Soporte Bluetooth en Linux 2.6

Doctorado URJC: Sistemas Ubicuos

Alvaro del Castillo San Félix

El presente trabajo tiene como objetivo mostrar el estado actual del soporte de Bluetooth en Linux 2.6. Para ello se introducirá los aspectos principales de la tecnología Bluetooth y se pasará a mostrar su aplicación en Linux, mostrando especial hincapié en su uso en redes personales (PAN) y la configuración que permite la transmisión de audio sobre un enlace bluetooth.

Este trabajo se enmarca dentro de la asignatura de Sistemas Ubicuos que se imparte dentro de los cursos de doctorado de la Universidad Rey Juan Carlos de Madrid.

1. Introducción a la tecnología Bluetooth

Bluetooth es una tecnología de comunicación inalámbrica que define una plataforma estandarizada en la que destaca el bajo consumo y bajo coste de sus elementos, facilitando una comunicación sin cables entre dispositivos móviles. Se incluye dentro de la especificación tanto los detalles de como se ha de llevar a cabo el enlace de comunicación entre los dispositivos, como los detalles de la capa de aplicación por la que se podrá explotar las funcionalidades de estas comunicaciones.

Bluetooth opera en el rango de radiofrecuencia de los 2.4GHz (2.400-2.4835 GHz) que no requiere licencia de uso en ningún lugar del mundo. Se guarda una banda de 2 MHz en el comienzo y 3.5 MHz final del rango para cumplir con las regulaciones de todos los países. Se usa la tecnología de transmisión de espectro amplio, con salto de frecuencia, señal full-duplex y hasta 1600 hops/s. La señal salta entre 79 frecuencias en intervalos de 1 Mhz para tener un alto grado de tolerancia a las interferencias y obtener unas comunicaciones robustas. Se dispone de comunicaciones punto a punto y multipunto, donde un dispositivo puede establecer de forma simultánea hasta 7 canales de comunicación a la vez con una sola radio.

Todos los dispositivos bluetooth tienen asociada una dirección de 48 bits, algo similar a la dirección MAC de las tarjetas de red Ethernet.

El ancho de banda que permite Bluetooth es de 1 Mbps en su funcionamiento básico, y de 2/3 Mbps en modos mejorados en Bluetooth 2.0. Se utiliza modulación GFSK (Gaussian Frecuency Shift Keying) para el modo básico, y en los modos mejorados se introducen nuevas codificaciones DQPSK y 8-DPSK.

Para lograr que la comunicación sea full-duplex se divide el tiempo de transmisión en slots con un esquema Time-Division Duplex (TDD). La cobertura es de 100 metros para los dispositivos de clase 1 (100 mW), de 20 metros para los de clase 2 (10 mW) y de 10 metros para los de clase 3 (1 mW). Lo normal es que los dispositivos actuales (año 2005) sean de clase 3.

1.1. La pila de protocolos de Bluetooth

Para poder realizar las comunicaciones entre los dispositivos Bluetooth es necesario establecer un enlace de radio entre los dispositivos. Sobre este enlace físico se irán construyendo enlaces lógicos que se ofrecerán a las aplicaciones para que puedan llevar a cabo las comunicaciones.

Estos enlaces lógicos puede ser punto a punto con tráfico síncrono, asíncrono e isócrono (basado en slots de tiempo), o enlaces de difusión. El tráfico de los enlaces lógicos se lleva al enlace físico mediante la asignación de slots de tiempo de transmisión a cada uno de los enlaces lógicos.

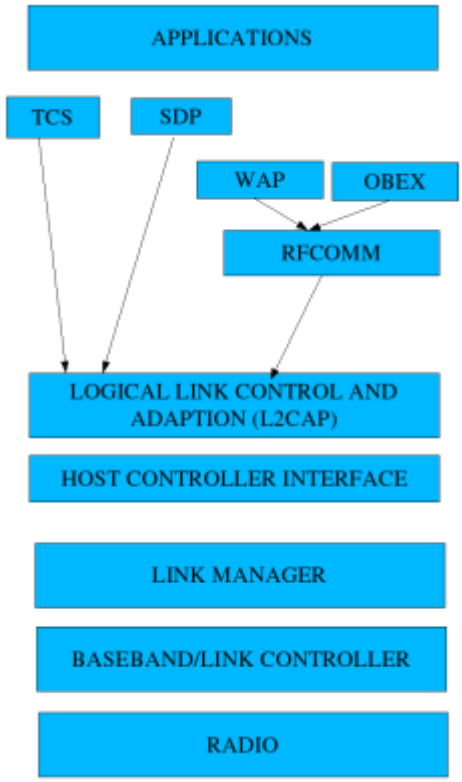
Dentro del nivel físico y de banda base existe un protocolo de comunicación entre los dispositivos de una Piconet conocido como Link Manager Protocol (LMP) que se produce sobre un canal lógico orientado a conexión asíncrono. Este protocolo es utilizado por la capa de gestión de enlaces (Link Manager) para controlar como operan los dispositivos de la Piconet, red inalámbrica de hasta ocho dispositivos, y proporciona servicios para manejar las capas de radio y banda base.

La capa L2CAP está por encima de la de banda base y que se encarga de ofrecen una abstracción de canales de comunicación a las aplicaciones y los servicios. Lleva labores como fragmentar y ensamblar paquetes con los datos de las aplicaciones, y la multiplexación de varios canales sobre un enlace lógico compartido. Un canal L2CAP es una conexión entre dos dispositivos Bluetooth.

Pasamos a entrar con algo más de detalle en cada uno de los niveles de la pila Bluetooth.

La pila que forma Bluetooth para proporcionar todas las funcionalidades es muy similar a la que propuso OSI en su torre de protocolos de referencia. Veremos que a diferencia de TCP/IP, sube por encima de la capa de transporte para definir las capas de Sesión y Presentación, dejando un entorno muy definido para el desarrollo de las aplicaciones. Ello provoca cierta complejidad a la hora de dar soporte hardware a los entornos Bluetooth, pero que su uso desde las aplicaciones sea sencillo.

Figura 1. Pila Bluetooth



En el nivel de radio se producen todos los mecanismos necesarios para la comunicación por el aire de la información. Es donde se encuentra el emisor de radio y receptor junto con sus antenas. Es en este nivel donde se produce la emisión con una determinada potencia de señal según la clase del dispositivo, que marcará el rango posible de comunicación del dispositivo: 10m, 20m o 100m. Para comunicarse los dispositivos Bluetooth utilizan un canal de comunicación radio físico compartido a una determinada frecuencia.

En el nivel de banda base (Baseband) y control de enlace (Link Controller) es donde se fijan los detalles como el direccionamiento de los dispositivos (48 bits), el reloj y la sincronización entre los dispositivos y los formatos de los paquetes a transmitir. Los datos son transmitidos en el aire divididos en paquetes con tres grandes campos: código de acceso, cabecera y datos. El código de acceso se utiliza para indicar el tipo de operación que se quiere llevar a cabo: acceso a dispositivo, canal o petición de localización. En este nivel se definen cuatro canales de transporte físicos: dos para las comunicaciones entre dispositivos de una piconet, uno para descubrimiento (inquiry) y otro para los procedimientos de conexión. Estos cuatro canales se subdividen en slots de tiempo para multiplexarlos entre todos los dispositivos que quieran hacer uso de ellos.

