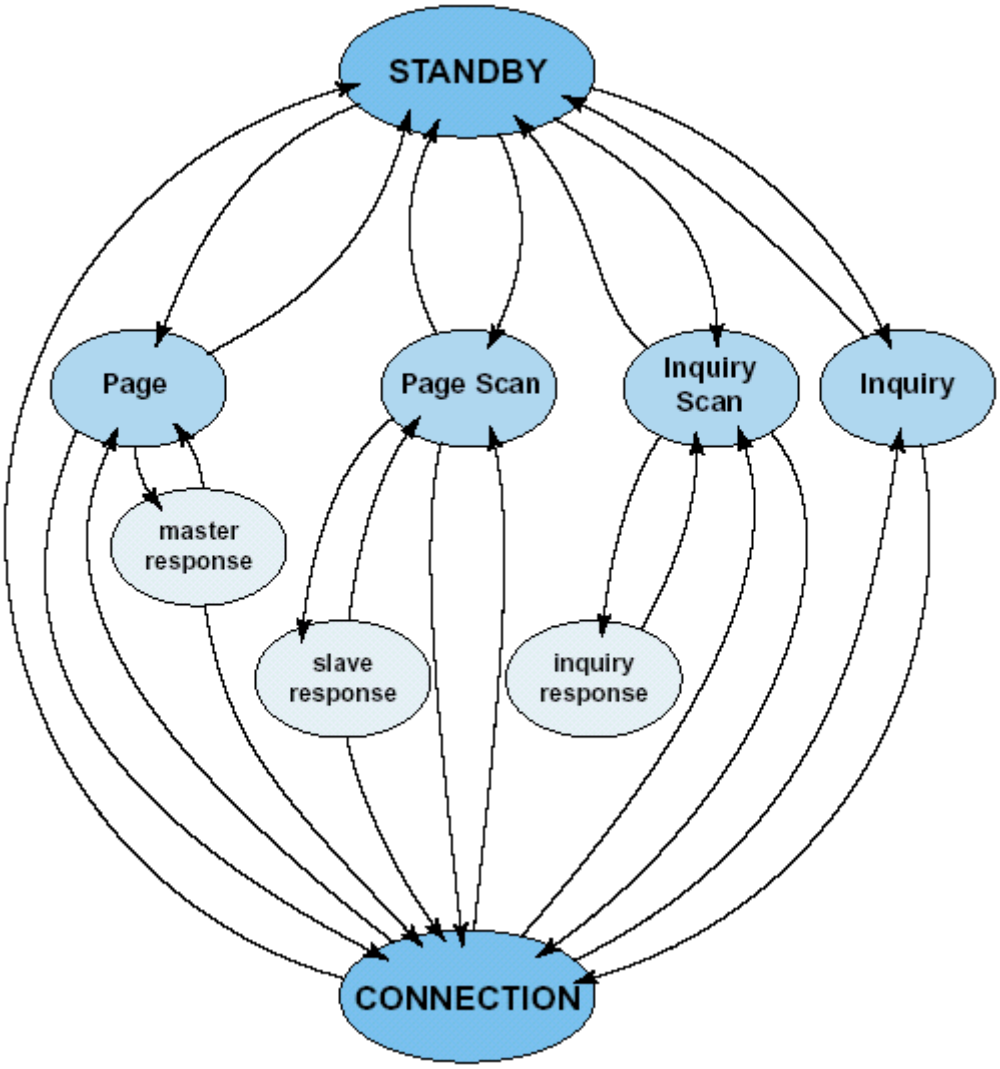


Figura 2-10 : Representación de los distintos estados de un dispositivo

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 rango. Durante un sub-estado 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, regularmente entra en un sub-estado 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, y la segunda, detecta un tipo específico de unidades. Los esclavos que se encuentran en el sub-estado de **page scan**, escuchan esperando su propio código de acceso de dispositivo. El maestro en el sub-estado **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 sub-estado de page scan. Si es el maestro quien lo excede, entonces retorna al sub-estado 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 2-11* se muestra la inicialización de la comunicación sobre el nivel banda base.

Cada transceiver (transmisor-receptor) Bluetooth tiene una única dirección de dispositivo de 48 bits asignada, la cual está dividida en tres campos: campo LAP, campo UAP y campo NAP.

Los campos LAP y UAP forman la parte significativa del código de acceso. En la Figura 2-11 se puede observar el formato de la dirección para un dispositivo Bluetooth. La dirección del dispositivo es conocida públicamente y puede ser obtenida a través de una llamada a la función inquiry.

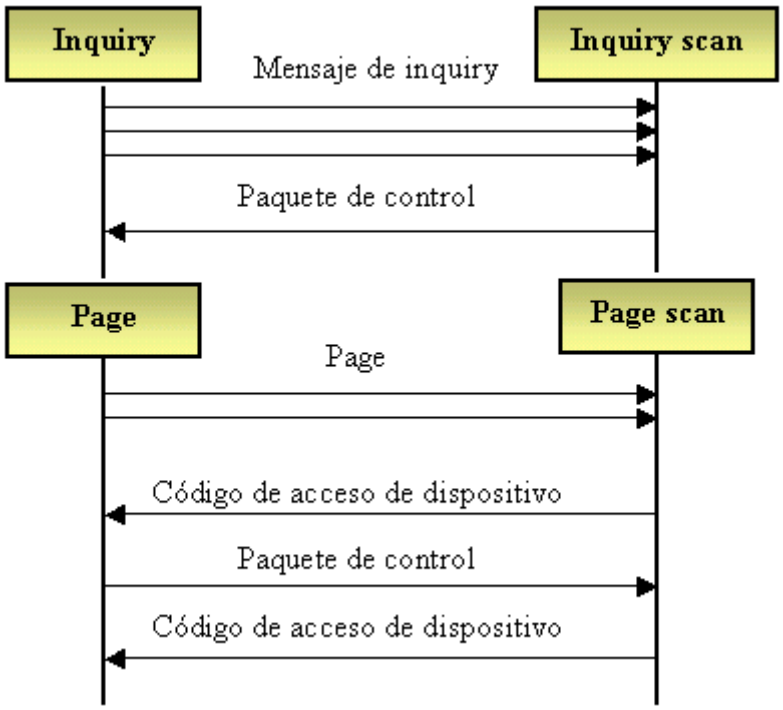


Figura 2-11 : Iniciación de comunicación sobre el nivel banda base

Figura 2-12 : Dirección de dispositivo Bluetooth

2.2.1.8. Seguridad en Bluetooth

Con el fin de brindar protección y confidencialidad a la información, el sistema debe ofrecer medidas de seguridad en las dos capas, la de aplicación y de enlace. Todas las unidades *Bluetooth* tienen implementadas las mismas rutinas de autenticación y encriptación. En la capa de enlace, estas rutinas constan de cuatro entidades diferentes: una única dirección pública, dos llaves secretas y un número aleatorio el cual es diferente para cada transacción. Solamente es encriptada la carga útil. El código de acceso y la cabecera de paquete nunca son encriptados.

Cada tipo de unidad *Bluetooth* tiene una rutina de autenticación común. El maestro genera un número aleatorio y lo envía al esclavo, el esclavo usa este número y su propia identidad para calcular el número de autenticación. Después, este número es enviado al maestro,

quien hace el mismo cálculo. Si los dos números generados son iguales entonces la autenticación es concedida.

La encriptación, frecuentemente está restringida por leyes de varios países. Para evitar estas limitaciones, en *Bluetooth* la llave de encriptación no es estática, ésta es deducida de la llave de autenticación cada vez que se activa la encriptación.

2.2.2. Protocolo de gestión de enlace (LMP)

En el protocolo de gestión de enlace 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 enlace soporta mensajes para:

- Autenticación
- Paridad
- Encriptación
- Temporización y sincronización
- Versión y características
- Switch 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 timeslots específicos.
- Modo **park:** para que el esclavo permanezca inactivo pero sincronizado en la *piconet.*
- Enlaces *SCO*
- Control de paquetes **multi-slot**
- Supervisión de enlace

2.2.2.1. Establecimiento de conexión

Después del procedimiento **paging**, el maestro debe encuestar al esclavo enviando paquetes de sondeo. El otro lado recibe este mensaje y lo acepta o lo rechaza. Si es aceptado, la comunicación, incluyendo las capas superiores están disponibles. En la Figura 2-13 se observan el flujo de mensajes para establecer la conexión.

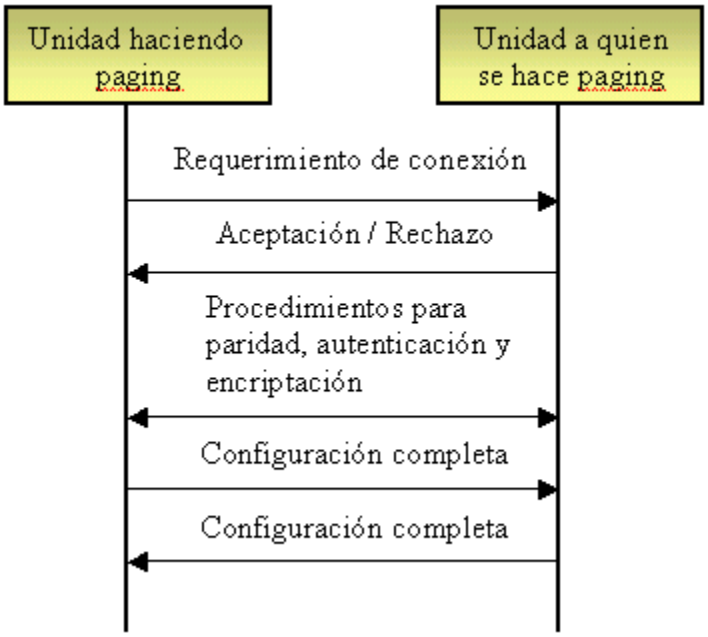


Figura 2-13 : Establecimiento de la conexión

2.2.3. Interfaz del controlador de host (HCI)

El HCI proporciona una interfaz de comando al controlador banda base y a la gestión de enlace, además de acceso al hardware y a los registros de control. Esta interfaz brinda un método estándar para acceder a los recursos de banda base Bluetooth .

2.2.3.1. Capas mas bajas del stack Bluetooth

A continuación se hace una breve descripción de las capas más bajas de la pila de software y hardware Bluetooth (*Figura 2-14*). El firmware HCI implementa los comandos HCI para el hardware a través del acceso de comandos banda base, comandos de la gestión de enlace, hardware de registros de estado, registros de control y registros de eventos. La *Figura 2-15* muestra el recorrido de un dato transferido de un dispositivo a otro. El **driver** HCI (en el **host)** intercambia datos y comandos con el firmware HCI (en el hardware). El **driver** de la

capa de transporte, por ejemplo un bus físico, brinda a las dos capas HCI la posibilidad de intercambiar información.

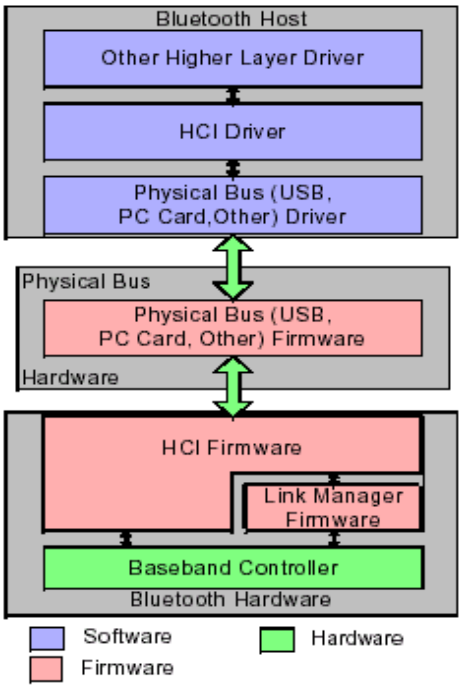


Figura 2-14 : Diagrama general de las capas más bajas

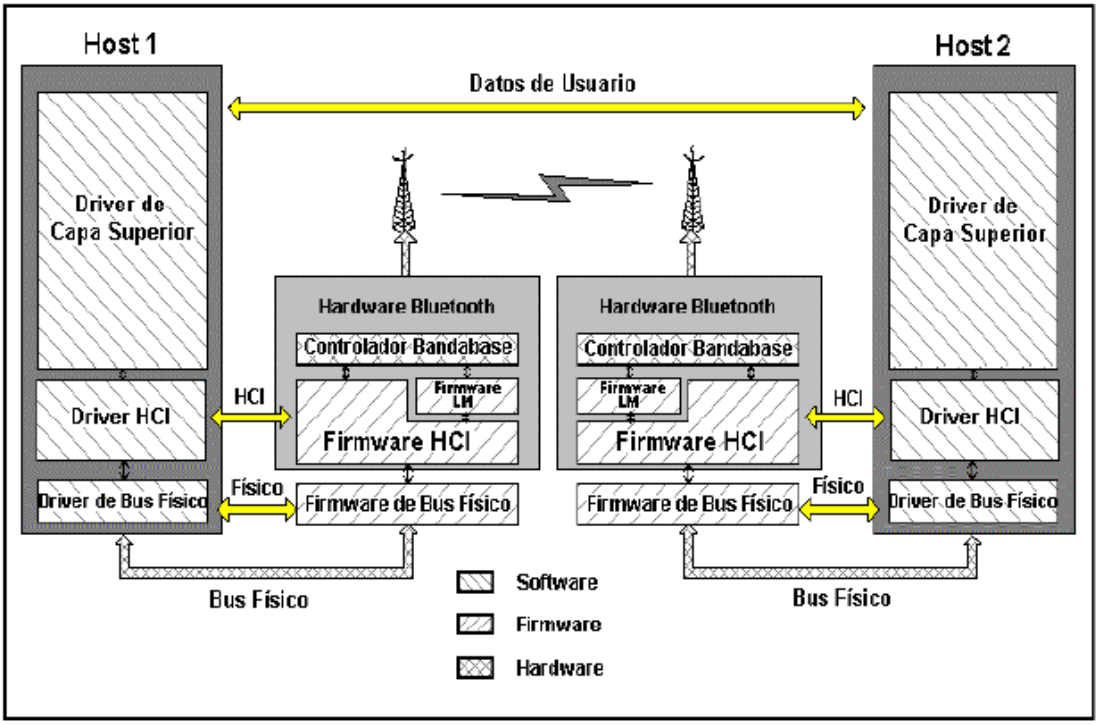


Figura 2-15 : Diagrama general end to end de las capas de software más bajas

2.2.3.2 Posibles arquitecturas de bus físico

Los dispositivos Bluetooth tienen varias interfaces de bus físicas que pueden ser usadas para conectar el hardware. Estos buses pueden tener diferentes arquitecturas y parámetros. El controlador de host soporta tres arquitecturas de bus físico, USB, UART y PC Card. Todas ellas pueden manejar varios canales lógicos sobre el mismo canal físico simple (a través de **endpoints**), por lo tanto los canales de control, datos y voz no requieren alguna interfaz física adicional. El objetivo principal de la capa de transporte del controlador de host es la transparencia entre el driver del controlador de host y el controlador de host.

2.2.4. 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 kilobytes, con capacidad de multiplexación de protocolo, operación de segmentación y reensamble y abstracción de grupos. 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) .

2.2.4.1 Canales

L2CAP está basado en el concepto de canales. Se asocia un identificador de canal, CID, a cada uno de los **endpoints** de un canal L2CAP. Los CIDs están divididos en dos grupos, uno con identificadores reservados para funciones L2CAP 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.

2.2.4.2. Operaciones entre capas

Las implementaciones *L2CAP* deben transferir datos entre protocolos de capas superiores e inferiores. Cada implementación debe soportar un grupo de comandos de señalización,

además, debe ser capaz de aceptar ciertos tipos de eventos de capas inferiores y generar eventos para capas superiores. En la *Figura 2-16* se muestra esta arquitectura.

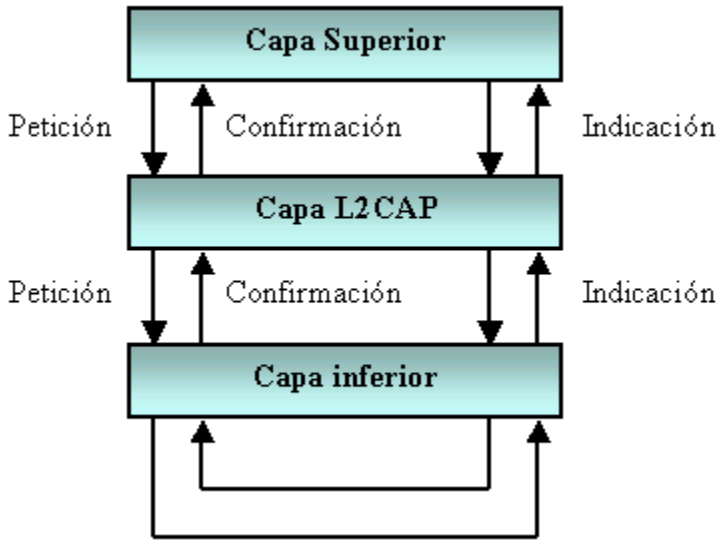


Figura 2-16 : Arquitectura L2CAP

2.2.4.3. 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 2-17* muestra la segmentación L2CAP.