El gestor de enlaces (Link Manager) se encarga de controlar y configurar los enlaces con otros dispositivos, una abstracción por encima de los enlaces de transporte lógicos ACL. Por ejemplo en este nivel dos dispositivos se comunican el tipo de paquetes que soportan, si puede ser o no maestro, si tienen soporte de medición de la señal de transmisión y control de potencia o los tipos de cifrado a utilizar.

La capa de interfaz con el controlador del dispositivo radio (Host Controller Interface) ofrece una interfaz basada en comandos para poder obtener toda la información relativa a las funcionalidades que ofrece la parte de banda base del dispositivo Bluetooth. Ejemplos son las características del dispositivo (dirección), los detalles de la piconet de la que forma parte, el estado de las conexiones, los enlaces que tiene creados ...

Dentro de la capa de control de enlaces lógicos y adaptación (Logical Link Control and Adaptation, L2CAP) lo que se hace es multiplexar los enlaces que se disponen (ACL) entre todos los módulos que quieran utilizar los enlaces radio existentes. Además, es responsable de la conversión entre diferentes tamaños de paquetes y de trabajar con parámetros de calidad de servicio.

RFCOMM ofrece una interfaz de comunicación RS232 como si lo que se tuviera por debajo fuera un dispositivo serie.

WAP (Wireless Access Protocol) y OBEX (Object Exchange) proporcionan protocolos que se usarán en capas de aplicaciones para llevar a cabo labores determinadas, como por ejemplo, sincronización de datos entre dispositivos utilizando el protocolo OBEX.

TCS (Telephony Control Protocol Specification) proporciona una interfaz para el tráfico de audio.

Para terminar, SDP (Service Discover Protocolo) ofrece una interfaz para poder descubrir servicios Bluetooth en otros dispositivos que se encuentren del rango de alcance del dispositivo.

1.2. Tipos de comunicación en los enlaces Bluetooth

Unos de los objetivos de bluetooth es el de permitir transmitir audio de calidad entre dispositivos, a parte de poder servir como mecanismo de comunicación de datos entre dispositivos genérico. Para cumplir ambos objetivos se han definido dos tipos de conexiones: SCO (Synchronous Connection Oriented) y ACL (Asynchronous Conectionless).

SCO está orientado a tráfico que no se divide en paquetes y que se ha de entregar en tiempo real. Si hay pérdidas de datos, no se intentará introducir mecanismos de control de error para retransmitir. Este tipo de conexiones es el que se utiliza para el audio y no pasa por la capa L2CAP. El objetivo es mantener una latencia constante en el flujo de datos. En SCO se establece un canal bidireccional entre el maestro y el esclavo punto a punto y el maestro reserva unos slots de tiempo constantes para poder transmitir un flujo de datos de forma permanente. El ancho de banda que se reserva es de 64 kbps por conexión SCO, y se pueden tener hasta tres conexiones de este tipo entre el maestro y el esclavo. Por lo tanto a todos los

efectos este tipo de comunicaciones es de conmutación de circuitos, frente a la conmutación de paquetes de las conexiones ACL. La calidad del audio a 64 kbps se puede acercar a la de GSM pero no es adecuada por ejemplo para la reproducción de música digital. En caso de requerir mayor ancho de banda se puede optar por utilizar conexiones ACL y por ejemplo, utilizar mecanismos de buffering para salvar las latencias, como se ha hecho en la emisiones de audio por Internet. Existe una extensión de SCO llamada eSCO que tiene la opción de retransmisión de datos.

Veremos un ejemplo de uso de este tipo de comunicaciones cuando veamos el funcionamiento del sistema de cascos con micrófono bluetooth que comunicaremos con el ordenador con GNU/Linux.

Las conexiones ACL están orientadas a tráfico de datos donde lo primordial es que se entreguen de forma correcta todos los datos, aunque se produzcan importantes latencias entre la emisión y la entrega, y pueda ser una latencia variable. Se utiliza la capa L2CAP para este tipo de conexiones que incluyen control de errores y retransmisión.

Sobre las conexiones ACL viajan paquetes formados por 72 bits de código de acceso, 54 bits para la cabecera del paquete y 16 bits de código CRC, junto con los datos de usuario (payload). Hay diferentes tipos de paquetes según el número de slots de tiempo que podemos ocupar para transmitir el paquete. En la configuración donde se envía el número máximo de slots consecutivos, 5 (DH5), se obtiene el máximo rendimiento real de bluetooth 1.2, que es 723.2 kbps en L2CAP. Como los niveles por encima de este también utilizan parte de los bits de los paquetes, una cifra más real que se puede esperar de las aplicaciones finales es de 650 kbps.

Cuando mostremos el funcionamiento de las redes PAN (Personal Area Networks) veremos un ejemplo del uso de este tipo de comunicaciones, así como el ancho de banda real que podemos esperar.

1.3. Arquitectura de un grupo de dispositivos Bluetooth

Como hemos comentado, con una sola radio un dispositivo es capaz de establecer siete comunicaciones simultáneas. Esta flexibilidad se explota en la formación de las llamadas redes Piconet que están formadas por un grupo de dispositivos equipados con radios Bluetooth y que están sincronizados a un mismo reloj y siguen un mismo patrón de salto de frecuencias, patrón pseudo-aleatorio de salto entre las 79 frecuencias disponibles que se obtiene de aplicar un algoritmo concreto al identificador y reloj del dispositivo que actúa como maestro, el cual marca por lo tanto la sincronización del grupo.

Para explotar el canal de radio compartido, este se divide en slots de tiempo, en lo que se conoce como una multiplexación del canal por tiempo (Time Division Multiplexing, TDM). El tráfico de datos se divide en paquetes que se transmiten en estos slots de tiempo. Un paquete puede ocupar más de un slot de tiempo. El salto en frecuencia se produce cada vez que se transmite o recibe un paquete en el canal radio compartido. Los demás dispositivos son esclavos del maestro, y para transmitir necesitan que el maestro les autorice a hacerlo, asignándoles slots de tiempo. El número de dispositivos esclavos activos que puede tener un maestro es de siete, así que una Piconet puede tener un máximo de ocho dispositivos activos. Los esclavos no pueden establecer enlaces de radio entre sí, por lo que todo el tráfico ha de pasar

por el maestro. Además de los dispositivos activos, se pueden tener otros en estado aparcado (parked), lo que permite que en la Piconet haya más de ocho dispositivos, aunque sólo ocho pueden estar activos de forma simultánea.

Para ampliar el tamaño de una red de dispositivos bluetooth es necesario unir una Piconet con otro tipo de redes (WiFi, Ethernet ...) o con otras Piconets, formando lo que se conoce como redes "scatternets". Para formar estas "scatternets" un dispositivo bluetooth es maestro/esclavo en una piconet y esclavo en otra, haciendo de pasarela entre ambas. Un dispositivo no puede ser maestro en más de una Piconet. Los dispositivos dentro de una Piconet están sincronizados por lo que no interfieren entre ellos, pero al aparecer dispositivos con una sincronización diferente de otra Piconet, pueden comenzar a existir interferencias. Después de la transmisión de cada paquete, se cambia la frecuencia de trabajo. Si ha existido interferencia, el paquete se retransmitirá en la siguiente frecuencia donde la probabilidad de que de nuevo haya interfencia, es baja.

1.4. Perfiles en Bluetooth

La especificación de Bluetooth no se detiene únicamente en la descripción del sistema de comunicación de Bluetooth si no que describe diferentes escenarios de uso a nivel de aplicaciones, en lo que se conoce como perfiles (profiles). Veremos diferentes ejemplos de estos perfiles de uso de la tecnología dentro de las implementaciones de bluetooth de Linux que vamos a analizar.

Dentro de la especificación base de Bluetooth se incluye como parte del núcleo el perfil llamado "Generic Access Profile", que describe como se llevan a cabo los procedimientos Bluetooth de descubrimiento de dispositivos bluetooth y los aspectos de gestión de enlace entre dos dispositivos bluetooth. Además se definen también los diferentes niveles de seguridad en una comunicación bluetooth.

En este trabajo nos vamos a centrar en el perfil PAN (Personal Area Networking) que describe como dispositivos bluetooth pueden formar redes ad-hoc que se pueden utilizar a su vez para acceder a redes de infraestructura y por ejemplo, tener conectividad a Internet.

1.5. El Perfil Bluetooth PAN

Este perfil de bluetooth describe como hacer uso del protocolo BNEP (Bluetooth Network Encapsulation Protocol) para dotar a los dispositivos Bluetooth de la posibilidad de realizar comunicaciones por red.

Nosotros nos vamos a centrar en el escenario en el que tenemos un NAP (Network Access Point, Punto de Acceso de Red) con un dispositivo bluetooth, y a su vez, está conectado a una red diferente con una tecnología WiFi. Veremos dos configuraciones de este escenario: NAP haciendo NAT para dar salida a la red WiFi a los dispositivos bluetooth y NAP haciendo de bridge (puente) para que los dispositivos bluetooth aparezcan directamente dentro de la red WiFi.

PANU (PAN User) es el sistema que tiene un dispositivo bluetooth y que hará uso de NAP para poder obtener conectividad a la red de infraestructura. NAP y PANU se comunican mediante el protocolo BNEP, que se encarga en nuestro escenario de encapsular el tráfico ethernet sobre la pila de comunicaciones Bluetooth.

Los pasos a seguir serán en primer lugar iniciar el NAP para que ofrezca el servicio de PAN a aquellos dispositivos bluetooth que estén dentro de su alcance de acción. Un dispositivo PANU realizará un procedimiento de descubrimiento de servicios utilizando el protocolo SDP de bluetooth, lo que le permitirá localizar al dispositivo bluetooth del NAP.

Se crea una conexión ACL entre el PANU y en NAP y, el PANU debe de crear un canal L2CAP sobre que se utilizará para llevar a cabo las comunicaciones necesarias por el protocolo BNEP. En este momento el tráfico Ethernet ya puede fluir por esta conexión L2CAP entre PANU y NAP. Podemos ya por ejemplo configurar direcciones IP en ambos extremos de la conexión para poder comenzar a utilizar tráfico IP sobre la conexión.

2. Bluetooth en Linux

2.1. Pilas de protocolos Bluetooth para Linux

En la actualidad existen dos pilas relevantes dentro del mundo de Linux: BlueZ y Affix (Nokia). BlueZ es la pila oficial y donde existe una mayor comunidad, por lo que actualmente debería de ser la opción a seguir para trabajar con Bluetooth en Linux. Está integrada en el núcleo por lo que el despliegue de aplicaciones basadas en BlueZ es más sencillo.

2.1.1. BlueZ:

BlueZ es una pila de protocolos que comenzó su desarrollo como código abierto el 3 de Mayo de 2001 y que, un mes después de su publicación por parte de Max Krasnyansky en Qualcomm, fue adoptada por Linus Torvalds como la pila Bluetooth de Linux, lo que significó que desde 2.4.6 Linux paso a tener su propia pila Bluetooth. Marcel Holtmann se está encargando del mantenimiento de la pila en la serie 2.6 de Linux.

En Abril de 2005 recibió la Certificación Bluetooth del SIG de Bluetooth, y ya existen 5 productos baja esta certificación. El 3 de Mayo de 2005 se cumplieron 4 años desde el nacimiento de BlueZ.

Actualmente soporta los perfiles:

- Human Interface Device Profile (HIDP)

- Advanced Audio Distribution Profile (A2DP)

  Utiliza el protocolo de transporte de distribución Audio/Video (AVDTP) por debajo para llevar a cabo la comunicación.

- Hardcopy Cable Replacement Profile (HCRP)
- Service Discovery Application Profile (SDAP)
- GAP
- DUN
- FAX
- LAN
- PUSH
- SYNC
- FTP
- PAN
- CIP
- HCRP

2.1.2. Axis OpenBT

Fue la primera implementación para Linux, que vio la luz en Abril de 1999. Creada originalmente como parte del soporte Linux del punto de acceso de la compañía AXIS, los perfiles Bluetooth:que soporta esta pila son

- Acceso LAN
- Dial Up
- Comunicaciones Serie

El 14 de Abril de 2005 Anders Torbjorn Johansson anunció el final del desarrollo de esta pila y redirigió a todos los que aún la usaban a BlueZ.

2.1.3. IBM BlueDrekar

En la actualidad ya se encuentra cerrado el sitio desde donde se podía descargar por lo que se puede dar por desaparecida a esta pila.

2.1.4. Nokia Affix Bluetooth Stack

Es la pila alternativa a BlueZ que tiene algún viso de poder hacerle la competencia. Se comenzó a desarrollar antes de que se publicara BlueZ lo que puede explicar su existencia a pesar de BlueZ. Dispone de una documentación completa tanto para usar la pila como para programar aplicaciones sobre ella.

Soporta los perfiles Bluetooth:

- General Access Profile
- Service Discovery Profile
- Serial Port Profile
- DialUp Networking Profile
- LAN Access Profile
- OBEX Object Push Profile
- OBEX File Transfer Profile
- PAN Profile
- FAX Profile
- HID Profile

3. Introducción a las herramientas de BlueZ

A partir de este momento nos vamos a centrar de forma única en la pila BlueZ y las utilidades asociadas a ella. Veremos que disponemos de un conjunto bastante completo de herramientas y además, también disponemos de unas librerías de programación para poder integrar soporte bluetooth en nuestras aplicaciones.

Respecto al entorno hardware, vamos a utilizar dos equipos portátiles equipados ambos con bluetooth por medio de un dispositivo USB integrado en la misma máquina. Además utilizaremos un móvil Siemens S55 para las pruebas de comunicación con entre los ordenadores y teléfonos móviles.