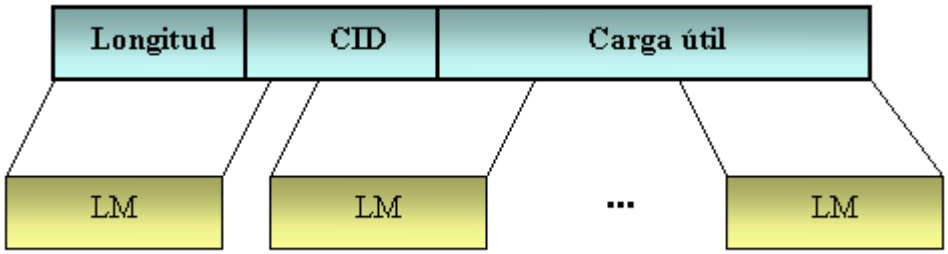


Figura 2-17 : Segmentación L2CAP

2.2.4.4. 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.

2.2.4.5 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.**

2.2.4.6 Formato del paquete de datos

L2CAP está basado en paquetes pero sigue un modelo de comunicación basado en canales. Un canal representa un flujo de datos entre entidades L2CAP en dispositivos remotos. Los canales pueden ser o no orientados a la conexión. Como se puede observar en la Figura 2-18, los paquetes de canal orientado a la conexión están divididos en tres campos: longitud de la información, identificador de canal, e información.

Figura 2-18 : Paquete L2CAP

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.

2.2.4.7 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 una determinada rata 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 **timeslots** y otros recursos libres.

2.2.5 Protocolo de descubrimiento de servicio (SDP)

El protocolo de descubrimiento de servicio brinda a las aplicaciones recursos para descubrir qué servicios están disponibles y determinar las características de dichos servicios.

2.2.5.1 Descripción General

Un servicio es una entidad que puede brindar información, ejecutar una acción o controlar un recurso a nombre de otra entidad. El SDP ofrece a los clientes la facilidad de averiguar sobre servicios que sean requeridos, basándose en la clase de servicio o propiedades específicas de estos servicios. Para hacer más fácil la búsqueda, el SDP la habilita sin un previo conocimiento de las características específicas de los servicios. Las unidades Bluetooth que usan el SDP pueden ser vistas como un servidor y un cliente. El servidor posee los servicios y el cliente es quien desea acceder a ellos. En el SDP esto es posible ya que el cliente envía una petición al servidor y el servidor responde con un mensaje. El SDP solamente soporta el descubrimiento del servicio, no la llamada del servicio.

2.2.5.2 Registros de servicio

Los registros de servicio contienen propiedades que describen un servicio determinado. Cada propiedad de un registro de servicio consta de dos partes, un identificador de propiedad y un valor de propiedad. El identificador de propiedad es un número único de 16 bits que distingue cada propiedad de servicio de otro dentro de un registro. El valor de propiedad es un campo de longitud variable que contiene la información.

2.2.5.3 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.

2.2.6 RFCOMM

El protocolo RFCOMM brinda emulación de puertos serie sobre el protocolo L2CAP. La capa RFCOMM es una simple capa de transporte provista adicionalmente de emulación de circuitos de puerto serie RS-232. El protocolo RFCOMM soporta hasta 60 puertos emulados simultáneamente. Dos unidades Bluetooth que usen RFCOMM en su comunicación pueden abrir varios puertos serie emulados, los cuales son multiplexados entre sí. La Figura 2-19 muestra el esquema de emulación para varios puertos serie.

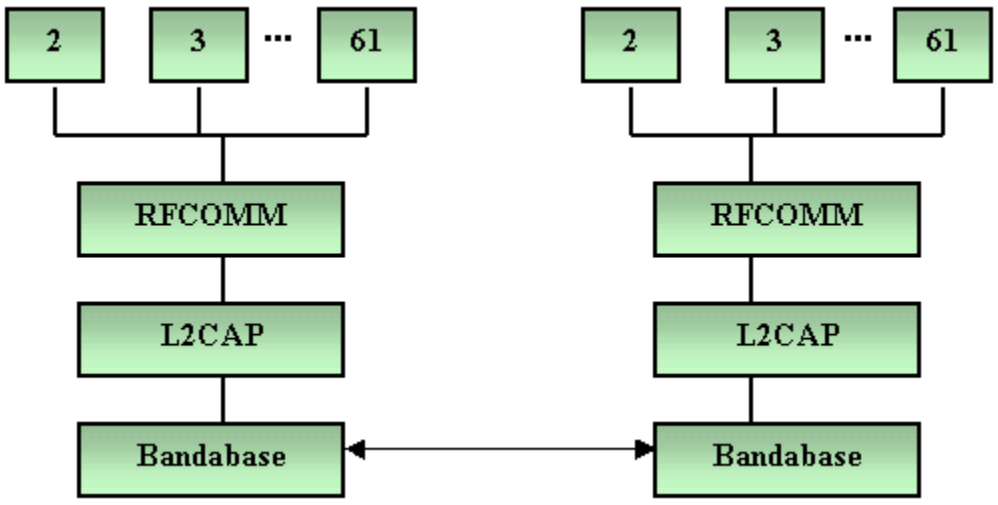


Figura 2-19 : Varios puertos serie emulados mediante RFCOMM

Muchas aplicaciones hacen uso de puertos serie. El RFCOMM está orientado a hacer más flexibles estos dispositivos, soportando fácil adaptación de comunicación Bluetooth. Un ejemplo de una aplicación de comunicación serie es el protocolo punto-a-punto (PPP). El RFCOMM tiene construido un esquema para emulación de **null modem** y usa a L2CAP para cumplir con el control de flujo requerido por alguna aplicación .

2.2.7 Perfiles Bluetooth

El estándar Bluetooth fue creado para ser usado por un gran número de fabricantes e implementado en áreas ilimitadas. 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 Bluetooth **Special Interest Group** (SIG), 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 usuario más importantes y sus perfiles. Estos cuatro modelos son :

- Perfil Genérico de Acceso (GAP),
- Perfil de Puerto Serie,
- Perfil de Aplicación de Descubrimiento de Servicio (SDAP) y
- Perfil Genérico de Intercambio de Objetos (GOEP).

A continuación se hace una breve descripción de estos y algunos otros perfiles Bluetooth. La Figura 2-20 muestra el esquema de los perfiles Bluetooth. En ella se puede observar la jerarquía de los perfiles, como por ejemplo que todos los perfiles están contenidos en el Perfil Genérico de Acceso (GAP).

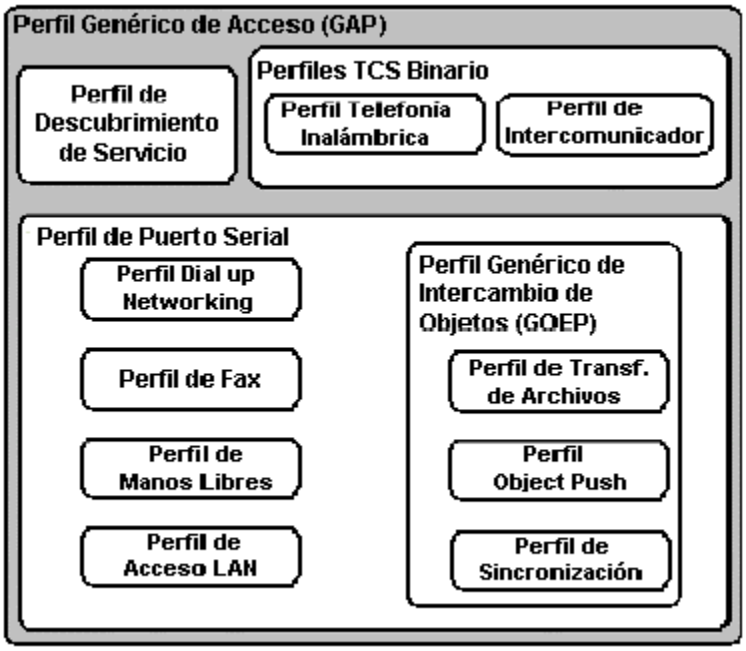


Figura 2-20 : Los Perfiles Bluetooth

2.2.7.1 Perfil Genérico de Acceso (GAP)

Este perfil define los procedimientos generales para el descubrimiento y establecimiento de conexión entre dispositivos *Bluetooth*. El *GAP* maneja el descubrimiento y establecimiento entre unidades que no están conectadas y asegura que cualquier par de unidades *Bluetooth*, sin importar su fabricante o aplicación, puedan intercambiar información a través de *Bluetooth* para descubrir qué tipo de aplicaciones soportan las unidades.

2.2.7.2 Perfil de Puerto Serie

Este perfil define los requerimientos para dispositivos *Bluetooth*, necesarios para establecer una conexión de cable serial emulada usando RFCOMM entre dos dispositivos similares. Este perfil solamente requiere soporte para paquetes de un **slot.** Esto significa que pueden ser usadas ratas de datos de hasta 128 kbps. El soporte para ratas más altas es opcional.

RFCOMM es usado para transportar los datos de usuario, señales de control de *modem* y comandos de configuración. *El perfil de puerto serial* es dependiente del *GAP*.

2.2.7.3 Perfil de Aplicación de Descubrimiento de Servicio (SDAP)

Este perfil define los protocolos y procedimientos para una aplicación en un dispositivo *Bluetooth* donde se desea descubrir y recuperar información relacionada con servicios localizados en otros dispositivos. El *SDAP* es dependiente del *GAP*.

2.2.7.4 Perfil Genérico de Intercambio de Objetos (GOEP)

Este perfil define protocolos y procedimientos usados por aplicaciones para ofrecer características de intercambio de objetos. Los usos pueden ser, por ejemplo, sincronización, transferencia de archivos o modelo **Object Push**. Los dispositivos más comunes que usan este modelo son agendas electrónicas, *PDAs*, teléfonos celulares y teléfonos móviles. El *GOEP* es dependiente del *perfil de puerto serial.*

2.2.7.5 Perfil de Telefonía Inalámbrica

Este perfil define cómo un teléfono móvil puede ser usado para acceder a un servicio de telefonía de red fija a través de una estación base. Es usado para telefonía inalámbrica de hogares u oficinas pequeñas. El perfil incluye llamadas a través de una estación base, haciendo llamadas de intercomunicación directa entre dos terminales y accediendo adicionalmente a redes externas. Es usado por dispositivos que implementan el llamado "teléfono 3-en-1".

2.2.7.6 Perfil de Intercomunicador

Este perfil define usos de teléfonos móviles los cuales establecen enlaces de conversación directa entre dos dispositivos. El enlace directo es establecido usando señalización de telefonía sobre *Bluetooth.* Los teléfonos móviles que usan enlaces directos funcionan como **walkie-talkies.**

2.2.7.7 Perfil de Manos Libres

Este perfil define los requerimientos, para dispositivos *Bluetooth,* necesarios para soportar el uso de manos libres. En este caso el dispositivo puede ser usado como unidad de audio inalámbrico de entrada/salida. El perfil soporta comunicación segura y no segura.

2.2.7.8 Perfil Dial-up Networking

Este perfil define los protocolos y procedimientos que deben ser usados por dispositivos que implementen el uso del modelo llamado *Puente Internet*. Este perfil es aplicado cuando un teléfono celular o *modem* es usado como un *modem* inalámbrico.

2.2.7.9 Perfil de Fax

Este perfil define los protocolos y procedimientos que deben ser usados por dispositivos que implementen el uso de fax. En el perfil un teléfono celular puede ser usado como un fax inalámbrico.

2.2.7.10 Perfil de Acceso LAN

Este perfil define el acceso a una *red de área local, LAN,* usando el protocolo punto-a-punto, *PPP*, sobre *RFCOMM. PPP* es ampliamente usado para lograr acceder a redes soportando varios protocolos de red. El perfil soporta acceso *LAN* para un dispositivo *Bluetooth* sencillo, acceso LAN para varios dispositivos Bluetooth y PC-a-PC (usando interconexión *PPP* con emulación de cable serial).

2.2.7.11 Perfil Object Push

Este perfil define protocolos y procedimientos usados en el modelo **object push**. Este perfil usa el *GOEP*. En el modelo **object push** hay procedimientos para introducir en el **inbox**, sacar e intercambiar objetos con otro dispositivo *Bluetooth.*

2.2.7.12 Perfil de Transferencia de Archivos

Este perfil define protocolos y procedimientos usados en el modelo de transferencia de archivos. El perfil usa el *GOEP*. En el modelo de transferencia de archivos hay procedimientos para chequear un grupo de objetos de otro dispositivo *Bluetooth,* transferir objetos entre dos dispositivos y manipular objetos de otro dispositivo. Los objetos podrían ser archivos o fólderes de un grupo de objetos tal como un sistema de archivos.

2.2.7.13 Perfil de Sincronización

Este perfil define protocolos y procedimientos usados en el modelo de sincronización. Éste usa el *GOEP*. El modelo soporta intercambios de información, por ejemplo para sincronizar calendarios de diferentes dispositivos.

2.3 Software Bluetooth

El software para un host Bluetooth corresponde a las capas de la pila de protocolos y utilidades Bluetooth implementadas en software e instaladas en el host. Un host puede ser cualquier sistema microprocesado programable (PC's, teléfonos celulares, mouse, impresoras, teclados, etc.), capaz de ejecutar las líneas de código correspondientes a la pila de protocolos.

Por tanto, es necesario tener una interfaz (UART, USB, etc..) para comunicar el host y el módulo Bluetooth. Son muchas las pilas de protocolos Bluetooth disponibles para el host, implementados en diversos lenguajes de programación y sobre distintas plataformas, siendo también muchas las empresas interesadas en su desarrollo. Independientemente de la plataforma o el lenguaje de programación, estos se basan en el **CORE** (Núcleo) y los **PROFILES** (Perfiles) de la especificación del sistema Bluetooth.

La Figura 2-21 describe esquemáticamente la implementación de la pila de protocolos Bluetooth al nivel de software, firmware y hardware, especificando su ubicación ya sea en el host o en el hardware Bluetooth.

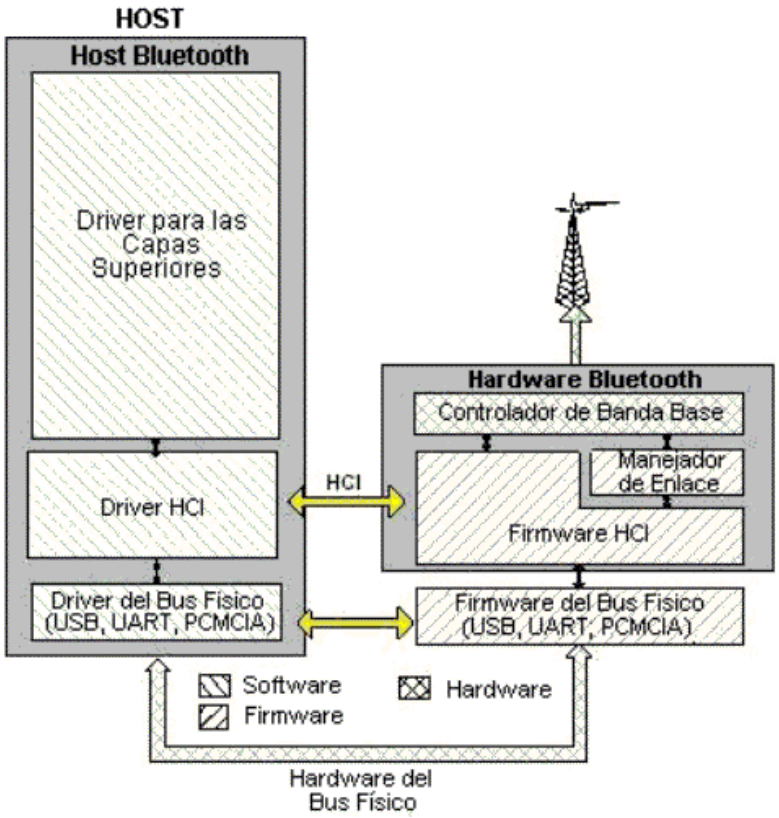


Figura 2-21 : Modelo de Implementación del protocolo Bluetooth. Software – Firmware –Hardware

El host y el hardware Bluetooth se comunican a través del HCI. El firmware HCI implementa los comandos HCI para el hardware Bluetooth teniendo acceso a los comandos de banda base, manejador de enlace, registros de estatus del hardware, registros de control y registros de eventos.

Muchas capas pueden existir entre el driver HCI ubicado en el host y el firmware HCI ubicado en el hardware Bluetooth. Estas capas intermedias, se encargan del control y transporte de datos a través de un medio físico (un bus físico ya sea USB, PC Card, RS232, u otro), para que se establezca una comunicación transparente entre el driver HCI y el firmware HCI, permitiendo el intercambio de datos y comandos entre estos dos.

El host recibirá notificaciones asíncronas de los eventos HCI independientemente de la capa de control de transporte del host que se esté usando. Los eventos HCI son usados para notificar al host que ocurre algo.

2.3.1. Tipos de software

Son muchas las compañías que trabajan en el desarrollo de stacks de protocolos *Bluetooth* para el host, en especial, los fabricantes de módulos *Bluetooth,* quienes ofrecen productos de prueba y kits de desarrollo que incluyen software para el host, cuya finalidad es facilitar y agilizar el desarrollo de aplicaciones. Este software no es usualmente de libre distribución, es decir que su licencia tiene un coste, el cual, en la mayoría de los casos, está incluido en el coste del kit de desarrollo.

Sin embargo, los fabricantes de módulos *Bluetooth* no son los únicos que ofrecen software para el host, hay muchas otras empresas y universidades interesadas en su desarrollo, razón por la cual es posible encontrar software demostrativo y de libre distribución con licencia pública **GPL** (GNU Public License). En esta tesis se trata únicamente el software para sistemas cuyo host corresponde a un PC, es decir, no se cubre software para sistemas embebidos.

Los requerimientos tales como sistema operativo, capacidad de memoria, entre otros, son propios de cada software.

2.3.1.1 Software Bluetooth con propiedad de licencia

Debido a que muchas de las características de estos diferentes tipos de software son muy parecidas, se da una breve descripción de los más representativos.

- **Bluetooth Host Stack** Software para el host desarrollado por *Ericsson*, el cual implementa las siguientes capas del stack de protocolos: driver *HCI*, *L2CAP*, *RFCOMM*, *SDP* y opcionalmente *OBEX* y *TCS*. Este stack está escrito en ANSI C, y es independiente del ambiente de desarrollo (compilador, debugger, linker, etc.);

esto sumado al concepto de Sistema Operativo Virtual, hace que este stack, sea adaptable tanto a sistemas operativos para sistemas embebidos como a los sistemas operativos estándar tales como Linux y Windows. Las siguientes son herramientas de software disponibles para Windows, basadas en el Bluetooth Host Stack de Ericsson: Bluetooth PC reference Stack, Bluetooth Development Kit y Bluetooth Application Tool Kit .

- **Bluetooth Software Suite** Desarrollado por **Digianswer A/S**, este software escrito para Microsoft Windows además de brindar un API (Interfaz de Programa de Aplicación) de fácil uso, soporta los siguientes perfiles: acceso general, servicio de descubrimiento de aplicaciones, puerto serial, red de acceso conmutado, fax, acceso a LAN, OBEX, Object Push, transferencia de archivos y Ethernet .

- **BlueStack** Software desarrollado por **MEZOE,** una división de Cambridge Consultants Ltd. Escrito en lenguaje C, implementa al igual que los dos anteriores, un **driver** HCI y las capas superiores del protocolo Bluetooth. Son ofrecidos varios productos basados en el BlueStack para Microsoft Windows: este es el caso de StackPrimer, Proto Developer e Interface Express .

2.3.2.2 Software Bluetooth con licencia pública (GPL)

En esta sección se tratan las principales características de los tipos de productos de libre distribución con licencia GPL desarrollados para Linux y disponibles en Internet. Los principales son: OpenBT, BlueDrekar, Affix y BlueZ.

- **OpenBT** Axis Communications Inc. desarrolló, en primera instancia, un driver Bluetooth para Linux llamado OpenBT, el cual puede ser usado tanto en ambientes eLinux (Linux Embebido) como en Linux para PC. Este fue el primer stack de protocolos disponible, siendo ahora un proyecto de código fuente abierto, desarrollado en un kernel de Linux 2.0.33, de tal manera que no debe presentar problemas de funcionamiento en versiones posteriores del kernel. Implementa el perfil de LAN mediante enlaces PPP sobre RFCOMM y además una forma simple del perfil de descubrimiento de servicios (SDPP). El código fuente en sus versiones para PC y eLinux, se encuentra disponible en el portal de Internet de Sourceforge, y es posible conseguir más información en el portal del proyecto .

- **BlueDrekar** El driver Bluetooth de IBM, disponible bajo licencia GPL, implementa la capa de transporte UART de la Interfaz Controladora del host (HCI). Este código sirve de referencia para escribir drivers para otras capas de transporte HCI, como USB. El BlueDrekar middleware es una implementación de referencia de las capas altas del protocolo Bluetooth, disponible bajo licencia alpha Works. Según el fabricante, este código ha sido desarrollado y probado en PCs con procesadores de arquitectura i486 o superior, corriendo un kernel de Linux 2.2.12-20 (Red Hat 6.1), 2.2.14-5.0 (Red Hat 6.2), 2.2.16-22 (Red Hat 7). Han sido utilizados para realizar

las pruebas de desempeño los módulos Bluetooth de **Ericsson** con la capa de transporte UART y firmware versión P7C (v1.0A) y P3E (v1.0B).

- **Affix** Este **stack** de protocolo para Linux fue desarrollado por el Centro de Investigación de Nokia en Helsinki, Finlandia. El software Bluetooth de Affix trabaja sobre plataformas Intel, ARM (iPaq, ARM9 y otras) y PowerPC (iMac). Las últimas versiones del software están disponibles en el sitio web de Affix .

- **Bluez** Este es el stack de protocolos Bluetooth oficial de Linux, el cual es parte del kernel 2.4 y posteriores. Bluez brinda a sus usuarios la capacidad de comunicación con dispositivos Bluetooth, entre los que se tienen adaptadores USB, teléfonos móviles y puntos de acceso entre otros, además de conexión inalámbrica entre dos o más computadoras. Bluez consta del **core** HCI, **drivers** HCI para UART, USB y emulador de HCI, módulo del protocolo L2CAP y utilidades de configuración y prueba .

2.3.2 Descripción de la pila de protocolos BlueZ

Bluez es la pila de protocolos Bluetooth oficial de Linux, el cual brinda soporte para las capas y protocolos del **core** Bluetooth. La información contenida en este ítem es una síntesis de los principales capítulos del **HowTo** de Bluez disponible en el sitio oficial de Bluez en Internet .

Estas son sus principales características :

- Arquitectura flexible, eficiente y modular
- Soporta múltiples dispositivos *Bluetooth*
- Procesamiento de datos para multienlaces
- Abstracción del Hardware
- Interfaz de **socket** estándar para todas las capas

Bluez consta de :

- **Core** HCI
- **drivers** HCI para UART, USB y emulador de HCI
- Módulos para el protocolo L2CAP
- Utilidades de configuración y prueba

La *Figura 2-22* es un esquema de la arquitectura de Bluez, que indica las capas implementadas por este software dentro del **stack** de protocolos Bluetooth. El código fuente de Bluez se puede obtener de la página oficial de Bluez en Internet. Requiere para su funcionamiento un Kernel de Linux 2.4.4 o posterior. Si se desea utilizar las últimas versiones del **stack**, es necesario deshabilitar del kernel el soporte nativo de Bluetooth. Bluez se puede usar con dispositivos Bluetooth con interfaz Serial o USB, además brinda

un dispositivo HCI virtual (vhci) el cual permite depurar y probar las aplicaciones sin necesidad de tener conectados dispositivos Bluetooth reales.

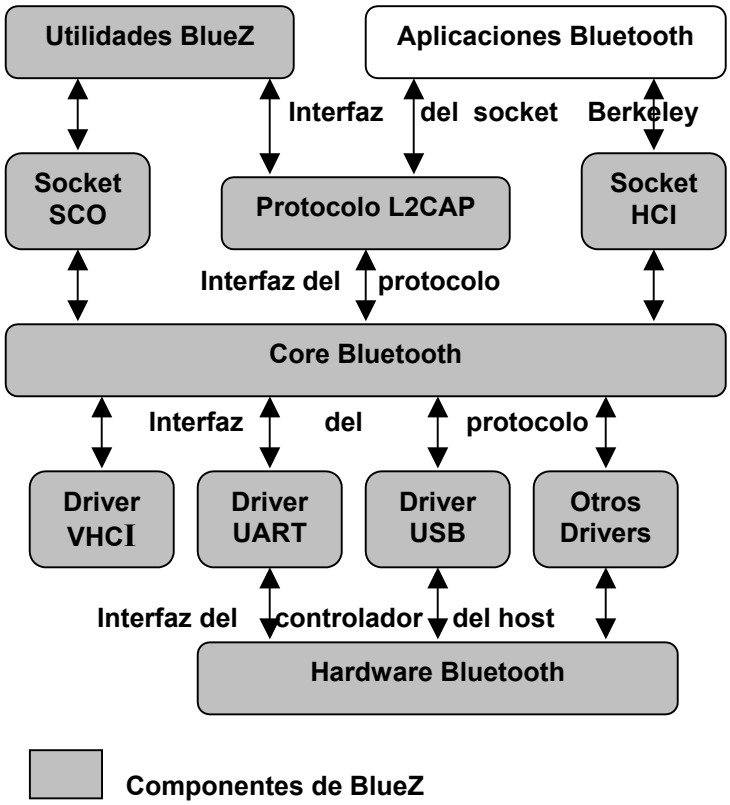


Figura 2-22 : Diagrama de la Arquitectura de Bluez

2.3.5.1 Compilación e Instalación

Bluez se compone de los siguientes paquetes :

- bluez-kernel-X.X.tar.gz
- bluez-utils-X.X.tar.gz
- bluez-libs-X.X.tar.gz
- bluez-SDP-X.X.tar.gz
- bluez-PAN-X.X.tar.gz.

Los archivos **README** o **INSTALL** incluidos dentro de cada paquete contienen toda la información sobre los requerimientos e instrucciones para su instalación. Los procedimientos de compilación e instalación son muy parecidos para cada uno de estos paquetes; en general el siguiente procedimiento para instalar el paquete bluez-kernel-

X.X.tgz se puede seguir con los demás paquetes, reemplazando el nombre del archivo por el correspondiente:

1. Copiar las fuentes en el directorio /usr/src

2. Desempaquetar el archivo fuente:

tar –xzvf bluez-kernel-X.X.tar.gz

3. Ubicarse dentro del directorio bluez-kernel-X.X (o el correspondiente a instalar)

cd bluez-kernel-X.X

4. Correr el **script** de configuración:

./configure

5. Compilar las fuentes:

make

6. Instalar las fuentes:

make install

Debe escribirse las siguientes líneas de código en el la línea de comandos de *Linux*, únicamente después de instalar el paquete bluez-kernel-X.X, esto con el fin de crear el dispositivo *HCI* virtual (*vhci*):

```
mknod /dev/vhci c 10 250 chmod 664/dev/vhci
C=0;
While [$C -lt 256];
do
if [ ! -c /dev/rfcomm$C ];
then
mknod -m 666 /dev/rfcomm$C c 216 $C
fi
C=`expr $C + 1`
done
```

Para que Bluez trabaje correctamente se deben escribir las siguientes líneas en el archivo /etc/modules.conf :

```
alias net-pf-31 bluez
alias bt-proto-0 l2cap
alias bt-proto-2 sco
alias bt-proto-3 rfcomm
alias bt-proto-4 bnep
alias tty-ldisc-15 hci_uart
alias char-major-10-250 hci_vhci
```

Después de esto se debe correr el comando "**depmod –a**" para habilitar la carga automática de los módulos Bluez. Estos módulos también se pueden cargar manualmente mediante el comando "**modprobe**" acompañado de uno de los siguientes módulos :

- **bluez** → Core Bluetooth
- **hci_usb** → Driver HCI USB
- **hci_uart** → Driver HCI UART
- **hci_vhci** → Driver para el dispositivo virtual HCI
- **l2cap** → Módulo L2CAP
- **sco** → Módulo SCO (Synchronous Connection Oriented)
- **rfcomm** → Módulo RFCOMM

Hay que decir, que todo el procedimiento de instalación explicado sería el que habría que seguir en un sistema Linux que no tuviera el núcleo de BlueZ ni los paquetes de aplicaciones en el propio CD de instalación. Ahora casi todas las distribuciones de Linux vienen al menos con el núcleo integrado en el Kernel del sistema y la mayoría también con todos los paquetes como opción de instalación. El método se ha explicado para aquellos casos en los que el sistema operativo no disponga de ninguna funcionalidad de BlueZ.

3. Desarrollo del proyecto

3.1 Introducción

Para llevar a cabo el proyecto existían dos obstáculos claros y principales que había que superar :

En primer lugar había que diseñar una interfaz robusta que nos hiciera de puente entre Bluetooth[1] y Python, ya que este último no dispone de soporte para esta tecnología. Por tanto, haciendo uso de las funciones que nos ofrece Python para la extensión del lenguaje, esta interfaz se tradujo en un modulo de código C que define e implementa una serie de funciones para poder tener acceso desde un programa escrito en Python a algunas de las características del nivel HCI de los dispositivos Bluetooth. En el punto 3.2 se entrará en detalle con este módulo.

En segundo lugar, y una vez tenemos acceso a la capa HCI de Bluetooth por medio de la interfaz descrita anteriormente, surge el problema de la comunicación y transferencia de información. La forma usual en redes de cualquier tipo de transferir información entre dos dispositivos a nivel de programa es mediante el uso de sockets (que son la abstracción de red para los terminales de un canal para un protocolo concreto), y aquí surge el problema. Si Python no ofrece soporte para interactuar con dispositivos Bluetooth aislados, evidentemente tampoco ofrecerá soporte para que estos puedan entablar ningún tipo de comunicación. Por lo tanto hay que crear otra interfaz, ahora con la capa L2CAP.

En este caso se ha optado por modificar el modulo llamado ***socketmodule.c*** que da soporte a sockets "normales" TCP/IP en Python, agregándole el código necesario para que también soporte sockets L2CAP de Bluetooth. Se preguntará el lector , ¿y porque los sockets de esta capa de la pila de protocolos? . Porque es la capa clave en la pila de protocolos de Bluetooth en cuanto a comunicación entre entidades, es la capa de mas alto nivel que ofrece sockets a las demás capas pero manteniendo los pilares de comunicación de Bluetooth (por ejemplo en cuanto a tamaño máximo de un paquete, etc..) Por encima de L2CAP, como se ha mostrado en capítulos previos, están RFCOMM, TCP/IP y otra serie de protocolos que ofrecen también sus sockets, pero evidentemente son para otros propósitos, como por ejemplo el formar parte de una red TCP/IP, aspecto que no abarca el presente proyecto y que en todo caso quedaría pendiente para posibles ampliaciones o mejoras del mismo. En el punto 3.3 se verán en profundidad los cambios efectuados en este módulo.

Una vez tenemos el soporte para acceder desde Python a Bluetooth, el siguiente paso es desarrollar la aplicación en sí. Como describe claramente el título del proyecto se trata de un peer-to-peer entre dispositivos Bluetooth que se intercambiaran archivos entre ellos, estos dispositivos conformaran una PAN (o wireless PAN) y podrán actuar tanto de servidores como de clientes en función de si en un determinado instante quieren ofrecer información al resto de dispositivos existentes en el radio de cobertura de la red o si por el

[1] Realmente entre la capa HCI de Bluetooth y Python. Esta capa es la que dispone de mayor número de funciones para interactuar con los dispositivos y se ubica en un nivel intermedio respecto al Hardware.

contrario quieren ellos obtenerla de otro u otros dispositivos. En el punto 4 se detallará la arquitectura de la aplicación.

3.2 Interfaz Python – HCI

El módulo implementado se ha llamado **py_hci.c** por entender que describe bastante bien cual es su función. Como se apuntaba en el punto anterior, este módulo esta formado por una serie de funciones que permiten al usuario realizar ciertas acciones con los dispositivos, como por ejemplo obtener su nombre, realizar un *inquiry* para detectar dispositivos en su radio de cobertura, etc ...

Estas funciones, y de forma general, enmascaran dentro de ellas una llamada a una de las funciones de la capa HCI de BlueZ. Las definiciones de estas funciones están en el modulo **hci_lib.h,** que es parte de los ficheros de cabecera que Bluez proporciona dentro del paquete bluez-libs. Ya que todos los módulos que proporciona BlueZ para linux están escritos en C es necesario crear este módulo "puente" para poder acceder a ellos desde Python. Hay que apuntar que en el módulo interfaz se han construido las funciones que nos iban a hacer falta después para ser llamadas desde un programa en Python y alguna otra mas que no se ha usado, pero de ningún modo se han creado todas las funciones definidas para la capa HCI de Bluez.

Por tanto, la estructura general de cada una de las funciones puente implementada es siempre la misma y se ajusta a un patrón. A continuación se muestra el esqueleto de las mismas y después se explica su estructura:

```
static PyObject *PyBT_nombre_función(PyObject *self, PyObject *args) {

        int       var_1, var_2;

        if ( !PyArg_ParseTuple(args, "i", &var_1 ) )
              return NULL

        /* otras acciones */

        var_2 = hci_nombre_función( var_1);

        return Py_BuildValue( "i", var_2 );
}
```

- En primer lugar se reciben los parámetros de la llamada que vienen desde el programa en Python y se copian en variables definidas en C. La función **PyArg_ParseTuple()** se encarga de esta misión. En este caso la "i" indica que el siguiente argumento es un entero.

- Después hay que llamar a la función HCI escrita en C correspondiente que nos ofrece la funcionalidad que deseamos. A esta función y según en que caso, habrá que llamarla con todos los parámetros que previamente hemos copiado en variables C. Además la llamada a la función HCI puede que requiera algún parámetro más que no nos pasa el programa en Python y que desde aquí lo incluiremos. En el esqueleto anterior esta función es **hci_nombre_función**.