Figura 2. Equipos del entorno de trabajo

En la primera máquina el hardware que tenemos es:

```
Bus 002 Device 003: ID 413c:8000 Dell Computer Corp.
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               1.10
  bDeviceClass          224 Wireless
  bDeviceSubClass         1 Radio Frequency
  bDeviceProtocol         1 Bluetooth
  bMaxPacketSize0        64
  idVendor           0x413c Dell Computer Corp.
  idProduct          0x8000
  bcdDevice            5.65
  iManufacturer           0
  iProduct                0
  iSerial                 0
  bNumConfigurations      1
```

En la segunda máquina la descripción del dispositivo Bluetooth es:

```
Bus 001 Device 002: ID 05ac:8203 Apple Computer, Inc.
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               1.10
  bDeviceClass          224 Wireless
  bDeviceSubClass         1 Radio Frequency
  bDeviceProtocol         1 Bluetooth
  bMaxPacketSize0        64
  idVendor           0x05ac Apple Computer, Inc.
  idProduct          0x8203
  bcdDevice            5.26
  iManufacturer           0
  iProduct                0
  iSerial                 0
  bNumConfigurations      1
```

Nos vamos a centrar en la distribución de GNU/Linux Ubuntu Hoary y vamos a intentar en todo momento utilizar versiones que ya se encuentren distribuidas de forma amplia, de forma que no sea necesario ningún esfuerzo especial para hacer funcionar Bluetooth para los usuarios de GNU/Linux.

Respecto a las herramientas de BlueZ la versión con la que vamos a trabajar la detallamos a continuación:

```
dpkg -l | grep bluez
ii  bluez-hcidump  1.12-1         Analyses Bluetooth HCI packets
ii  bluez-pin      0.24-1         Bluetooth PIN helper with D-BUS support
ii  bluez-utils    2.10-5ubuntu3  Bluetooth tools and daemons
```

Las herramientas básicas la principal se llama "hcitool" y es la que permite enviar configurar y enviar comandos a los dispositivos bluetooth. Uno de los comandos básicos es el de buscar los dispositivos bluetooth accesible:

```
acs@freebook:~ $ hcitool scan
Scanning ...
        00:01:E3:6B:F4:69       Acssm
        00:10:C6:2A:C8:A6       amigo-0
```

En este caso este viendo uno de los sistemas el dispositivo radio del otro sistema y el móvil con soporte bluetooth. Una vez que tenemos la dirección de un dispositivo bluetooth es cuando podemos interactuar con este. Por ejemplo podemos pedir información sobre las capacidades de un dispositivo bluetooth. En el caso del teléfono móvil:

```
root@freebook:/home/acs # hcitool info 00:01:E3:6B:F4:69
```

```
Requesting information ...

        BD Address:  00:01:E3:6B:F4:69
        Device Name: Acssm
        LMP Version: 1.1 (0x1) LMP Subversion: 0x555
        Manufacturer: Infineon Technologies AG (9)
        Features: 0xef 0xea 0x19 0x00 0x00 0x00 0x00 0x00
                <3-slot packets> <5-slot packets> <encryption> <slot offset>
                <role switch> <hold mode> <sniff mode> <RSSI> <SCO link>
                <HV3 packets> <u-law log> <A-law log> <CVSD> <transparent SCO>
```

Podemos también establecer conexiones y mostrar el nivel de la señal de una conexión:

```
# hcitool cc 00:01:E3:6B:F4:69
# hcitool con
Connections:
        < ACL 00:01:E3:6B:F4:69 handle 45 state 1 lm MASTER
        < ACL 00:10:C6:2A:C8:A6 handle 41 state 1 lm SLAVE
# hcitool rssi 00:01:E3:6B:F4:69
RSSI return value: -2

Alejando un poco el teléfono móvil

# hcitool rssi 00:01:E3:6B:F4:69
RSSI return value: -6
```

Disponemos de otras utilidades, como un analizador de tráfico L2CAP para comprobar si este tipo de tráfico fluye de forma correcta, o un ping para el protocolo L2CAP (l2ping).

```
# l2ping 00:01:E3:6B:F4:69
Ping: 00:01:E3:6B:F4:69 from 00:0A:95:2E:4B:EA (data size 20) ...
0 bytes from 00:01:E3:6B:F4:69 id 200 time 1523.03ms
0 bytes from 00:01:E3:6B:F4:69 id 201 time 36.75ms
0 bytes from 00:01:E3:6B:F4:69 id 202 time 33.71ms

# hcidump
HCIDump - HCI packet analyzer ver 1.12
device: hci0 snap_len: 1028 filter: 0xffffffff
< ACL data: handle 0x0029 flags 0x02 dlen 95
    L2CAP(d): cid 0x0040 len 91 [psm 0]
> HCI Event: Number of Completed Packets (0x13) plen 5
> ACL data: handle 0x0029 flags 0x02 dlen 17
> ACL data: handle 0x0029 flags 0x01 dlen 17
> ACL data: handle 0x0029 flags 0x01 dlen 17
> ACL data: handle 0x0029 flags 0x01 dlen 17
> ACL data: handle 0x0029 flags 0x01 dlen 17
> ACL data: handle 0x0029 flags 0x01 dlen 10
    L2CAP(d): cid 0x0040 len 91 [psm 0]
< ACL data: handle 0x0029 flags 0x02 dlen 47
    L2CAP(d): cid 0x0040 len 43 [psm 0]
```

A lo largo del artículo veremos más utilidades que cubren acciones específicas relaciones con las comunicaciones bluetooth.

4. Uso del Perfil Bluetooth PAN: Redes Personales

Como ya vimos anteriormente, con este perfil podemos construir redes que utilicen el protocolo TCP/IP sobre enlaces bluetooth. En este apartado vamos a demostrar como funciona este perfil en Linux y las posibilidad que ofrece. Para ellos vamos a utilizar los dos ordenadores portátiles que hemos utilizado hasta el momento. Uno de ellos dispone de una tarjeta inalámbrica conectado por ADSL a Internet. El otro de ellos tiene también tarjeta inalámbrica pero no funciona en GNU/Linux, por lo que está sin conexión a la red. Esta conexión la realizaremos utilizando los dispositivos Bluetooth.