- Tras esto y tras recibir los resultados de la llamada en C puede ser necesario llamar a otro tipo de funciones para hacer conversiones de tipos de datos, etc...

- Por último habrá que devolver los resultados al programa Python, por tanto habrá que construir una o varias variables Python para devolverlas al programa. De esto se encarga la función **Py_BuildValue**.

- Hay que apuntar que el control de errores se realiza en la medida que sea posible dentro de cada una de estas funciones.

Una vez hemos definido las funciones que nos hacen falta siguiendo el esquema anterior, el módulo debe inicializarse, indicando el nombre de módulo que se exporta **(bluezhci)** en este caso). Este nombre es el que tendrán que importar los programas Python para poder usar las funciones que define. Además hay que indicar esta lista de funciones. Para inicializar el módulo hay que usar la siguiente función :

```
void initbluetooth( void ) {
        Py_InitModule("bluezhci", BluetoothMethods);
}
```

BluetoothMethods es la lista des los métodos que se exportan. El esquema de código siguiente muestra la forma que tendría :

```
static PyMethodDef BluetoothMethods[] = {

  {"nombre_función",PyBT_nombre_función,METH_VARARGS, "Descripción"},
  …
  …
  …
  {NULL, NULL, 0, NULL}        // Sentinel
}
```

Tras escribir en el programa de la siguiente directiva :

import bluezhci o **from bluezhci import ***

tendremos ya la lista de funciones que pueden ser usadas directamente desde un programa en Python para que realicen el trabajo que les hayamos encomendado. La forma de llamarlas será :

nombre_función(parametro1, parametro2, ..., parametroN)

La manera de generar el módulo a importar (**bluezhci**) se explicará en el capítulo 5, dedicado a la instalación.

3.2.1 Funciones

A continuación se describen cada una de las funciones implementadas junto con un ejemplo de uso de las mismas desde un programa en Python . La implementación de las funciones se encuentra en el CD en el que se incluye este documento en el modulo **py_hci.c**.

3.2.1.1 open_dev()

Esta función se encarga de abrir un dispositivo Bluetooth para posteriormente realizar alguna acción sobre el como por ejemplo obtener su nombre. Toma como argumento el identificador del dispositivo y nos devuelve un descriptor para el dispositivo de forma similar a como cuando se abre un fichero. Veamos un ejemplo de llamada :

```
from bluezhci import *

dev_id = 0      # device identifier
dd = open_dev( dev_id )
```

3.2.1.2 close_dev()

Esta función se encarga de cerrar un dispositivo Bluetooth. Toma como argumento el descriptor del dispositivo devuelto en la apertura previa para cerrarlo y no devuelve nada, igualmente haciendo una analogía con los ficheros. Un ejemplo de llamada :

```
from bluezhci import *
close_dev( dd )                 # dd → device descriptor
```

3.2.1.3 inquiry()

Esta función realiza una búsqueda de dispositivos en el radio de cobertura. Toma como argumentos el identificador del dispositivo, el tiempo en segundos durante el que se llevará a cabo el proceso y el número máximo de dispositivos que estamos dispuestos a descubrir. Como resultado obtenemos el número de dispositivos descubiertos y las direcciones físicas en el formato Bluetooth de los mismos. Veamos un ejemplo de código (**inquiry.py**) :

```
from bluezhci import *

dev_id = 0
print "Scanning for devices ..."

(num, bdaddrs) = inquiry( dev_id, 8, 10 )
print "Num responses:", num

for i in range( num ):
      print "%s" %  bdaddrs[i]
```

En la figura 3.1 se puede observar el resultado de la ejecución con 3 dispositivos :

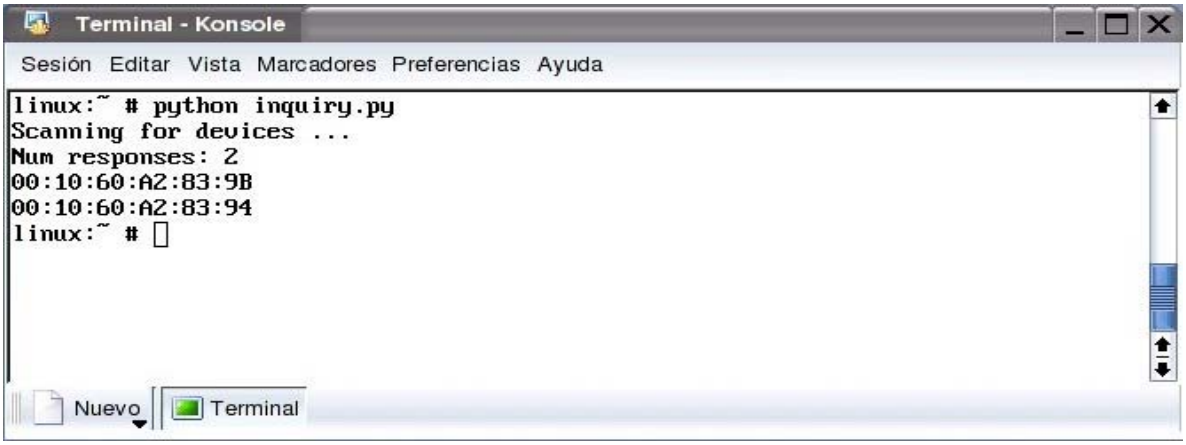


Figura 3-1 : Ejecución inquiry.py

3.2.1.4 read_local_name()

Esta función obtiene el nombre de un dispositivo local. Toma como argumento el descriptor de un dispositivo abierto previamente. Un ejemplo de código (**local_name.py**) :

```
from bluezhci import *

dev_id = 0      # device identifier
dd = open_dev( dev_id )

if dd < 0:
        print "Error opening device"
else:
        print "Name --> %s" % read_local_name(dd)
        close_dev( dd )
```

En la figura 3-2 se puede observar el resultado de la ejecución :

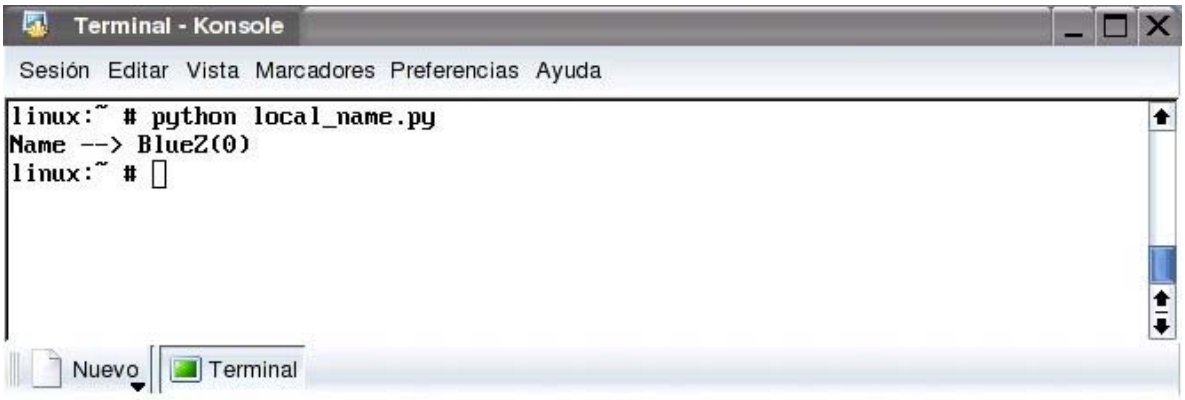


Figura 3-2 : Ejecución local_name.py

3.2.1.5 read_remote_name()

Esta función obtiene el nombre de un dispositivo remoto2. Toma como argumentos el descriptor del dispositivo origen abierto previamente y la dirección física del dispositivo remoto (obtenida por ejemplo tras realizar inquiry). Un ejemplo de código (**remote_name.py**) :

```
from bluezhci import *

dev_id = 0      # device identifier

(num, bdaddrs) = inquiry( dev_id, 8, 10 )
```

2 Aquí hay que apuntar que remoto quiere decir que desde un dispositivo origen queremos interactuar con el, por tanto, aunque sea remoto para nuestro dispositivo puede estar ubicado físicamente en la misma máquina.

```
dd = open_dev( dev_id )

if dd < 0:
        print "Error opening device"
else:
        for i in range( num ):
                print "%Name -> %s" % read_remote_name(dd, bdaddrs[i])
                close_dev( dd )
```

En la figura 3.3 se puede observar el resultado de la ejecución :

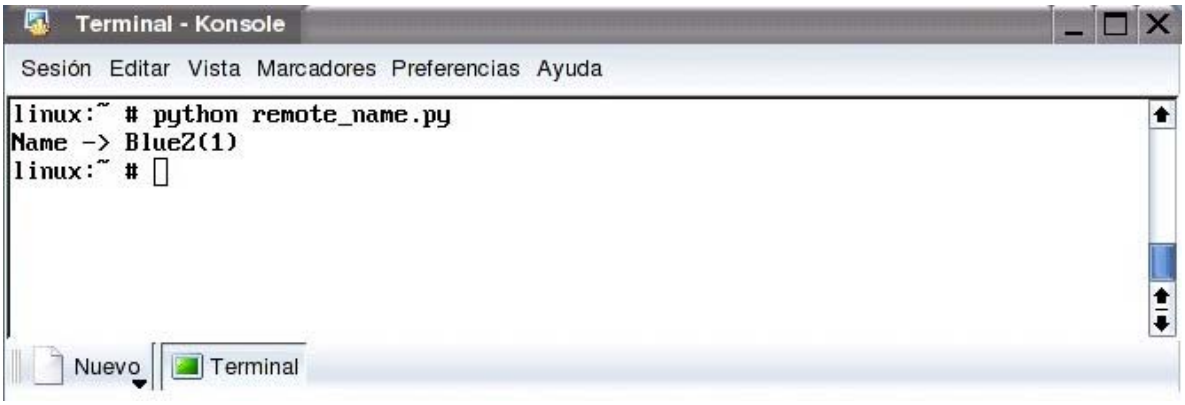


Figura 3-3 : Ejecución remote_name.py

3.2.1.6 local_device_type()

Esta función devuelve el tipo de un dispositivo local (USB, PCCARD, UART, etc...). Toma como argumento el identificador del dispositivo. Un ejemplo de código (**local_device_type.py**) :

```
from bluezhci import *

dev_id = 0      # device identifier

print "Type --> %s" % local_device_type(dev_id)
```

La figura 3-4 muestra el resultado de la ejecución :

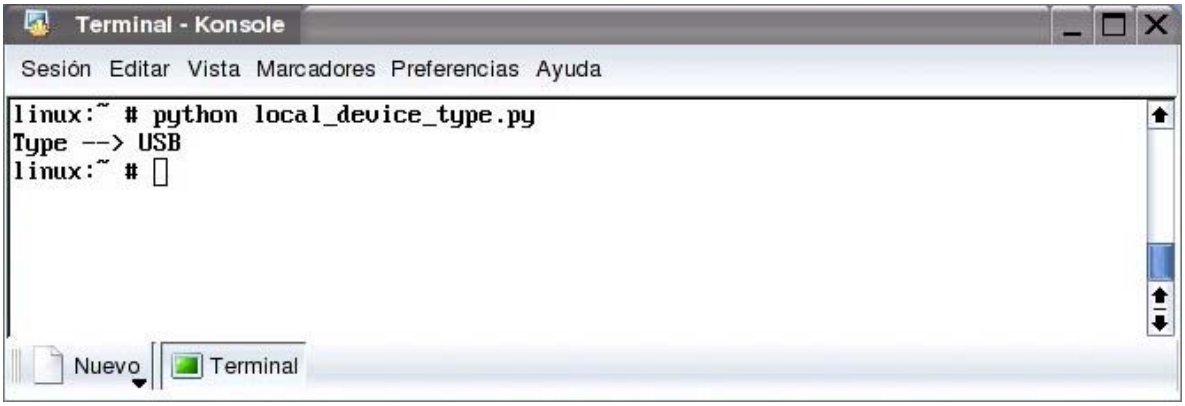


Figura 3-4 : Ejecución local_device_type.py

3.2.1.8 local_device_ba()

Esta función obtiene la dirección física de un dispositivo local. Toma como argumento el identificador del dispositivo. Un ejemplo de código (**local_device_ba.py**) :

```
from bluezhci import *

dev_id = 0      # device identifier

print "Address --> %s" % local_device_ba(dev_id)
```

La figura 3-5 muestra el resultado de la ejecución :

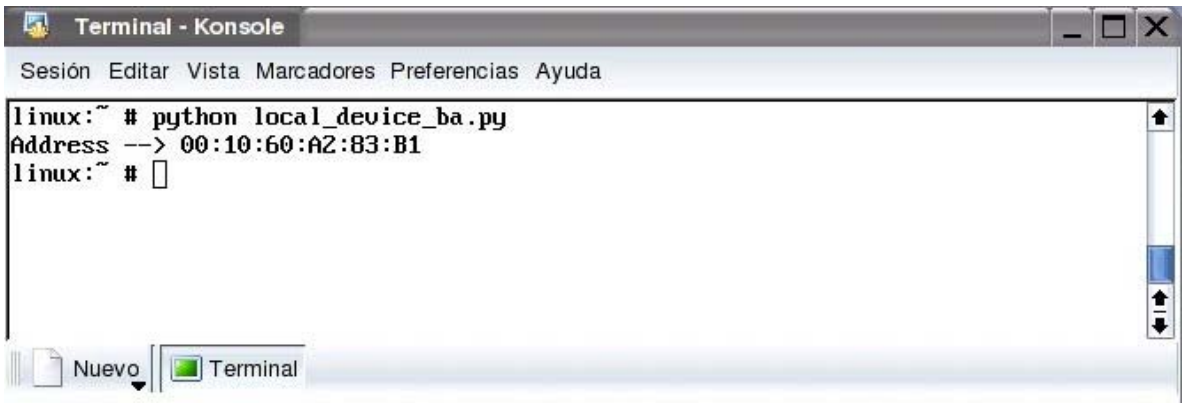


Figura 3-5 : Ejecución local_device_ba.py

3.2.1.9 switch_role ()

Esta función permite cambiar el rol (o modo de funcionamiento) de un dispositivo remoto siempre que entre ellos exista una conexión. Es decir, si dos dispositivos están conectados siendo el dispositivo local el esclavo y el remoto el maestro, el dispositivo local puede hacer que el remoto pase a ser el esclavo, y por lo tanto él pasara a ser el maestro. En esta situación, ahora puede hacer lo contrario y pasar otra vez a ser maestro el dispositivo remoto, y por tanto él, el esclavo.

Es importante destacar que para hacer uso de esta función no importa que un dispositivo sea maestro o esclavo, se puede usar independientemente del rol que se posea. La función toma como argumentos el descriptor del dispositivo origen, la dirección del dispositivo destino y el rol que queremos que asuma el dispositivo destino[3]. Hay que darse cuenta que un dispositivo no puede cambiarse el rol a si mismo directamente, al cambiar el del dispositivo remoto, indirectamente se cambia el suyo. El valor devuelto es un string, "OK" si todo salió bien o "FAIL" si hubo algún error.

A continuación se muestra un ejemplo de código (**switch_role.py**) :

```
from bluezhci import *

dev_id = 0      # device identifier

dd = open_dev( dev_id )

if dd < 0:
        print "Error opening device"
else:
      s = switch_role(dd,"00:10:60:A2:83:94",1)    # 0 --> master, 1 --> slave
      close_dev( dd )

      print s
```

En la figura 3-6 se puede ver el resultado de la ejecución. En ella, primero se observa la configuración de los dos dispositivos. Después, el hci0 establece una conexión con un dispositivo remoto (hci1) siendo este ultimo el maestro de la conexión. Después se muestra el estado de la conexión y se ve como efectivamente cada uno cumple su rol. Tras esto se ejecuta el programa anterior en el que se indica que el dispositivo remoto (hci1) pase a ser esclavo. Al observar de nuevo el estado de la conexión tras la llamada se ve como los dispositivos han cambiado los papeles.

[3] 0→ maestro , 1 → esclavo

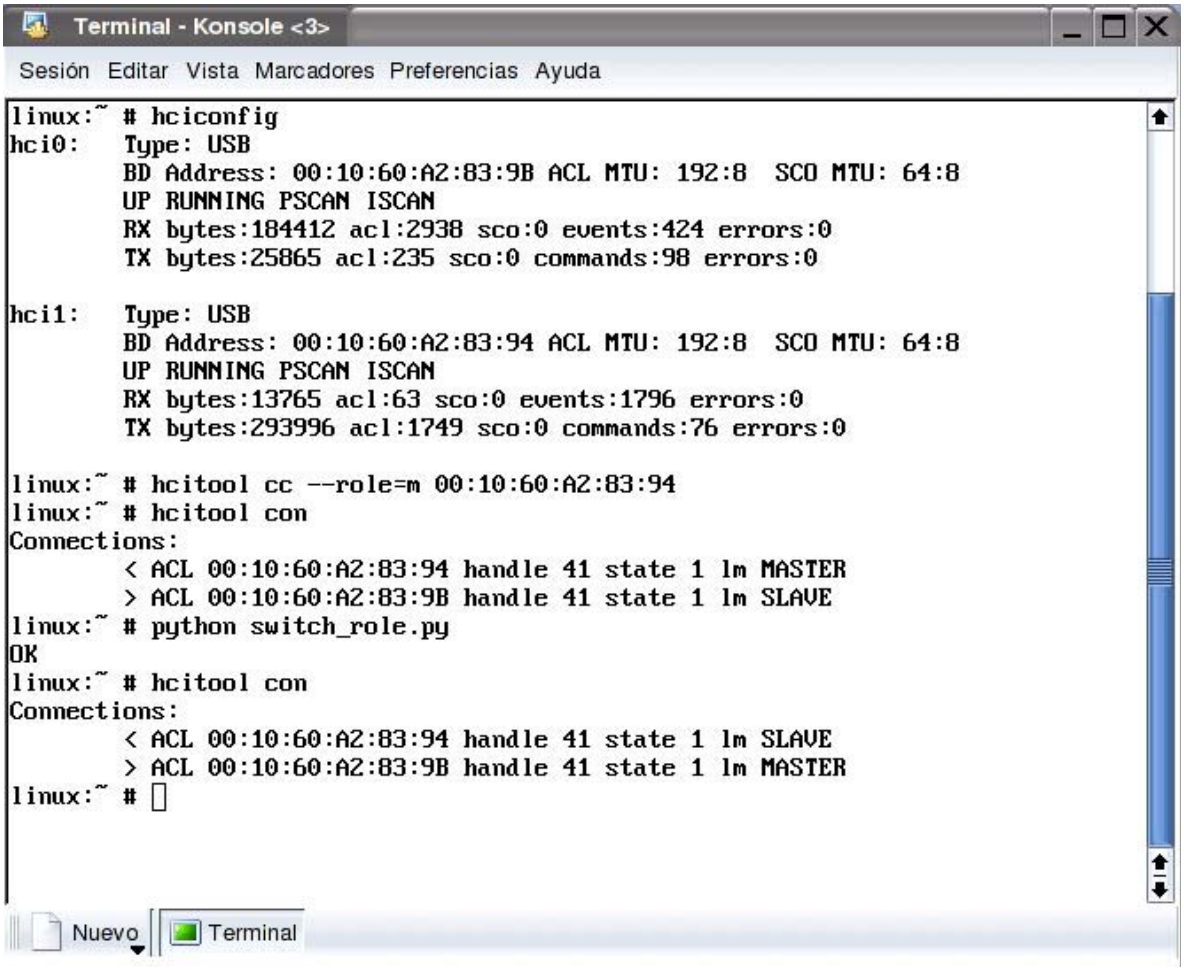


Figura 3-6 : Ejecución switch_role.py

3.2.1.10 Otras funciones

Otras funciones que se han implementado pero que no se ha hecho uso de ellas han sido las siguientes :

- create_connection(), establece una conexión hci entre dos dispositivos
- disconnect(), deshace una conexión hci creada previamente

3.3 Interfaz Python – L2CAP

Como se mencionaba en el punto 3.1 el objetivo siguiente es ahora dotar a Python de soporte para sockets L2CAP de BlueZ. Por tanto hay que realizar cambios en todos aquellos ficheros de código que intervengan en la definición e implementación del soporte para sockets. En nuestro caso este soporte lo brindan los siguientes ficheros :

- socketmodule.c
- socketmodule.h
- addrinfo.h

Tras los cambios, el primer y segundo fichero han pasado a llamarse **bluezsocket.c** y **bluezsocket.h** respectivamente. En el tercer fichero se ha mantenido el nombre porque no se ha realizado ningún cambio de código en el mismo. Este fichero de cabecera se limita a definir las estructuras que proporcionan información sobre la dirección de un socket además de una serie de flags para indicar condiciones de error, etc..

Hay que decir, que además de los cambios necesarios para proporcionar acceso a Bluetooth se han suprimido de estos ficheros todas las partes que hacen referencia a aquellos entornos para los que la aplicación final no está destinada, como por ejemplo Windows. Hay que tener en cuenta que Python está hecho para trabajar tanto en Linux como en Windows, además de otros sistemas, pero hay que recordar que BlueZ es tan solo para entornos Linux, por tanto se puede obviar todo lo que haga referencia a otros sistemas. A continuación se describen los cambios realizados en los ficheros.

socketmodule.h → bluezsocket.h

En este módulo hay que realizar pocos cambios, en primer lugar hay que incluir los ficheros de cabecera de bluetooth que definen los nombres de entorno necesarios, en este caso son :

```
#include <bluetooth/bluetooth.h>
#include <bluetooth/l2cap.h>
```

Después, hay que añadir dentro del campo que indica los tipos de direcciones permitidas en la definición de los objetos socket (PySocketSockObject), el tipo de dirección que queremos agregar, que en este caso es L2CAP :

```
struct sockaddr_l2 bt_l2;
```

socketmodule.c → bluezsocket.c

1) En cuanto a ficheros incluidos y definiciones, como es lógico, lo primero que hay que cambiar es la línea

#include "socketmodule.h" por #include "bluezsocket.h"

y añadir las siguientes definiciones

```
#define USE_BLUETOOTH 1
#define _BT_SOCKADDR_MEMB(s,proto) (&((s)->sock_addr).bt_##proto)
#define _BT_L2_MEMB(sa, memb) ((sa)->l2_##memb)
```

que representan que está definido Bluetooth (y por tanto se puede usar en las funciones), la estructura dirección de un objeto socket en función del protocolo y un campo de la estructura dirección de un socket L2CAP respectivamente.

2) Se han añadido nuevas funciones :

setbdaddr : setbdaddr(char *name, bdaddr_t *bdaddr)

Esta función convierte una dirección física Bluetooth representada por un string a formato numérico.

makebdaddr : makebdaddr(bdaddr_t *bdaddr)

Realiza lo contrario que la función anterior, toma como argumento una dirección Bluetooth en formato numérico y devuelve un string que representa a la misma.

3) Por último, se han modificado algunas funciones, concretamente aquellas donde interviene el tipo de familia de una dirección. Las funciones son :

makesockaddr :

PyObject *makesockaddr(int sockfd, struct sockaddr *addr, int addrlen, int proto)

Esta función devuelve un objeto Python que representa la dirección del socket pasada como segundo argumento. Este objeto se usará después para pasarlo como argumento al llamar a funciones como bind() o connect(). Para crear este objeto hay que analizar el campo familia de la estructura (struct) que representa la dirección, ya que según sea una u otra el objeto

devuelto tendrá unas características determinadas, como por ejemplo un *puerto* si es una dirección de internet o un *psm*, si es una dirección Bluetooth.

Las familias representadas en la función son AF_INET, AF_UNIX y AF_INET6, por lo que hay que añadir el código para el caso que nos ocupa, AF_BLUETOOTH. A continuación se muestra este código, que es bastante sencillo :

```
#ifdef USE_BLUETOOTH
  case AF_BLUETOOTH:
    switch (proto) {

          case BTPROTO_L2CAP: {

            struct sockaddr_l2 *a = (struct sockaddr_l2 *) addr;
                PyObject *addrobj = makebdaddr(&_BT_L2_MEMB(a, bdaddr));
                PyObject *ret = NULL;

                if (addrobj) {
                  ret = Py_BuildValue("Oi", addrobj, _BT_L2_MEMB(a, psm));
                  Py_DECREF(addrobj);
                }
                return ret;
          }
    }
#endif
```

getsockaddrarg :

```
int getsockaddrarg(PySocketSockObject *s, PyObject *args,struct sockaddr
                    **addr_ret, int *len_ret)
```

Esta función obtiene la estructura dirección de un socket y su tamaño en los dos últimos argumentos. Para ello, toma como argumentos un objeto dirección y el objeto socket en el que está incluida esta, y analiza si el objeto dirección pasado como segundo argumento concuerda con la familia que especifica el objeto socket (en nuestro caso AF_BLUETOOTH) pasado como primer argumento.

Si concuerda, la función devuelve un 1 y en los dos últimos argumentos tenemos el resultado, sino devuelve un 0. Como en la función anterior, aquí tampoco está representado el caso que nos ocupa, por lo que hay que añadir el siguiente código :

```
#ifdef USE_BLUETOOTH

  case AF_BLUETOOTH: {
    switch (s->sock_proto) {

      case BTPROTO_L2CAP: {

            struct sockaddr_l2 *addr = (struct sockaddr_l2 *)_ BT
                              _SOCKADDR_MEMB(s, l2);
            char *straddr;

            _BT_L2_MEMB(addr, family) = AF_BLUETOOTH;

            if (!PyArg_ParseTuple(args, "si", &straddr,&_BT_L2_MEMB(addr,
                                                psm))) {

          PyErr_SetString(socket_error, "getsockaddrarg: " "wrong
                        format");
              return 0;
            }

          if (setbdaddr(straddr, &_BT_L2_MEMB(addr, bdaddr)) < 0)
               return 0;

              *addr_ret = (struct sockaddr *) addr;
              *len_ret = sizeof *addr;
              return 1;
          }
      }
    }
#endif
```

getsockaddrlen :

int getsockaddrlen(PySocketSockObject *s, socklen_t *len_ret)

Esta función copia en el segundo argumento el tamaño de la estructura dirección del objeto socket pasado como primer argumento en función de la familia del mismo. La función devuelve 1 si la familia es conocida y 0 en caso contrario. Seguidamente se muestra el código añadido para la familia AF_BLUETOOTH :

```
#ifdef USE_BLUETOOTH

  case AF_BLUETOOTH: {
```

```
   switch(s->sock_proto) {

   case BTPROTO_L2CAP:
     *len_ret = sizeof (struct sockaddr_l2);
          return 1;
   }
  }
#endif
```

initbluezsocket :

```
PyMODINIT_FUNC initbluezsocket(void)
```

En este caso, en primer lugar hay que añadir algunas constantes al entorno como la familia de sockets, el protocolo y las direcciones de difusión ANY y LOCAL.

```
#ifdef USE_BLUETOOTH

PyModule_AddIntConstant(m, "PF_BLUETOOTH", AF_BLUETOOTH);
PyModule_AddIntConstant(m, "BTPROTO_L2CAP", BTPROTO_L2CAP);
PyModule_AddObject(m, "BDADDR_ANY", Py_BuildValue("s","00:00:00:00:00:00"));
PyModule_AddObject(m, "BDADDR_LOCAL", Py_BuildValue("s","00:00:00:FF:FF:FF"));

#endif
```

Además también hay que añadir las constantes necesarias para que en la función *setsockopt()*, que establece las opciones de un socket, podamos habilitar el socket L2CAP como master y así poder atender a múltiples clientes concurrentemente. Este caso se dará cuando el servidor cree el socket. La función setsockopt() necesita tres parámetros, el nivel de la opción, el nombre de la opción y el valor de la opción. A continuación se muestra la forma genérica de la función con sus parámetros:

```
setsockopt( level, optname, value)
```

Por tanto las constantes a añadir son :

```
PyModule_AddIntConstant(m, "SOL_L2CAP", SOL_L2CAP)

PyModule_AddIntConstant(m, "L2CAP_LM", L2CAP_LM)

PyModule_AddIntConstant(m, "L2CAP_LM_MASTER", L2CAP_LM_MASTER)
```

SOL_L2CAP → Representa que el nivel de la opción será L2CAP

L2CAP_LM → Representa que el nombre de la opción es LM (Link Management)

L2CAP_LM_MASTER → Representa el valor, es decir, actuar como master

Por último, y una vez hechas todas las modificaciones habría que generar el módulo **bluezsocket** que ya podríamos importar directamente en cualquier programa Python para usar sus nuevas características. No hay que olvidar que como este módulo es la extensión de uno previo, al importarlo también podríamos usar los sockets convencionales. La forma de generar el módulo Python se describirá en el apartado de instalación de la aplicación.

4. Aplicación p2p

El objetivo principal de la aplicación es hacer posible que distintos dispositivos Bluetooth puedan intercambiar información. Por tanto, una vez tenemos acceso a las funciones HCI de Bluetooth por medio de la interfaz HCI y podemos intercambiar información entre los dispositivos mediante la interfaz de sockets L2CAP, es importante saber cual es el papel que juega un dispositivo dado en el entorno en un instante de tiempo.

Lo importante, y siguiendo la filosofía de las aplicaciones peer-to-peer puras es que no existe un servidor central, por tanto, no hay ningún dispositivo que me informe de la existencia de otros ni por supuesto de la información que tienen. En una red cableada, no se podría seguir este esquema ya que un dispositivo cliente no tiene manera (en principio) de comocer la existencia de otros si no hay un servidor de por medio, pero en nuestro caso este problema está subsanado porque un dispositivo puede saber por el mismo si hay otros en su radio de cobertura por medio de la llamada a la función inquiy.

Cada nodo o dispositivo, en un momento dado puede asumir el papel de *servidor* , el papel de *cliente* o *ambos*, es decir que un dispositivo puede estar sirviendo y obteniendo información de otros a la vez. En cada instante de tiempo se puede dar cualquier situación .

Este paradigma tiene la gran ventaja de que está totalmente descentralizado y es ad-hoc, siguiendo la filosofía de una red PAN, pero la gran desventaja de que no exista un “sitio” donde se informe globalmente de la información disponible que existe en el radio de cobertura de un dispositivo, ya que no existe una base de datos global a tal efecto. Esto implica que las búsquedas de información son más lentas, ya que para ver si existe determinado archivo en el conjunto de dispositivos visibles habría que consultarlos todos.

La figura 4-1 muestra en esquema de funcionamiento de la aplicación, donde las flechas que salen de cada nodo indican que se está enviando información a un cliente. El color de la flecha indica cual es el archivo que se envía.

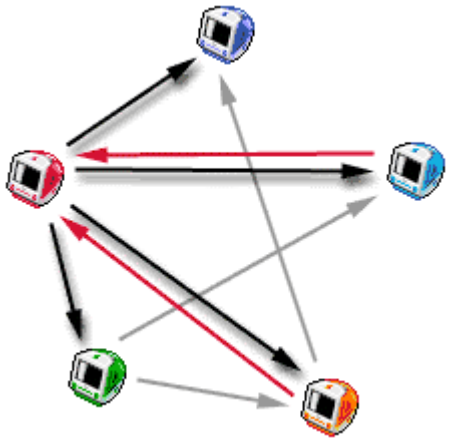


Figura 4-1 : Esquema de funcionamiento

Al lanzar la aplicación, cada dispositivo lanza su servidor estando a la escucha de posibles clientes que quieran interactuar con él, y al mismo tiempo, es un cliente que puede querer buscar otros dispositivos en el entorno que puedan servirle información. El integrar estas dos partes es importante, en un principio se pensó que un dispositivo se podía lanzar como cliente o como servidor pero esto tenía el inconveniente de que mientras era cliente de otros no podía servir información. Se podía haber optado por lanzar dos aplicaciones para cada dispositivo, una cliente y otra servidora, pero tenía el problema de no poder integrar en una misma interfaz gráfica información relativa a conexiones entrantes y la conexión saliente, a parte de ser más engorroso para el usuario.

La figura 4-2 muestra la interacción entre un cliente y un servidor remoto :

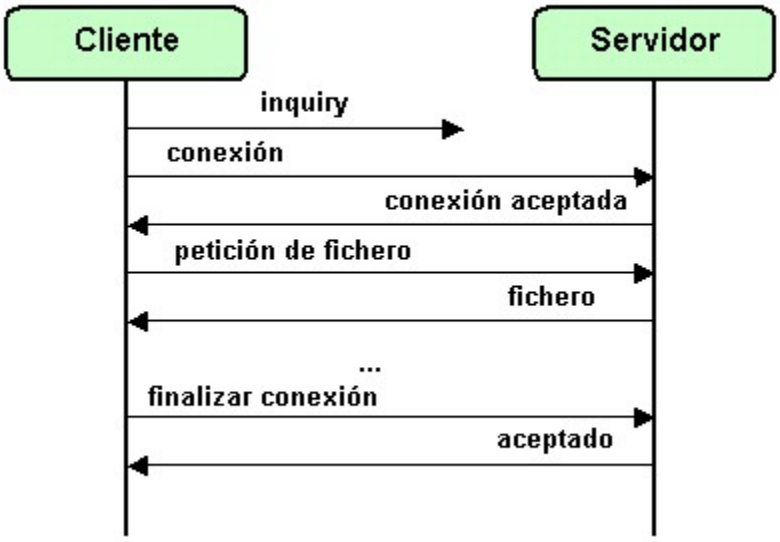


Figura 4-2 : Interacción entre un cliente y un servidor remoto

Evidentemente la figura anterior solo muestra a un cliente contra un servidor, pero el servidor estaría atendiendo los flujos de información anteriores para múltiples clientes.

4.1. Modelo de objetos

La aplicación hace uso del soporte para orientación a objetos que nos brinda Python. Por lo tanto, esta se basa en una serie de clases que ofrecen un conjunto de métodos para distintos propósitos, lo que permite aislar distintas funcionalidades. El esquema propuesto sería fácilmente ampliable para soportar nuevas funcionalidades como la mencionada anteriormente base de datos global entre otras. La figura 4-3 muestra la jerarquía de clases junto con los métodos más importantes que ofrecen.