Lo primero que vamos a hacer es configurar el sistema que hará el papel de NAP. Esta máquina tiene conexión inalámbrica a Internet y dará salida al sistema sin conexión por medio de Bluetooth. En ambos sistemas tenemos que habilitar el soporte del protocolo BNEP en el núcleo Linux. Para ello:

```
# modprobe bnep

Podemos ver los registros sobre la carga de este módulo:

# dmesg | tail -2
Bluetooth: BNEP (Ethernet Emulation) ver 1.2
Bluetooth: BNEP filters: protocol multicast
```

Tras ello en la máquina que hace de NAP lo único que tenemos que hacer es poner en marcha un servicio que implementa NAP.

```
# pand --listen --role NAP

Se queda un servicio, demonio Unix, iniciado en la máquina:

# ps ax | grep pand
12276 ?        Ss     0:00 pand --listen --role NAP
```

Desde el cliente de BNEP, PANU, lo que tenemos que hacer ahora es conectarnos a este servicio. Para ello necesitamos conocer la dirección bluetooth de la máquina que ofrece el servicio, algo que podemos por ejemplo lograr utilizando el perfil de descubrimiento de servicios (SDP: Services Discovery Protocol). Desde el PANU ejecutamos:

```
# sdptool browse
Inquiring ...
Browsing 00:10:C6:2A:C8:A6 ...
```

```
Service Name: SDP Server
Service Description: Bluetooth service discovery server
Service Provider: BlueZ
Service RecHandle: 0x0
Service Class ID List:
  "SDP Server" (0x1000)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 1
    Version: 0x0001
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
...
Service Name: Network Access Point
Service RecHandle: 0x804d6f0
Service Class ID List:
  "Network access point" (0x1116)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 15
  "BNEP" (0x000f)
    Version: 0x0100
    SEQ16: 800 806
Profile Descriptor List:
  "PAN access point" (0x1116)
    Version: 0x0100
```

Vemos que utilizando el protocolo SDP hemos localizado un servidor de SDP en el sistema que tenemos funcionando como NAP, y dentro de los servicios que ofrece este sistema tenemos el de "Network Access Point". Tenemos ya pues la dirección (00:10:C6:2A:C8:A6) de conexión de un sistema Bluetooth con este servicio activado y pasamos a utilizarla.

```
# pand --connect 00:10:C6:2A:C8:A6
```

A partir de este momento tenemos en los dos sistemas un nuevo interfaz de red llamado "bnep0". Les asignamos una dirección IP de la misma red a cada una de las interfaces y pasaremos a tener conexión TCP/IP sobre estas interfaces en las dos máquinas.

```
Sistema 1:
# ifconfig bnep0
bnep0     Link encap:Ethernet  HWaddr 00:10:C6:2A:C8:A6
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
# ifconfig bnep0 10.0.0.1
```

```
Sistema 2:
# ifconfig bnep0 10.0.0.2
$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=35.4 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=30.4 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=26.5 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=28.5 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=48.5 ms
...
```

Vemos que la dirección MAC Ethernet se obtiene de la dirección bluetooth de los dispositivos. En este momento tenemos ya las dos máquinas conectadas e nivel TCP/IP. Podemos ya ver datos interesantes como el retardo de los paquetes ping que es de unos 30ms de media en este caso. Pruebas más completas nos arrojan un máximo de cerca de 50ms y un mínimo de 16ms, con el retardo oscilando entre ambos sin ningún patrón claro. Este retardo aumenta con la distancia entre las dos máquinas hasta alcanzar un máximo valor de más de 3 segundos. Al alejar los sistemas dentro del margen de cobertura (10 mts) los retardos aumentan y se vuelven más cambiantes, aunque no se producen pérdidas de paquetes.

Respecto a la velocidad que proporciona la conexión, se ve muy afectada por el uso de CPU de la máquina. Si no se realiza otra tarea mientras se descarga, se logran velocidades de hasta 55.8 kB/s, que por ejemplo, es el límite habitual de las ADSL de 512 Kbps por lo que no supondría límite en esta configuración. Cuando el sistema está siendo más utilizado se logran tasas de transferencia de 22.6 kB/s. En cualquier caso es una conexión muy razonable para gestión remota de máquinas e intercambio puntual de ficheros.

Para dar salida a Internet al sistema que no la tiene, basta en una primera aproximación con activar los servicios de enmascaramiento en al máquina con conexión a Internet para dar salida a la máquina sin conexión. Para ello hemos utilizado la herramienta "Firestarter" de GNOME que facilita mucho este tipo de labores, aunque se pueden obtener resultados similares con la herramienta "iptables" directamente.

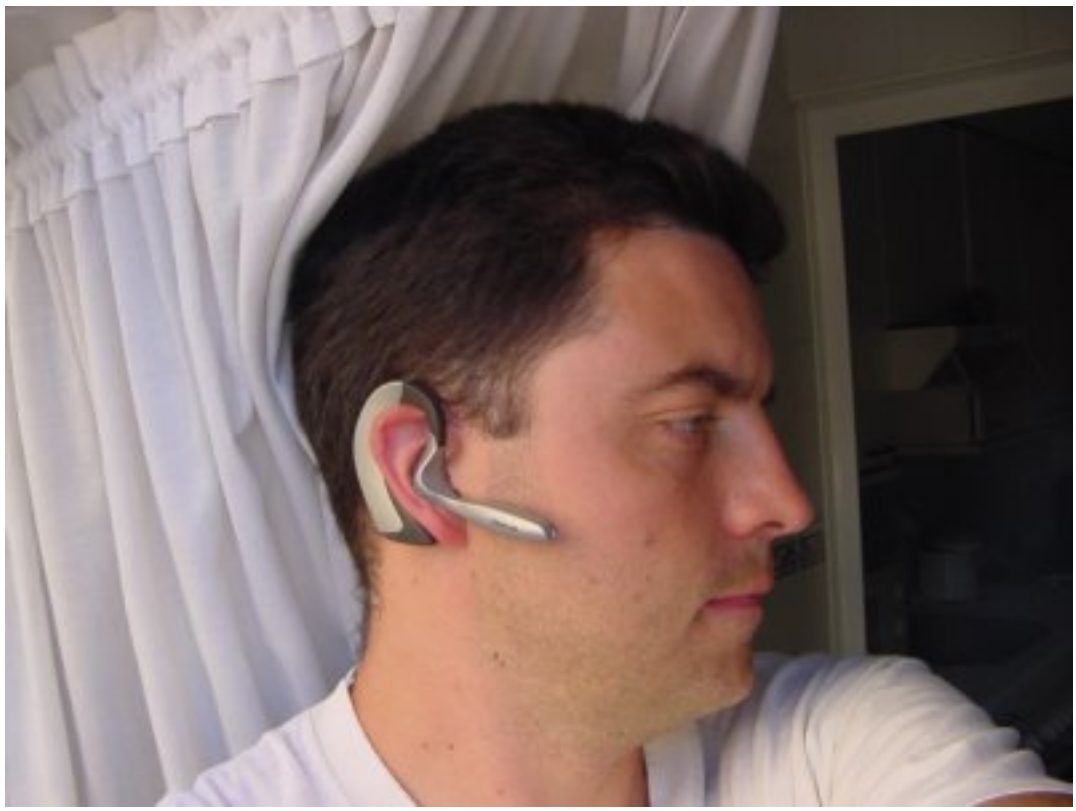
Uno de los problemas que se observan con esta configuración es que por cada dispositivo bluetooth al que se conecte el NAP, se crea una nueva red. Es decir, por defecto, en una Piconet acabaríamos con 7 redes diferentes, con un sistema en cada red. Esto crea problemas de gestión y mantenimiento al obligar a gestionar una complejidad innecesaria. La solución en GNU/Linux pasa por utilizar el sistema de "bridge" (puente) que incluye el núcleo de Linux de forma que podemos unir las 7 interfaces diferentes de red que tendría el maestro en una única red, de forma que por ejemplo, todos los dispositivos bluetooth se vean en una misma red.