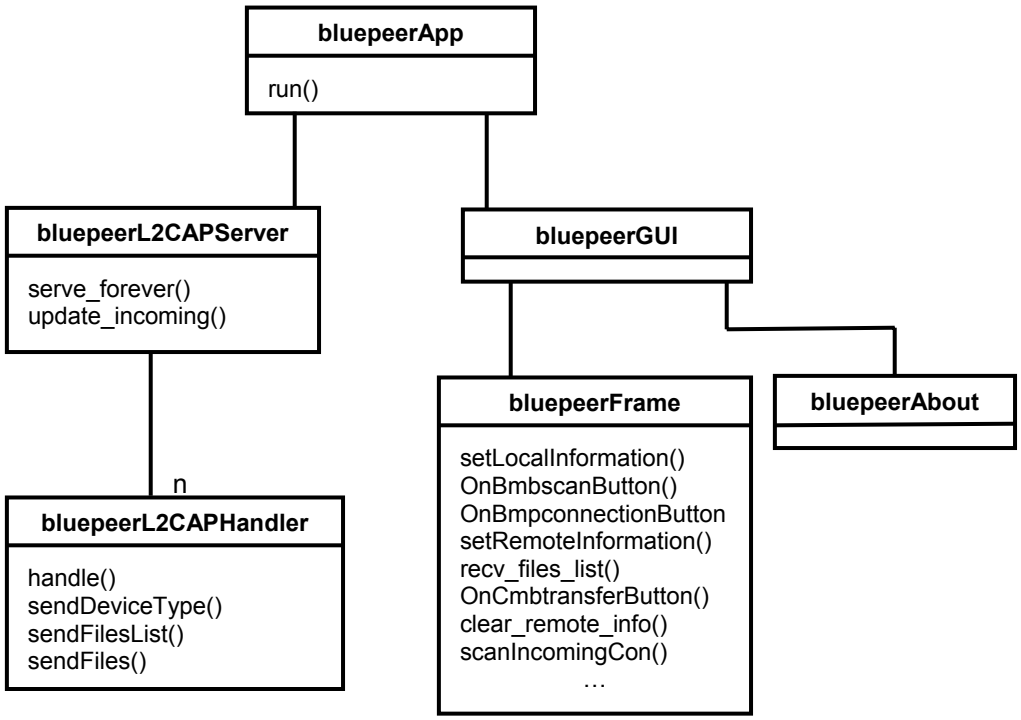


Figura 4-3 : Jerarquía de clases de la aplicación

bluepeerApp

Es la clase principal de la aplicación, encargada de lanzar el servidor y el cliente junto con la interfaz gráfica mediante el método run(). A continuación se muestra el código de este método.

```
def run(self):
        ''' Lanza la aplicacion '''

        pid = fork()

        if pid > 0:     # parent
           guiApp = bluepeerGUI(0)
           guiApp.MainLoop()

           y = getpid()
           z = getpgid(y)

           print "Server OFF"

           killpg(z,SIGQUIT)
```

```
else:   # child

    s = bluepeerL2CAPServer()
    s.serve_forever()
```

Lo primero que se hace es una llamada a fork() para bifurcar la ejecución del programa[4]. El proceso padre lanza la interfaz gráfica, que representa principalmente al cliente y a su interacción con el entorno. Además, también representa en cierta medida al servidor, ya que muestra las conexiones entrantes que llegan a este en una caja de texto. El proceso hijo lanza el servidor que se encargará de atender las peticiones que lleguen.

Para mostrar las conexiones entrantes del servidor en la interfaz, los dos procesos tienen que interactuar de algún modo. Como son procesos distintos, el espacio de nombres de cada uno es diferente, lo que implica, que para que el servidor y el cliente puedan comunicarse debe ser de una manera externa al espacio de nombres, es decir , de una manera distinta a la instanciación de clases. El servidor no puede instanciar una clase que represente al cliente y llamar por ejemplo a un método que copie el nombre de una conexión entrante en una variable y el contenido de esta a su vez en una caja de texto, ya que por la razón mencionada anteriormente nunca leerían las mismas variables por ser estas diferentes y pertenecientes al entrono de nombres de cada proceso.
La solución por la que se ha optado ha sido que se comuniquen mediante un fichero de texto temporal en el cual el servidor va anotando las conexiones que llegan y el cliente va leyendo periódicamente de ese fichero para poder mostrarlas en el lugar correspondiente. Posiblemente no sea la única ni la mejor solución, pero tras bastante tiempo intentando solventar el problema se optó por solucionarlo por esta vía.

Cuando se cierra la interfaz gráfica, y por tanto la aplicación, el proceso padre moriría y abandonaría la ejecución. Sin embargo si esto fuera así sin mas, el proceso hijo, encargado de gestionar al servidor quedaría **zombie** y consumiría recursos de sistema. Si esto pasara una vez no lo notaríamos, pero sin lanzásemos la aplicación una y otra vez nos daríamos cuenta de que las prestaciones del sistema descenderían notablemente.

Dicho esto, sería suficiente con matar al proceso hijo antes de salir para solucionar el problema , pero no es así, ya que como se ha dicho anteriormente el proceso hijo representa a un servidor de conexiones que a su vez instancia procesos hijos por cada conexión que le llega, por lo que estaríamos en el mismo problema que al principio. Para solucionarlo se ha optado por llamar a la función **killpg()** antes de salir. Esta función mata a todos los procesos que pertenecen al grupo al que pertenece el proceso que llama, por tanto mata a todos los descendientes del proceso.

[4] Aquí se podrían haber usado hilos de ejecución pero se ha optado por la primera opción por considerarla en su momento más robusta frente a la existencia de procesos zombies en el sistema.

bluepeerL2CAPServer

Esta clase representa al servidor de conexiones L2CAP con soporte para múltiples conexiones. Solo se instancia una vez y se utiliza para admitir conexiones de los clientes y poder tratarlas adecuadamente. Cada conexión instanciará a la clase bluepeerHandler que se encarga del proceso de la comunicación con el cliente.

bluepeerL2CAPHandler

Representa al manejador de conexiones. Esta clase es la encargada de gestionar cada una de las conexiones entrantes de los clientes y desencadenar todo el proceso de servicio adecuado a cada una. Posteriormente se detallará más este proceso cuando se describan los métodos que forman parte de ella. Se instanciará tantas veces como conexiones se produzcan en el tiempo de vida de la aplicación.

bluepeerGUI

Esta clase es la encargada de inicializar y lanzar el entorno gráfico. De ella dependen las clases *bluepeerFrame* y *bluepeerAbout*.

bluepeerFrame

Representa la ventana principal de la aplicación desde donde el usuario lleva a cabo toda la interacción con el entorno. Representa al cliente y al servidor y es instanciada por la clase bluepeerGUI. Más adelante se explicará la funcionalidad de los métodos más importantes.
En los dos puntos siguientes se describen las partes servidora y cliente de la aplicación indicándose las funciones que cumplen cada una y los métodos que les dan soporte.

4.2. Servidor

La parte servidora de la aplicación en un principio se limita a escuchar posibles conexiones de clientes que quieran establecer comunicación. Cuando se establece una, guarda la dirección de este dispositivo en un fichero temporal de conexiones entrantes. Este fichero se utiliza para tener un registro de quien está conectado al servidor. En el momento en que un dispositivo cliente deja de serlo, se borrara inmediatamente del fichero.

Hecho esto, la siguiente acción que puede realizar el servidor es enviar un fichero a cierto cliente que lo solicite, por tanto buscara el fichero pedido y se lo enviará vía el socket L2CAP creado anteriormente para atender a esta conexión. El proceso se repetirá tantas veces como ficheros quiera descargarse el cliente. La última acción que realizaría el servidor respecto al cliente sería cerrar la conexión abierta y matar al proceso encargado de atenderla para que no quede *zombie* por el sistema.

En la figura 4-4 quedan reflejadas a modo de esquema las distintas acciones que puede realizar el servidor.

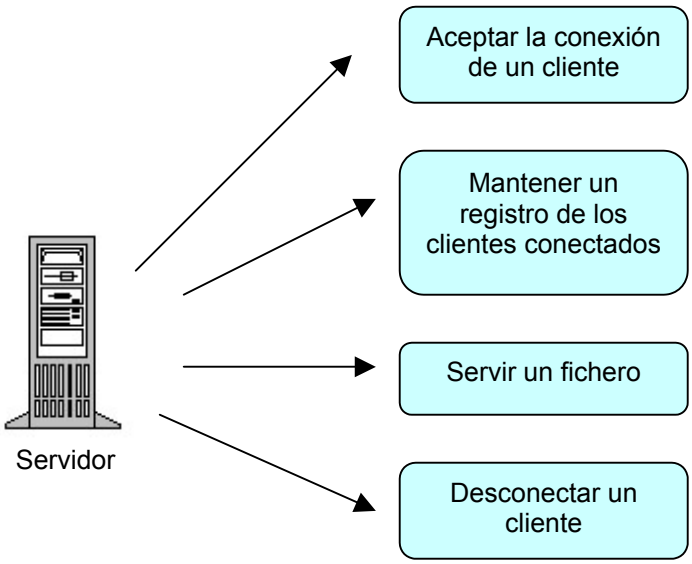


Figura 4-4 : Diagrama de acciones del servidor

A continuación se muestran todos los métodos que usa el servidor para llevar a cabo su cometido. Estos métodos son los que contienen las clases bluepeerL2CAPServer y bluepeerL2CAPHandler.

serve_forever()

Este método pertenece a la clase *bluepeerL2CAPServer*. Es el encargado de implementar el servidor de peticiones . Las acciones que realiza son las siguientes:

- En primer lugar crea un socket L2CAP que será el que esté a la escucha en todo momento de posibles clientes. Una vez creado lo asocia a la dirección física del dispositivo y aun PSM con la función ***bind()***. El PSM para no entrar en muchos detalles podría equipararse a un puerto TCP. Después hay que llamar a setsockopt() para habilitar al socket como *master* y poder atender varias peticiones concurrentemente. Tras esto se llama a la función ***listen()*** para ponerse a la escucha, indicando el número máximo de conexiones concurrentes.

- Por último, se entra en el bucle general de este método que estará atendiendo clientes mientras el servidor esté activo, a continuación de muestra el código :

```
while(1):

connection,address = self.blue_sock.accept()

p = os.fork()

if p > 0:       # parent
          connection.close()
          continue
else:       # child

               self.blue_sock.close()

               print "Server connected by",address[0]

               self.incoming.append(address[0])

               f = open('con.txt','a')
               f.write(address[0] + '\n')
               f.close()

               bluepeerHandler(connection)
               connection.close()
               self.update_incoming(address[0])

               print "Diconnect from", address[0]
               print "\n"

               pid = os.getpid()
               os.kill(pid,SIGKILL)
```

En primer lugar se llama al método **accept()** para aceptar una conexión entrante. En este momento crea un proceso hijo para atender la conexión y el padre se desentiende de ella y vuelve a ejecutar una nueva iteración del bucle.

El proceso hijo guarda en la variable de clase "incoming" la dirección del dispositivo remoto para el que se acaba de establecer la conexión con el fin de tener una lista de los dispositivos remotos que están siendo atendidos. Además también escribe esta dirección en el fichero "**con.txt**" que será el medio para establecer comunicación con la interfaz gráfica para que muestre los dispositivos conectados, como se ha comentado anteriormente.

Tras esto se le pasa el socket recién creado al manejador de conexiones que será el encargado de realizar las comunicaciones. Una vez el cliente decide abandonar la conexión la ejecución vuelve al punto donde se quedó, es decir, después de la llamada al manejador, entonces se cierra el socket correspondiente a la conexión y se llama a la función

update_incoming que actualiza los clientes conectados. Por último se mata al proceso encargado de gestionar la conexión y la ejecución termina.

update_incoming()

Este método también pertenece a la clase *bluepeerL2CAPServer*. Como se ha comentado este método actualiza los clientes que están conectados, para ello, borra de la variable de clase *incoming* la entrada correspondiente a la dirección del dispositivo que se quiere eliminar. La dirección se pasa como parámetro de la llamada a la función. Además de esto actualiza el fichero "cont.txt" machacando su contenido escribiendo el conjunto de valores de la variable "incoming" que corresponden con las conexiones que aún están activas.

handle()

Este y los métodos siguientes ya pertenecen todos a la clase que se encarga de manejar las conexiones (*bluepeerL2CAPHandler)*. Este es el método principal de esta clase y se limita a llamar a los métodos necesarios para gestionar la conexión. A continuación se muestra su código :

```
def handle(self):
        '''Manejar una peticion entrante'''

        self.sendDeviceType()
        ret = self.sendFilesList()

        if ret == 1:
                pass
        else:
                self.sendFiles()
```

En el código se observa que se llama a tres funciones, *sendDeviceType()*, *sendFileList()* y *sendFiles()*. A continuación se describe cada una de ellas.

sendDeviceType()

Esta función se encarga de enviar al dispositivo cliente conectado el tipo de dispositivo del servidor (USB, tarjeta, etc...), es decir su propio tipo. Esta información la mostrará el dispositivo remoto en la interfaz gráfica en el sitio reservado para mostrar datos del dispositivo al que se está conectado. El hecho de enviar esta información y que el cliente no la descubra por el mismo, como hace por ejemplo con la dirección física y el nombre, es porque no se ha encontrado ninguna función HCI que la proporcionara.

sendFileList()

El objetivo principal de esta función es enviar al cliente la lista de ficheros disponibles para descargar. Si el servidor no dispusiera de ninguno devolvería el valor "1" al método que le ha llamado (*handle()*), valor que indicaría que no debe llamar al método *sendFiles()*, porque no se dispone de nada que poder enviar por el momento.

Para llevar a cabo su cometido la función lee el contenido del directorio ***/data/outgoing*** que se encuentra dentro del directorio de la aplicación. Tras esto va enviando al cliente cada una de las entradas que lee, por el socket L2CAP. Cuando ha terminado envía la cadena "FIN" para que el cliente pueda determinar la conclusión del envío. Si el directorio estaba vacío envía simplemente la cadena "EMPTY".

Es importante destacar que los archivos que se quieren servir deben encontrarse necesariamente en el directorio que se ha indicado, porque de lo contrarío el servidor no compartiría nada. En mejoras del programa se podría añadir la opción de poder configurar este directorio para poder seleccionar uno en cualquier ubicación.

sendFiles()

Esta función se encarga de enviar los ficheros solicitados por un cliente. Para ello ejecuta un bucle "*while*" que estará activo mientras el cliente lo considere oportuno. Este bucle consiste en primer lugar en recibir por el socket L2CAP el fichero requerido.

Una vez el servidor sabe cual es lo único que hay que hacer es abrirlo y comenzar a enviarlo por el socket en paquetes de 512 bytes. Una vez se ha terminado de enviar el fichero se envía la cadena "FIN" para indicarle al cliente que hemos acabado. Tras esto se retorna al principio del bucle esperando el nombre de un nuevo fichero a enviar. Si en algún momento se recibe la cadena SOCKET_CLOSED" significa que el cliente no quiere más ficheros y ha cerrado la otra parte del socket, por tanto la función de envío termina.

4.3 Cliente

La parte cliente de la aplicación la primera acción que debe hacer es realizar una llamada a la función inquiry para descubrir que dispositivos existen en su radio de cobertura. Tras esta, el conjunto de dispositivos descubiertos se mostrará en la caja de texto destinada a tal efecto de la interfaz gráfica.

Hay que decir que la función inquiry puede variar bastante su rendimiento en función de varios parámetros como la distancia, los obstáculos intermedios, etc.. Para ello, la interfaz dispone de una cajita donde podemos configurar el tiempo durante el cual se realiza la búsqueda. Este aspecto es importante ya que en ocasiones con poco tiempo no se descubren

dispositivos y se puede pensar que no existe ninguno alrededor, pero al reintentarlo aumentando un poco el tiempo se descubre que no era así.

Una vez sabe que dispositivos rondan cerca lo siguiente es conectarse a alguno de ellos para ver de que ficheros dispone y si alguno de ellos interesa. Si no fuera así se puede conectar a otro, y así sucesivamente mientras existan dispositivos. Hay que decir que si se decide conectar a otro, previamente hay que cerrar la conexión anterior, como es lógico. Esta sería la última acción que realizaría un cliente respecto a un servidor.

Ni que decir tiene que para realizar la conexión se usará un socket L2CAP que se conectará al del servidor que permanece a la espera.

En la figura 4-5 quedan reflejadas a modo de esquema las distintas acciones que puede realizar el cliente.

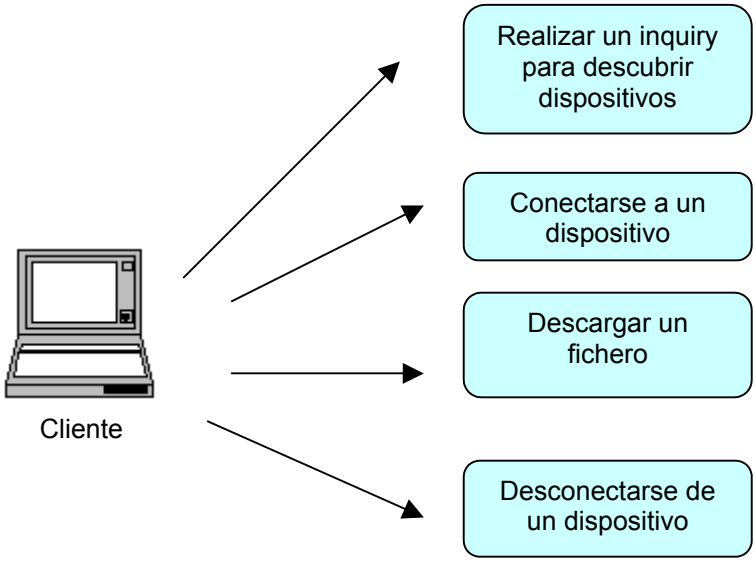


Figura 4-5 : Diagrama de acciones del cliente

A continuación se muestran todos los métodos que usa el cliente para llevar a cabo su cometido. Estos métodos son los que contienen la clase *bluepeerFrame*. Además también se muestra el método que hace posible la comunicación con el servidor del propio dispositivo (OJO !!, no con el remoto) para conocer cuales son las conexiones entrantes.

setLocalInformation()

Este método se llama automáticamente desde el método init() (que es el constructor) de la clase *bluepeerFrame*, por tanto se ejecutará el solo cuando esta se instancie. Su cometido

es obtener algunos datos del dispositivo local para mostrarlos en la interfaz y que el usuario disponga de alguna información. La información que obtiene es :

- La dirección física del dispositivo
- El nombre del dispositivo en el sistema
- El tipo de dispositivo

Para conseguir estos datos el método llama a las funciones correspondientes que proporciona el módulo ***bluetooth*** al importarlo. Estas son :

- *local_device_ba()*
- *read_local_name()*
- local_device_type()

OnBmbscanButton()

Este método está asociado al botón de *inquiry* de la interfaz. Como es lógico realiza el descubrimiento de dispositivos llamando a esta función HCI. Si no encuentra nada nos lo indica con un mensaje en pantalla y si lo consigue agregará las direcciones físicas de los dispositivos encontrados en el lugar correspondiente. Al pulsar el botón en la interfaz obtiene el valor del tiempo máximo de descubrimiento indicado y se lo pasa como parámetro a la llamada de la función.

Figura 4-6 : Mensaje de no haber encontrado dispositivos

OnBmpconnectionButton()

Este método como el anterior también está asociado a un botón, al de realizar una conexión. Tras haber seleccionado un dispositivo descubierto y pulsar el botón, el método se pone a trabajar. En primer lugar comprueba que no queremos conectarnos a un dispositivo al que ya estamos conectados, de lo contrario nos muestra un mensaje indicándonoslo.

Figura 4-7 : Mensaje que indica que ya estamos conectados a un dispositivo

Si por el contrario estamos conectados con un dispositivo A y queremos conectarnos con otro distinto B se envía un mensaje al servidor de A indicándole que terminamos la conexión, seguidamente se cierra el socket.

En el último caso y en el caso en que no estemos conectados a ningún dispositivo hay que crear un nuevo socket L2CAP que de soporte a la conexión que se quiere realizar. A continuación conectamos con el dispositivo remoto por medio de la llamada a la función ***connect()*** del socket. Si todo va bien el dispositivo remoto nos envía su tipo de dispositivo, como se ha explicado en la parte del servidor, y seguidamente se llama a la función *setRemoteInformation()*, para mostrar en la interfaz algunos datos del dispositivo, al igual que se ha hecho con el dispositivo local.

Si la llamada a *connect()* fallara nos mostraría un mensaje de alerta indicando que no está disponible. Esto puede suceder cuando descubramos a un dispositivo pero este no esté operativo, es decir, no haya lanzado la aplicación, la llamada a *inquiry()* lo detectaría pero no podríamos interactuar con él.

Figura 4-8 : Mensaje de dispositivo no disponible

Después de todo esto se llama al método *recv_files_list()* que recibe la lista de ficheros que nos ofrece el servidor remoto para descargar. Para ello se van recibiendo por el socket los paquetes de 512 bytes que nos envía el servidor hasta la señal de fin de envío. Por último se muestra en la barra de estado de la interfaz la dirección del dispositivo al que estamos conectados.

setRemoteInformation()

Como se ha mencionado antes este método realiza lo mismo que *setRemoteInformation()* pero para el dispositivo remoto, por tanto no es necesario explicar nada más. La única función del módulo ***bluetooh*** que utiliza es *read_remote_name( ),* ya que la dirección física la obtenemos de la llamada a *inquiry()* y el tipo de dispositivo es un parámetro con el que se llama a este método y nos lo pasa la función *OnBmpconnectionButton()*.

OnCmbtransferButton()

Este método esta asociado al botón de transferencia de un archivo remoto seleccionado previamente de la ventana de los archivos disponibles. Lo primero que hace es enviar al servidor remoto por el socket el nombre del archivo que queremos descargarnos. En este punto hace una comprobación de si el servidor todavía sigue activo, ya que hemos podido conectar con él pero en un momento posterior en que queremos descargarnos un archivo este ha cerrado su aplicación y ha dejado de servir. En esta situación se nos informa con un mensaje de error.

Figura 4-9 : Mensaje de dispositivo remoto no operativo

Después de esto abre un archivo en modo escritura (es decir, lo crea) con el nombre del archivo que le ha solicitado al servidor en el directorio */data/incoming/* . Seguidamente va escribiendo en dicho archivo los bytes que el servidor le está enviando. Tras la señal del servidor de fin de fichero, cierra el archivo creado. Se informa al usuario al final, de si el envío ha llegado correctamente o si ha habido algún problema que ha impedido que llegue correctamente.

Hay que volver hacer hincapié respecto al directorio en el que se reciben los datos, por el momento está predefinido y se deja para las mejoras el poder elegirlo.

clear_remote_info()

Este método se usa cuando se quiere borrar toda la información asociada al dispositivo remoto que hay en la interfaz. Esta incluye a los ficheros compartidos, el conjunto de la dirección, el nombre y el tipo de dispositivo y la información de la barra de estado.

Esta función se suele llamar en distintas situaciones de error, como por ejemplo la anteriormente mencionada de que estábamos conectados a un dispositivo pero a la hora de la transferencia de un archivo este ha abandonado la red y ya no existe , etc..

scanIncomingCon()

Este método es bastante importante porque es el que permite, como se ha comentado ya anteriormente, mostrar en la interfaz las direcciones físicas de las conexiones entrantes que tiene un dispositivo. Es el único camino por el que el cliente y servidor de un mismo dispositivo se comunican.

El método se llama automáticamente desde la función *init()* de la clase cuando esta es instanciada. El método, inmediatamente después de ser llamado para llevar a cabo su labor, crea un hilo de ejecución que llama a otro método (***startScan()***), que es quien realiza realmente el trabajo), luego en este punto la ejecución del programa se vuelve a bifurcar.

La creación del hilo se hace con la siguiente llamada :

thread.start_new_thread(self.startScan,tuple())

Es necesario crear un nuevo hilo de ejecución porque como se ha comentado, el método *init()* es quien lo invoca. Si la ejecución no se bifurca, el método *init()* se quedaría bloqueado y no podría seguir con la carga de la interfaz gráfica, por lo tanto pararía la ejecución de la aplicación.

¿Pero porque se quedaría bloqueado? Porque la función a la que se llama, *startScan()*, consiste en un bucle *while* infinito que periódicamente va leyendo el fichero temporal de conexiones entrantes donde escribe el servidor cada vez que llega una nueva. Como este bucle no termina hasta que no se cierra la ventana de la aplicación, init() no podría continuar.

Hay que decir, que a veces no se muestra la información correctamente en la interfaz (aunque en el fichero “con.txt” siempre está correcta) y es debido a que los hilos de ejecución dan problemas cuando se lanzan desde una clase que representa a un entorno gráfico. Sería una mejora notable para la aplicación poder solventar este problema en un futuro, bien por el camino que se ha pretendido resolver o bien por otro distinto, pero de todas formas queda pendiente.

startScan()

En un primer momento esta función crea el fichero vacío antes de que el servidor pueda escribir nada. Hecho esto, la función consulta las entradas del fichero cada tres segundos por medio de un temporizador. Si encuentra información en él, borra el cuadro de texto de la interfaz donde se representan las conexiones y lo refresca con la información recién leída. Si el servidor ha anotado en el fichero alguna conexión nueva la interfaz la mostrará.

A continuación se muestra el código de este método :

```
def startScan(self):
    '"escanea conexiones entrantes"'

    f = open('con.txt','w')
    f.write('')
    f.close()
    i=1

    while 1:

        try:
            time.sleep(3)

            if self.lbxConecctions.GetCount() == 0:
                pass
            else:
                self.lbxConecctions.Clear()

            f = open('con.txt','r')

            while 1:
                data = f.readline()

                if data == '':
                    break
                else:
                    data2 = data[:-1]
                        self.lbxConecctions.Append(data2)

            f.close()

        except:
            print "EXIT SCAN"
            return
```

4.4. Interfaz gráfica

Por último nos queda presentar la pantalla principal de la aplicación. La figura 4-10 muestra el aspecto de la misma . En ella se pueden observar todos los elementos que se han ido comentando a lo largo del capítulo, los dispositivos descubiertos mediante la llamada a *inquiry()*, las conexiones entrantes, la información de los dispositivos local y remoto con el que se ha establecido una conexión, los ficheros disponibles en el dispositivo remoto, la casilla del tiempo de inquiry y los tres botones que permiten interaccionar con ella

El tiempo de búsqueda de dispositivos se puede variar en un intervalo de entre 3 y 30 segundos y está fijado por defecto a tres.

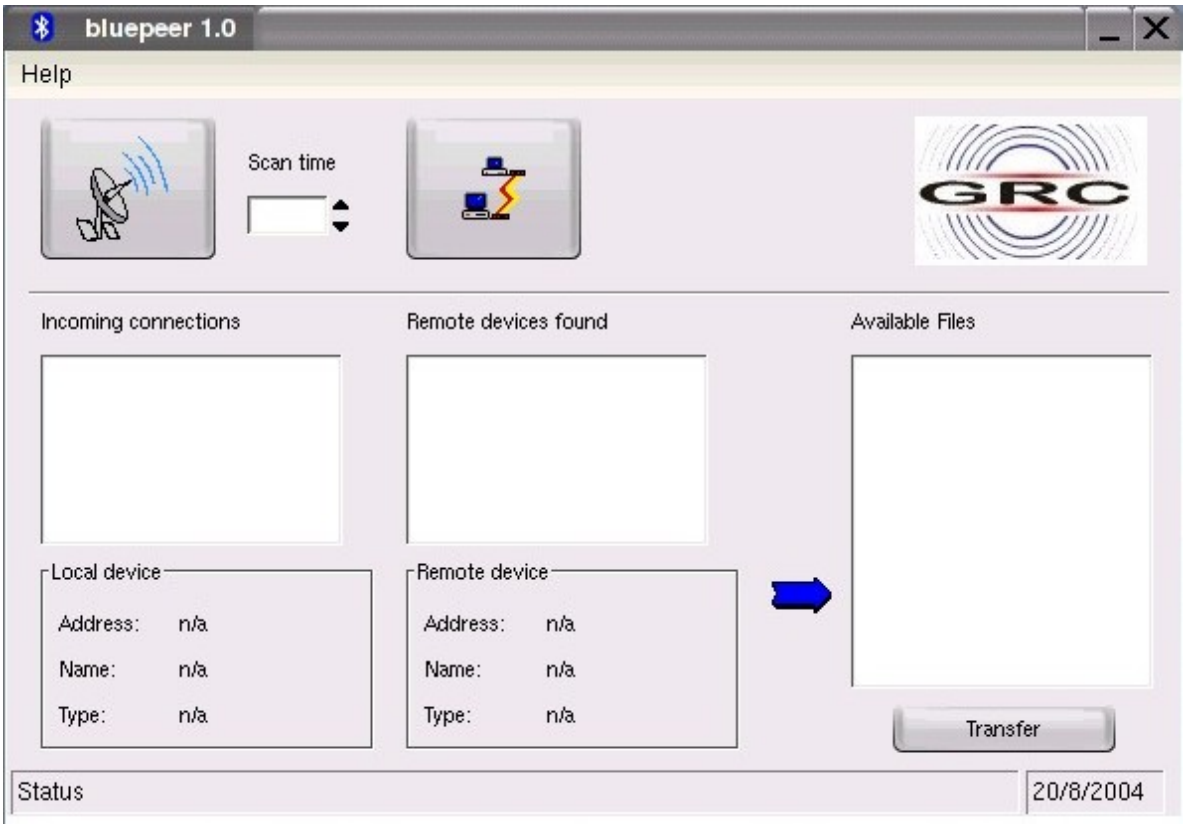


Figura 4-10 : Pantalla de la aplicación

En la barra de menús solo existe el menú *Help* , que contiene una única entrada (About) con información sobre el creador y la versión de la aplicación. Esta barra se ha creado pensando en futuras ampliaciones y mejoras de la aplicación. En la figura 4-11 se muestra la pantalla del submenú *About*.

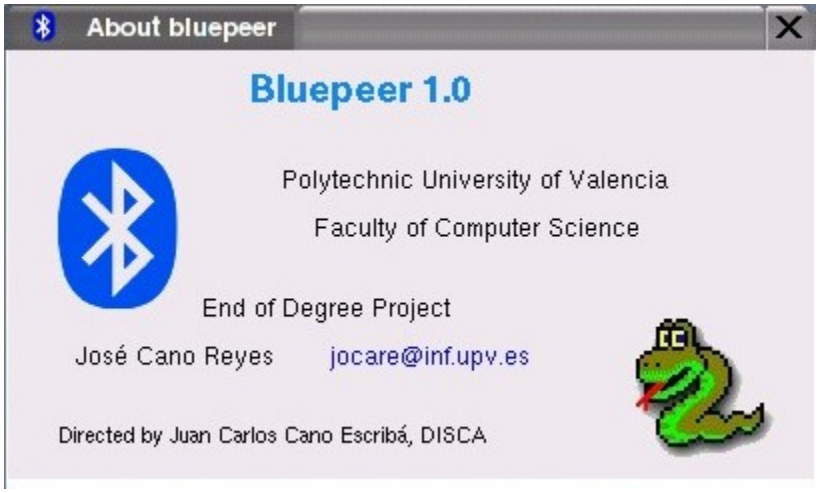


Figura 4-11 : Submenú about de la aplicación

De entre los variados programas que existen para diseñar interfaces gráficas con la librería wxWindows, se eligió ***Boa Constructor 0.2.3***, por ser uno de los de mas sencillo aprendizaje y uno de los más potentes. El único problema que planteaba su uso era que en los programas de diseño de este tipo de interfaces se busca ubicar los elementos en los formularios en base a unas cuadrículas denominadas **sizers**, además del seguimiento de una serie de patrones para todos los elementos, como por ejemplo el texto, los iconos, etc..

Esto es así para que cuando se diseñan interfaces por ejemplo para wxPython que dan lugar a programas que pueden correr sobre distintas plataformas, lo que se diseña, esté igualmente ubicado y con las mismas características independientemente de la plataforma donde se esté ejecutando la aplicación. Hay que tener en cuenta que los sistemas Windows, Linux y Mac tienen distintos tipos de fuentes de letras, distintos tamaños para estas, distintas funciones para el posicionamiento de ventanas en el monitor, etc, etc ..

La versión utilizada de Boa Constructor utilizada no cumplía lo anterior, pero como la aplicación solo está diseñada para sistemas Linux por la naturaleza de BlueZ este problema estaba resuelto.

5. Instalación

5.1 Requerimientos

Para que todo funcione correctamente hay que tener en cuenta una serie de pasos en cuanto a las versiones requeridas por los distintos elementos que intervienen en el sistema y la forma de instalar los mismos. Es muy frecuente que por tener instalada una versión más nueva o más antigua de cierto paquete la aplicación no funcione o deje de funcionar de la forma adecuada.

Como la aplicación esta hecha para sistemas Linux, en principio debe funcionar en cualquiera de sus distribuciones (Red Hat, Suse, Debian, etc...) La versión que se ha usado para el desarrollo del proyecto ha sido la Suse 9.0.

Una vez tenemos claro el sistema, es necesario tener instalado el software de BlueZ. Este software se distribuye en un conjunto de paquetes de los cuales el principal es el núcleo. En las últimas versiones de cualquier distribución Linux viene integrado en el Kernel del sistema operativo, concretamente a partir del kernel 2.4.20, pero si se está trabajando con versiones anteriores existen parches para el mismo(http://www.holtmann.org/linux/kernel).

Además del núcleo de BlueZ es necesario también tener instalado un paquetes más, bluez-libs-XX. Este paquete, proporciona el conjunto de librerias de BlueZ necesarias para la aplicación. Además, puede ser útil tener también instalado el paquete bluez-utils-XX, que aunque no es estrictamente necesario, dispone de varias herramientas interesantes. Estos dos paquetes están disponibles en http://bluez.sourceforge.net/ . Las versiones usadas en el proyecto han sido :

- bluez-libs-2.7
- bluez-utils-2.7

Tras esto, es imprescindible tener instalado un interprete de Python para poder lanzar la aplicación. Cualquiera de las versiones 2.3.X serviría aunque para el desarrollo se ha usado la 2.3-43 (http://www.python.org/).