5. Emisión de audio por Bluetooth

Unos de los perfiles más utilizados dentro de bluetooth es el que permite comunicar el audio entre dos

equipos. En GNU/Linux dentro del proyecto ALSA se ha desarrollado un controlador específico para que sea posible el uso de los cascos con micrófono bluetooth para el envío y recepción de audio al sistema fijo. De este modo por ejemplo, podemos caminar mientras hablamos y toda el sonido de nuestra voz se envía al ordenador. Los cascos además permiten trabajar con manos libres, de forma que se han convertido en un mecanismo ideal para la comunicación telefónica bien con los móviles o bien a través de sistemas VoIP, como Skype.

Figura 3. Cascos manos libres Bluetooth



En el entorno de pruebas vamos a comprobar que podemos reproducir música en el equipo portátil y que dicha música se escucha dentro de los cascos bluetooth. Como ya hemos comentado no es el entorno ideal del perfil de audio el de escuchar música debido a las limitaciones de calidad, pero nos permite confirmar el correcto funcionamiento. En cualquier caso el soporte de este tipo de sistemas en GNU/Linux aún está en desarrollo por lo que hay que comprobar todo muy bien antes de desplegar este tipo de configuraciones en producción. De hecho, hemos trabajado con la versión del CVS del proyecto btsco, aunque es posible que el lector encuentre ya una versión estable y soportada publicada.

Vamos a dar los pasos necesarios para activar la comunicación entre el ordenador y los cascos. Lo primero es bajar del proyecto Bluetooth-ALSA el controlador necesario para que el sistema de sonido

ALSA funcione con bluetooth. Tras ello cargamos el controlador y pasamos a conectar las radios de los dispositivos bluetooth del ordenador y de los cascos en modo síncrono. Una vez hecho esto, podemos reproducir un fichero de música dentro del ordenador y lo oiremos dentro de los cascos.

Cargamos el controlador que hemos compilado para nuestro núcleo del proyecto btsco:

```
# modprobe snd-bt-sco
# dmesg | tail -2
snd-bt-sco revision 1.6 $
snd-bt-sco: snd-bt-scod thread starting
```

Eliminamos el demonio "esd" de sonido de GNOME para que no interfiera:

```
      # killall esd
```

Configuramos el dispositivo bluetooth del ordenador para que funcione en modo síncrono:

```
      # hciconfig hci0 voice
hci0:   Type: USB
        BD Address: 00:10:C6:2A:C8:A6 ACL MTU: 192:8  SCO MTU: 64:8
        Voice setting: 0x0060 (Default Condition)
        Input Coding: Linear
        Input Data Format: 2’s complement
        Input Sample Size: 16 bit
        # of bits padding at MSB: 0
        Air Coding Format: CVSD
```

Comprobamos que los cascos bluetooth se ven de forma correcta desde el ordenador:

```
      # hcitool scan
Scanning ...
        00:08:C6:49:36:18       Nokia HS-4W
```

Dentro de "btsco" viene un programa llamado "btsco" que es el que se encarga de establecer el canal de comunicación de audio. Como vemos de sus librerías, es el que hace de puente entre bluetooth y alsa:

```
      # ldd /usr/local/bin/btsco
        linux-gate.so.1 =>  (0xffffe000)
        libbluetooth.so.1 => /usr/lib/libbluetooth.so.1 (0xb7fcc000)
        libasound.so.2 => /usr/lib/libasound.so.2 (0xb7f22000)
        libm.so.6 => /lib/tls/i686/cmov/libm.so.6 (0xb7f00000)
        libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0xb7efc000)
        libpthread.so.0 => /lib/tls/i686/cmov/libpthread.so.0 (0xb7eeb000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7dc2000)
        /lib/ld-linux.so.2 (0xb7feb000)
```

La forma de establecerse la comunicación se hace con:

```
# btsco 00:08:C6:49:36:18
```

Podemos ver que tenemos un nuevo dispositivo de audio en:

```
# cat /proc/asound/card1/id
Headset
```

Para escuchar la música en los cascos podemos por ejemplo reproducir un fichero MP3 en el equipo con:

```
# mpg123 -a /dev/dsp1 /home/acs/doc/CarlinhosBrownDjDero-MariaCaipirinha.mp3
```

y si analizamos el tráfico que se genera:

```
# hcidump
HCIDump - HCI packet analyzer ver 1.12
device: hci0 snap_len: 1028 filter: 0xffffffff
> SCO data: handle 0x002f dlen 48
> SCO data: handle 0x002f dlen 48
> SCO data: handle 0x002f dlen 48
> SCO data: handle 0x002f dlen 48
< SCO data: handle 0x002f dlen 48
< SCO data: handle 0x002f dlen 48
< SCO data: handle 0x002f dlen 48
< SCO data: handle 0x002f dlen 48
> SCO data: handle 0x002f dlen 48
...
```

Vemos que es tráfico SCO, es decir, orientado a conexión y síncrono. La calidad que se escucha en los cascos es bastante razonable y la degradación del sonido sólo se produce cuando salimos de la cobertura de la señal de radio es muy similar a la que por ejemplo, se produce en radio FM cuando salimos de cobertura.

De igual modo podemos utilizar el dispositivo de entrada que crea ALSA para grabar del micrófono que incorporan los cascos y completar así la función manos libres.

6. La API de programación de BlueZ

Junto con el conjunto de herramientas que nos permite explotar la pila de protocolos de BlueZ, el proyecto BlueZ ofrece una API de programación que permite lograr realizar usos específicos de los dispositivos bluetooth desde nuestros programas. En el ejemplo que se ha desarrollado para este trabajo el objetivo es el de medir el nivel de la señal de radio bluetooth, que viene dado por el factor RSSI. Veremos que el programa lo que hace es enviar un comando a HCI utilizando una API de programación y leer el resultado del parámetro RSSI. Es un programa de 134 líneas que se incluye de forma completa en el apéndice de este trabajo. Vamos aquí a destacar las partes principales del mismo.

Las librerías que tenemos que utilizar son a parte de las habituales de C son:

```
/* Librerias generales de sockets */
#include <sys/ioctl.h>
#include <sys/socket.h>

/* Librerias de Bluetooth */
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>
```

Para comunicarnos con el dispositivo bluetooth los desarrolladores de BlueZ han creado una abstracción basada en sockets, por lo que los comandos que enviaremos vía bluetooth los enviaremos dentro del código escribiendo y leyendo de sockets Bluetooth.