Para el entorno gráfico son necesarias las librerias de wxPython, y aquí hay que tener mucho cuidado porque con el uso de la versión 2.5 la aplicación NO FUNCIONA. En el desarrollo se ha utilizado la versión 2.4.1. (http://wxpython.org/)

5.2 Módulos

Una vez tenemos instalado todo el entorno necesario para poder lanzar la aplicación, queda un aspecto muy importante, sin el cual, esta no funcionará. Se trata de incluir dentro de Python los dos módulos (**bluetooth** y **socketmodule**) que hemos descrito en el punto 3. Para ello hay que generarlos (a partir de sus ficheros de código C correspondientes) y

ubicarlos en un directorio adecuado de Python para que luego permita importarlos al programa (ahora veremos que se ubican de forma automática)

Para llevar a cabo esta tarea existen dos ficheros Python de setup (**setup.py**) que describen las características de las extensiones del lenguaje que se van a crear junto con los ficheros fuentes y librerías necesarias para las mismas. A continuación se muestra a modo de ejemplo el correspondiente al módulo *bluetooth* :

```
#!/usr/bin/python

from distutils.core import setup, Extension

bluetooth_module = Extension(
        'bluetooth',
        libraries = ['bluetooth'],
        runtime_library_dirs = ['/usr/lib/'],
        sources = ['py_bluetooth.c']
)

setup(
        name = "bluetooth",
        version = "1.0",
        description = "Python interface for hci functions",
        maintainer = "Jose Cano Reyes",
        maintainer_email = "jocare@inf.upv.es",
        ext_modules = [ bluetooth_module ]
)
```

Por tanto, para generar los módulos hay que ejecutar en línea de comandos de linux[5] la siguiente orden para cada módulo :

python setup.py install

Esta orden genera un directorio /build dentro del directorio actual con los módulos objeto **bluetooth.so y bluezsocket.so** que se copiaran automáticamente en el directorio

/usr/lib/python2.3/site-packages

para que puedan ser importados posteriormente desde una aplicación.

Finalmente y si todo ha ido bien ya estamos en disposición de lanzar la aplicación. Para ello nos situamos en el directorio donde esté la aplicación y escribimos el siguiente comando en línea de órdenes :

python bluepeerApp.py

[5] Hay que estar ubicados en el directorio donde se encuentra el fichero setup.py en cada caso

6. Pruebas

Las diferentes pruebas que se han realizado han consistido básicamente en lanzar la aplicación para tres dispositivos USB distintos actuando dos de ellos como clientes y el tercero como servidor. El objetivo ha sido calcular el tiempo que tardan los clientes en recibir archivos. Para ello se han variado algunos parámetros como:

- el tamaño de los archivos (representado en kbytes)
- la distancia entre los dispositivos
- el numero de clientes del servidor

La figura 6-1 muestra el esquema seguido para realizar las pruebas :

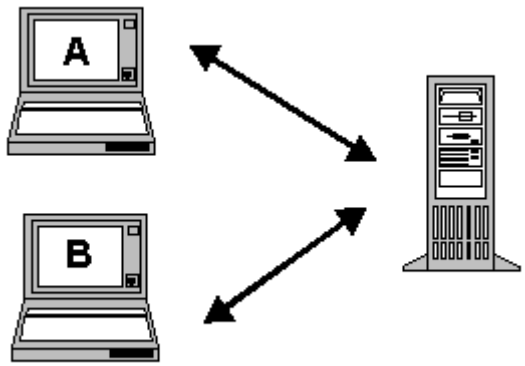
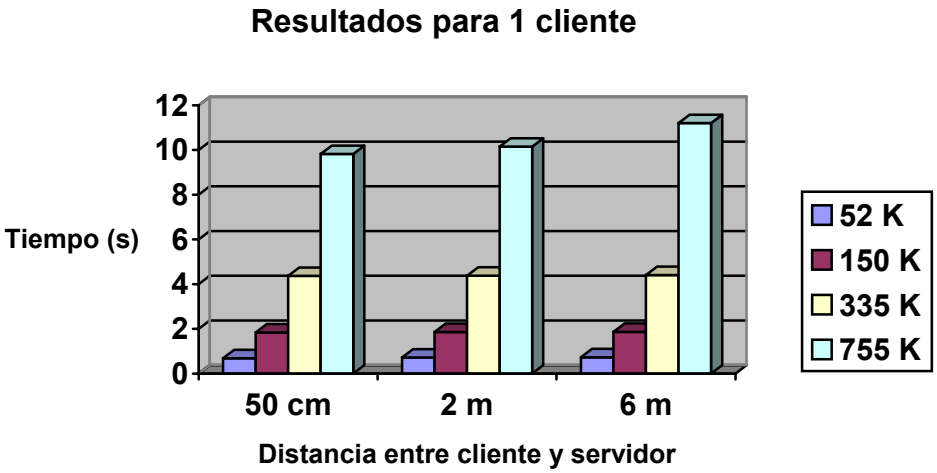
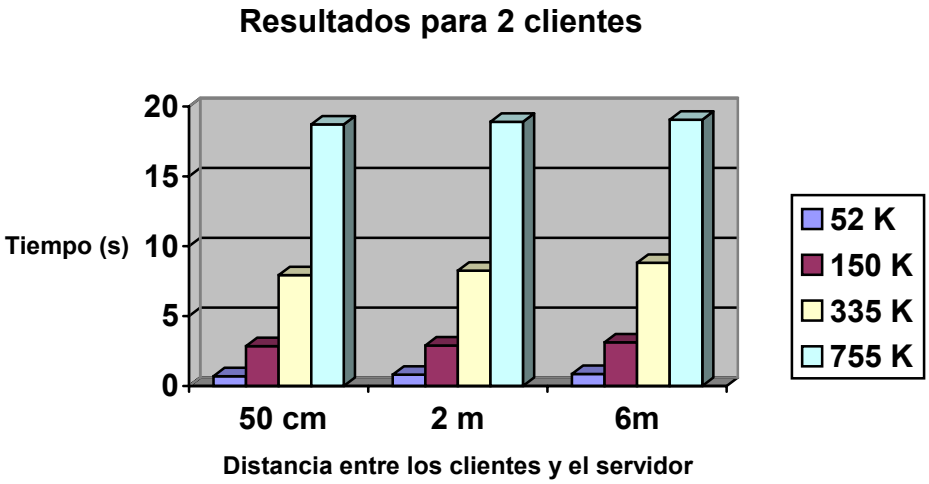


Figura 6-1 : Esquema seguido para realizar las pruebas

A continuación se muestran los gráficos de medidas de tiempos para uno y dos clientes conectados al servidor :





Al observar las gráficas se puede observar claramente que mientras aumenta la distancia, no aumenta demasiado el tiempo que tarda en transmitirse un archivo (tanto con un cliente como con dos), algo bastante lógico, ya que Bluetooth mantiene las tasas de transmisión hasta los 7 u 8 metros. A partir de esta distancia los tiempos se disparan y aunque estos se han tomado, no están representados porque no se apreciarían los valores para un tamaño de fichero de 52 kbytes.

Por otro lado también se aprecia un aumento del tiempo cuando los dos clientes están concurrentemente descargando archivos del servidor respecto a cuando solo hay uno, un hecho sin duda predecible (a mayor carga del servidor, mayores tiempos de respuesta). Los dos clientes estaban a la misma distancia del servidor conectados físicamente a un ordenador portátil. Los resultados habrían sido mas sutiles si los dos clientes se hubiesen ubicado a distancias diferentes respecto al servidor para poder comparar los tiempos entre ambos pero como solo se disponía de un portátil y un ordenador de sobremesa esta prueba no se ha podido realizar.

Por último hay que decir que al margen de la distancia a la que se esté de un dispositivo remoto influye bastante el hecho de que entre medias de ambos existan obstáculos como puertas cerradas, una pared, etc ..

7. Conclusiones

Las tecnologías inalámbricas tiene muchos atractivos debido a la creciente proliferación de las mismas. En este proyecto se ha podido estudiar en gran medida la tecnología Bluetooh y las grandes posibilidades que ofrece en distintos campos, lo cual ha sido muy gratificante. A pesar de ello, y como suele ocurrir muchas veces, no todo lo que se había pensado se ha podido llevar a la práctica, evidentemente por cuestiones de tiempo, pero se deja un amplio abanico de posibilidades de ampliación y mejora.

Una primera ampliación de la aplicación sería implementar de algún modo una base datos global que informara de todos los ficheros disponibles en la red y de quien los posee. De este modo la red ya no trabajaría como un peer-to-peer puro sino como uno híbrido. La solución a este problema es fácilmente abordable en redes cableadas pero en redes inalámbricas de corto alcance ad-hoc no lo es, ya que se podrían seguir distintos caminos para resolverlo. Además puede que no interese trabajar como red peer-to-peer sino como una scatternet Bluetooth.

En primer lugar habría que determinar quien o quienes de los dispositivos participantes en la red deberían mantener la información o si por el contrario todos deberían mantenerla. Si se opta por la primera solución habría que indicar al resto de nodos quienes son los que van a mantener la información para que puedan recurrir a ellos en su momento, lo que en principio no parece demasiado obvio. Si se opta por la segunda opción se puede entrar en problemas de excesiva redundancia e inconsistencias en la información.

Otra ampliación posible sería dotar al interfaz python-hci de mayores posibilidades agregando nuevas funciones. Hay que tener en cuenta que Bluetooth , como el resto de tecnologías punteras están en constante desarrollo y cambio, por lo que no pasa demasiado tiempo hasta que la especificación incluye funcionalidad nueva. De hecho, en la realización de este proyecto, que ha venido ha durar unos seis meses, las funciones que ofrece Bluez en esta capa han ido aumentando, agregando funciones , por ejemplo para que un nodo entre y salga del estado *parked*.

Otro tema que se puede abordar es ampliar también el módulo de sockets para dar soporte a enlaces síncronos (SCO) y para el protocolo RFCOMM.

Implementar las funciones necesarias para proporcionar una interfaz al protocolo de descubrimiento de servicio (SDP) sería también bastante interesante ya que de este modo se podrían registrar modos de actuar o privilegios especiales que podrían resolver por ejemplo el problema planteado anteriormente de la base de datos global entre otros muchos. En resumen y como se ha dicho anteriormente las posibilidades de mejora son muy amplias.

Por último, hay que decir que el uso del lenguaje Python ha valido la pena, aunque aún no tiene demasiados seguidores, es un lenguaje muy potente y de fácil aprendizaje, por lo que desde aquí lo recomiendo.

Apéndice A : The GNU General Public License

Version 2, June 1991

59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) o_er you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the

original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in e_ect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program).

Whether that is true depends on what the Program does. 1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty;

keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option o_er warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written o_er, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the o_er to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an o_er, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components ompiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by o_ering access to copy from a designated place, then o_ering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under his License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this `License.`

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to efrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may di_er in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are di_erent, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most e_ectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

> one line to give the program’s name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) yyyy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.

This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Glosario

ACK : Acknowledgement

ACL: asynchronous Connectionless (asíncrono no orientado a la conexión).

AD-HOC NETWORK : tipo de red formada por un grupo de nodos o *hosts* móviles que forman una red temporal sin la ayuda de ninguna infraestructura externa

ANSI: American National Standards Institute.

API: application program interface (Interfaz del programa de aplicación).

BD : Bluetooth Device

BD_ADDR: dirección del dispositivo Bluetooth.

BQP Bluetooth Qualification Program.

CID: identificador de canal.

CRC: chequeo de redundancia cíclica.

DRIVER: controlador que permite gestionar los periféricos que están conectados al ordenador (http://www.guiahost.com/servicios/glosario).

ESPACIO DE KERNEL: espacio de memoria ocupado por el código compilado del kernel de linux.

ESPACIO DE USUARIO: espacio de memoria ocupado por el código compilado de los programas de aplicación del usuario.

ETSI: European Telecommunications Standards Institute. Organización sin ánimo de lucro cuya misión es crear estándares de telecomunicaciones para ser usados por décadas en Europa (http://www.etsi.org).

FIRMWARE: parte del software de un ordenador que no puede modificarse por encontrarse en la ROM o memoria de sólo lectura, Read Only Memory. Es una mezcla o híbrido entre el hardware y el software, es decir tiene parte física y una parte de programación consistente en programas internos implementados en memorias no volátiles.

FHS Frequency Hopping Synchronization

GAP: generic access profile (perfil genérico de acceso).

GNU: es un acrónimo recursivo para ``no es Unix'' Se crea en 1984 sin mucho éxito, bajo la filosofía del software libre, muy al estilo de UNIX.

GOEP: generic object exchange profile (perfil genérico de intercambio de objetos).

GPL: licencia pública general (general public license), desarrollada por la FSF o Free Software Foundation. Puede ser instalado sin limitación en uno o varios ordenadores. En las distribuciones de estos programas debe estar incluido el código fuente.

HARDWARE: componentes electrónicos y electro-mecánicos de una computadora o cualquier otro sistema. Este término es usado para distinguir estos componentes físicos de los datos y programas.

HCI: host controller interface (interfaz controladora del host).

HEC: chequeo de redundancia cíclica de encabezados.

HW / FW: hardware / firmware.

IEEE Institute of Electrical and Electronics Engineers

ISM: industrial, scientific, medical**.**

KERNEL: núcleo, es la parte fundamental de un programa, por lo general de un sistema operativo, que reside en memoria todo el tiempo y que provee los servicios básicos.

L2CAP: logical link controller and adaptation protocol (protocolo de adaptación y control de enlace lógico).

LMP: link manager protocol (Protocolo del administrador de enlace).

ME: management entity (entidad de administración o manejo).

MÓDULO BLUETOOTH: módulo multichip que implementa en hardware y firmware las capas bajas del stack de protocolos Bluetooth.

MTU: maximun transmission unit.

PAD: punto de conexión del terminal de un dispositivo.

PAGING: servicio para transferencia de señalización o información en un sentido, mediante paquetes, tonos, etc…

PAN: personal area networking.

PDA Personal Digital Assistant

PPP: point to point protocol (Protocolo punto-a-punto).

QoS: calidad de servicio.

RF: radio frecuencia.

RFCOMM: serial port emulation basado en el estándar ETSI TS07.10.

SAR: segmentación y reensamblado.

SCO: synchronous connection oriented (Síncrono orientado a la conexión).

SDAP: service discovery aplication protocol (perfil de aplicación de descubrimiento de servicio).

SDP: service discovery protocol (protocolo de descubrimiento de servicio).

SIG: special interest group.

SOCKET: es una abstracción de red para los terminales de un canal. El Socket está asociado con el protocolo.

TDD Time-Division Duplex

TIMEOUT: tiempo de espera excedido.

TIMESLOT: ranura de tiempo. En Bluetooth tiene una duración de 625 us.

TRANSCEIVER: transmisor-receptor.

UART: el transmisor receptor universal asíncrono (universal asynchronous receiver transmitter), es un dispositivo que multiplexa datos paralelos en seriales para ser transmitidos y convierte en paralelos los datos seriales recibidos.

USB: la característica principal del bus serie universal (universal serial bus) reside en que los periféricos pueden conectarse y desconectarse con el equipo en marcha, configurándose de forma automática

WAP: wireless application protocol (protocolo para aplicaciones inalámbricas).

WLAN: wireless local area network

WPAN: wireless personal area network

Enlaces y referencias

[1] T. de Miguel. “Aplicaciones P2P: el nuevo paradigma

[2] M Marino. “Redes Compañero a Compañero como soporte de sistemas de archivos distribuidos”

[3] Proyecto SETI@home: http://www.setiahome.ssl.berkeley.edu/

[4] Proyectos Overnet y eDonkey2000: http://www.edonkey2000.com/

[5] Web oficial de Python : http://www.python.org/

[6] Otra web sobre python : http://starship.python.net/

[7] Web oficial de wxPython http://wxpython.org/

[7] Bluetooh Special Interest Group : http://www.bluetooth.com/

[8] Web oficial de BlueZ : http://bluez.sourceforge.net

[9] MEZOE, Cambridge Consultants Ltd. BlueStack User Manual, 2001, http://www.mezoe.com/

[10] Web oficial de AFFIX : http://affix.sourceforge.net/

[11] AXIS, Código fuente de OpenBt : http://sourceforge.net/projects/openbt/

[12] Demonstrate the PAN in Linux system. http://affix.sourceforge.net/archive/PAN/affix_pan_demo.html

[13] Heuser, Werner. Laptops, Bluetooth(TM) and Linux, 23 de Abril de 2003. http://tuxmobil.org/bluetooth_linux.html

[14] BTnod rev 2.2 project. Disponible en Internet: http://www.inf.ethz.ch/vs/res/proj/smart-its/btnode.html - sw

[16] C. Gehrmann y K. Nyberg. Enhancements to Bluetooth baseband security. En *NORDSEC'01*, Copenhague, Dinamarca, nov. 2001.

[17] T. Godfrey. Blueware: Bluetooth simulator for NS. MIT Technical Report MIT-LCS-TR-866, 2002.