El programa principal llama al método cmd_rssi pasándole una dirección bluetooth que el usuario del programa ha debido de pasar al programa como parámetro a la hora de invocarlo. :

```
int main (int argc, char *argv[]) {
  if (argc < 2) {
    printf ("mide-rssi <btaddr>\n");
    exit (1);
  }
  printf ("Midiendo RSSI\n");
  cmd_rssi (argv[1]);
  return 0;
}

...

static void cmd_rssi (const char *baddr_txt)

{
struct hci_conn_info_req *cr;
struct hci_request        rq;
read_rssi_rp              rp;
bdaddr_t                  bdaddr;
uint16_t                  handle;
```

```
int                     dd, dev_id;
const int               numero_lecturas = 100;


str2ba(baddr_txt, &bdaddr);

dev_id = hci_for_each_dev (HCI_UP, find_conn, (long) &bdaddr);
if (dev_id < 0) {
  fprintf(stderr, "No estas conectado al dispositivo: %s\n", baddr_txt);
  exit(1);
}
```

Acabamos de comprobar que tenemos una conexión abierta con el dispositivo que queremos ver que nivel de señal tenemos. Si no es así no podemos realizar la medida y salimos del programa.

```
    dd = hci_open_dev(dev_id);
if (dd < 0) {
perror("Fallo al abrir el dispositivo con HCI");
exit(1);
}
```

Acabamos de abrir la conexión con el dispositivo para poder enviar el comando de medición de la señal.

```
      cr = malloc(sizeof(*cr) + sizeof(struct hci_conn_info));
if (!cr) {
  perror ("No se ha podido obtener memoria\n");
  return;
}

bacpy(&cr->bdaddr, &bdaddr);
cr->type = ACL_LINK;
if (ioctl(dd, HCIGETCONNINFO, (unsigned long) cr) < 0) {
perror("Fallo obteniendo la información de la conexión");
exit(1);
}

handle = htobs(cr->conn_info->handle);
```

Y ahora ya estamos en el punto en el que tenemos toda la información que necesitamos de la conexión y nos preparamos para enviar el comando HCI a través de la conexión. Vamos a enviar la petición 100 veces para poder ir moviendo el dispositivo remoto y ver como evoluciona el nivel de la señal.

```
    /* Preparamos la petición para HCI */
memset(&rq, 0, sizeof(rq));
rq.ogf    = OGF_STATUS_PARAM;
rq.ocf    = OCF_READ_RSSI;
rq.cparam = &handle;
rq.clen   = 2;
```

```
rq.rparam = &rp;
rq.rlen   = READ_RSSI_RP_SIZE;

int i;
for (i=0; i < numero_lecturas; i++) {
  sleep (1);
  /* Enviamos la peticion */
  if (hci_send_req(dd, &rq, 100) < 0) {
    perror("Fallo al leer RSSI");
    exit(1);
  }

  if (rp.status) {
    printf("Le lectura de RSSI devolvio el estado (error) 0x%2.2X\n", rp.status);
    exit(1);
  }
  printf("Valor de RSSI: %d\n", rp.rssi);
}

close(dd);
free(cr);
}
```

La API de programación es mucho más amplia pero con este ejemplo se muestran los fundamentos de su uso que no son excesivamente complejos. A la hora de compilar la aplicación hay que enlazar con la librería de bluetooth:

```
gcc mide-rssi.c -g -Wall -O2 -o mide-rssi -lbluetooth
```

Probando a medir el nivel de RSSI con el teléfono móvil desde el ordenador moviendo el teléfono móvil obtenemos los valores:

```
 # ./mide-rssi 00:01:E3:6B:F4:69
Midiendo RSSI
Numero de conexiones: 2
Direccion de dispositivo remoto: 00:01:E3:6B:F4:69
Valor de RSSI: 0
...
Valor de RSSI: -1
Valor de RSSI: -1
Valor de RSSI: -1
Valor de RSSI: -1
Valor de RSSI: -2
...
Valor de RSSI: -1
Le lectura de RSSI devolvió el estado (error) 0x02
```

La última lectura es cuando salió de rango el dispositivo. La verdad es que las pruebas nos indican que este parámetro no es muy útil ya que no parece tener gran precisión.

7. Uso de Bluetooth en el escritorio(GNOME)

En esta última parte previa a las conclusiones, vamos a analizar el estado actual de explotación de Bluetooth desde un escritorio para Linux, GNOME en este caso, y veremos que posibilidades se le ofrecen al usuario para hacer uso de la tecnología Bluetooth.

Nos vamos a centrar en el intercambio de ficheros entre el teléfono móvil equipado con bluetooth y el ordenador. Actualmente se está realizando trabajo en varios frentes: sincronización de datos, manejo de la interfaz gráfica desde el móvil o envío y recepción de ficheros. En esta sección nos centramos en este último caso.

El software que vamos a utilizar para estas pruebas se conoce como gnome-bluetooth y la versión que utilizamos es de nuevo, la que se incluye en Ubuntu Hoary:

```
ii  gnome-bluetooth     0.5.1-1ubuntu7      GNOME Bluetooth tools.
```

En primer lugar vamos a realizar el envío de datos desde el teléfono móvil hacía el ordenador. Para ello tenemos que habilitar el servicio de bluetooth en GNOME, algo que se consigue desde el menú de "Herramientas de Sistemas" arrancando la aplicación "Bluetooth Filesharing". Al hacerlo, nos aparece un icono nuevo en el panel de GNOME. En ese momento podemos ya ir a nuestro teléfono móvil y enviar un fichero, en este ejemplo un pequeño fichero.

Figura 4. Confirmar que aceptamos los datos por bluetooth

Figura 5. Datos recibidos por bluetooth

Figura 6. La imagen dentro del escritorio

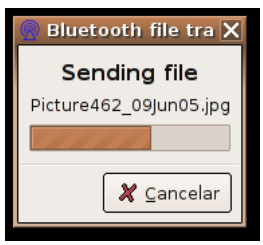
En el envío del escritorio al teléfono móvil aún se está desarrollando en GNOME la tecnología "SendTo" que permite enviar objetos del escritorio a diferentes sistemas, entre ellos bluetooth. Mientras se termina de integrar este tecnología, posiblemente para GNOME 2.12 o 2.14, hay que utilizar la utilidad "gnome-obex-send" desde la línea de comandos:

```
# gnome-obex-send Picture462_09Jun05.jpg
Browsing 00:01:E3:6B:F4:69 ...
Service Name: OBEX Object Push
Service RecHandle: 0x11105
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 4
  "OBEX" (0x0008)
```

```
Service Class ID List:
  "OBEX Object Push" (0x1105)
** Message: device 00:01:E3:6B:F4:69 (OBEX Object Push) port 4
```

La selección del dispositivo remoto se realiza mediante una ventana que nos muestra todos los equipos bluetooth conocidos y nos permite localizar nuevos dispositivos.

Figura 7. Envío de un fichero por bluetooth



8. Conclusiones

A lo largo del presente trabajo hemos presentado la tecnología de comunicación por radio Bluetooth, muy madura y que ya está ampliamente implantada dentro de la industría y ha llegado de forma masiva a los consumidores.

Hemos destacado las virtudes principales de bluetooth: dispositivos muy baratos de construir, un consumo muy reducido y una alta inmunidad a las interferencias radio. Junto a estas tres características, el que el uso a nivel de aplicación esté también estandarizado, ha ayudado a que a día de hoy dispongamos ya diversos escenarios, como el de los cascos bluetooth, que han llegado hasta los consumidores. Hemos visto que quizá su punto más débil sea el ancho de banda máximo que permite y la cobertura, campos en los que 802.11abg (redes wireless) son superiores.

Centrándonos en el uso de la tecnología en GNU/Linux, la conclusión es que se dispone de un soporte completo gracias a la pila BlueZ que se encuentra dentro del propio núcleo Linux, por lo que cualquier sistema GNU/Linux con un núcleo superior al 2.4.6 tiene ya incorporado el soporte bluetooth.

El explotar el perfil de redes de area personal (PAN) en Linux, y lo poco costoso que es el tener conectividad entre dos equipos con la adquisición de dos dispositivos USB Bluetooth abre unas grandes posibilidades de conectividad en sistemas GNU/Linux.

El perfil de audio, a pesar de que le queda un poco de madurez, la funcionado perfectamente en las pruebas que hemos realizado, por lo que aplicaciones que necesiten enviar y recibir audio del usuario, pueden ya hacer uso de las posibilidades que ofrece este perfil. Se están ya comenzando a comprar los cascos bluetooth para utilizar programas de VoIP como Skype, haciendo que su uso sea mucho más cómodo. Y es muy previsible que en centros de soporte telefónico se implanten de forma masiva estos aparatos con sistemas VoIP.

Para finalizar, hemos visto que en el escritorio GNOME ya tenemos una importante integración con Bluetooth a la hora de intercambiar información con dispositivos bluetooth remotos, aunque aún faltan unos meses para que el soporte sea completo.

Como cierre, GNU/Linux está dotado de un soporte bluetooth maduro, robusto y completo y constituye una plataforma ideal para desarrollar soluciones que exploten esta tecnología.

9. Referencias

- Bluetooth Special Interest Group (http://www.bluetooth.com/)
- La pila oficial de protocolos Bluetooth de Linux (http://www.bluez.org/)
- Principios de diseño de BlueZ (http://mhonarc.axis.se/bluetooth-dev/msg01881.html)
- Axis OpenBT (http://developer.axis.com/software/bluetooth/)
- IBM BlueDrekar (http://www.alphaworks.ibm.com/tech/BlueDrekar/)
- Asignatura Sistemas Ubicuos, Programa de Doctorado en Informática y Modelización Matemática, Universidad Rey Juan Carlos. (http://gsyc.escet.urjc.es/docencia/asignaturas/doct-computacion-ubicua/)
- Advanced Audio Distribution Profile (A2DP) para BlueZ (http://www.holtmann.org/linux/bluetooth/audio.html)
- http://bluetooth-alsa.sourceforge.net/ (http://bluetooth-alsa.sourceforge.net/)
- Bluetooth Scatternets (http://www.cs.ucla.edu/NRL/hpi/tcpw/ tcpw_papers/bluetooth-icc2003.pdf)
- IP sobre Bluetooth, BOF de trabajo en 2000, Transparencias (http://www.ietf.org/proceedings/00jul/SLIDES/ipobt-agenda/index.htm)
- Capítulo de Ejemplo (http://vig.pearsoned.com/samplechapter/0130661066.pdf) sobre Bluetooth del libro de Person Education Bluetooth 1.1: Connect Without Cables, 2nd edition.

- Seguridad en Bluetooth, Artículo de Juha T. Vainio de Helsinki University of Technology, 2000 (http://vig.pearsoned.com/samplechapter/0130661066.pdf)

A. Código del programa de lectura del nivel de señal Bluetooth

```
/* Librerias habituales de C */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

/* Librerias generales de sockets */
#include <sys/ioctl.h>
#include <sys/socket.h>

/* Librerias de Bluetooth */
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>

/* Buscar conexion Bluetooth */
static int find_conn (int  s,
      int  dev_id,
      long arg)
{
struct hci_conn_list_req *cl;
struct hci_conn_info     *ci;
int i;
char addr[18];

if (!(cl = malloc(10 * sizeof(*ci) + sizeof(*cl)))) {
perror ("No se puede obtener meemoria");
exit (1);
}

cl->dev_id = dev_id;
/* Se miran un máximo de 10 conexiones */
cl->conn_num = 10;
ci = cl->conn_info;

if (ioctl(s, HCIGETCONNLIST, (void*)cl)) {
perror("No se puede obtener el listado de conexiones");
exit(1);
}

printf ("Numero de conexiones: %d\n", cl->conn_num);
for (i=0; i < cl->conn_num; i++, ci++) {
  ba2str (&ci->bdaddr, addr);
  printf ("Direccion de dispositivo remoto: %s\n", addr);
```

```
  if (!bacmp((bdaddr_t *) arg, &ci->bdaddr)) {
    return 1;
  }
}
return 0;
}


/* Leer RSSI */

static void cmd_rssi (const char *baddr_txt)

{
struct hci_conn_info_req *cr;
struct hci_request        rq;
read_rssi_rp              rp;
bdaddr_t                  bdaddr;
uint16_t                  handle;
int                       dd, dev_id;
const int                 numero_lecturas = 100;


str2ba(baddr_txt, &bdaddr);

dev_id = hci_for_each_dev (HCI_UP, find_conn, (long) &bdaddr);
if (dev_id < 0) {
  fprintf(stderr, "No estas conectado al dispositivo: %s\n", baddr_txt);
  exit(1);
}

dd = hci_open_dev(dev_id);
if (dd < 0) {
perror("Fallo al abrir el dispositivo con HCI");
exit(1);
}

cr = malloc(sizeof(*cr) + sizeof(struct hci_conn_info));
if (!cr) {
  perror ("No se ha podido obtener memoria\n");
  return;
}

bacpy(&cr->bdaddr, &bdaddr);
cr->type = ACL_LINK;
if (ioctl(dd, HCIGETCONNINFO, (unsigned long) cr) < 0) {
perror("Fallo obteniendo la información de la conexión");
exit(1);
}

handle = htobs(cr->conn_info->handle);

/* Preparamos la petición para HCI */
memset(&rq, 0, sizeof(rq));
```

```
rq.ogf    = OGF_STATUS_PARAM;
rq.ocf    = OCF_READ_RSSI;
rq.cparam = &handle;
rq.clen   = 2;
rq.rparam = &rp;
rq.rlen   = READ_RSSI_RP_SIZE;

int i;
for (i=0; i < numero_lecturas; i++) {
  sleep (1);
  /* Enviamos la peticion */
  if (hci_send_req(dd, &rq, 100) < 0) {
    perror("Fallo al leer RSSI");
    exit(1);
  }

  if (rp.status) {
    printf("Le lectura de RSSI devolvio el estado (error) 0x%2.2X\n", rp.status);
    exit(1);
  }
  printf("Valor de RSSI: %d\n", rp.rssi);
}

close(dd);
free(cr);
}

int main (int argc, char *argv[]) {
  if (argc < 2) {
    printf ("mide-rssi <btaddr>\n");
    exit (1);
  }
  printf ("Midiendo RSSI\n");
  cmd_rssi (argv[1]);
  return 0;
}

/* compile: gcc mide-rssi.c -g -Wall -O2 -o mide-rssi -lbluetooth */
```