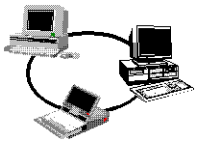
Redes de Área Local e Interconexión de Redes



Práctica 4

Bluetooth y la pila de protocolos BlueZ

Introducción

Bluetooth es la especificación técnica de una nueva tecnología de comunicación inalámbrica, que posibilita la transmisión de voz y datos entre diferentes equipos mediante un enlace por radiofrecuencia. El principal objetivo que se pretende conseguir con esta norma es facilitar la comunicación entre equipos móviles y fijos, eliminando cables y conectores entre éstos. Está previsto adoptar a Bluetooth como el estándar IEEE 802.15 dentro del grupo PAN (Personal Area Networks) de la organización IEEE.

La tecnología Bluetooth comprende hardware, software y mecanismos de interoperabilidad, por lo que para su desarrollo ha sido necesaria la participación de los principales fabricantes de los sectores de las telecomunicaciones e de la informática. Las marcas que han participado son principalmente Ericsson, Nokia, Toshiba, IBM e Intel. Posteriormente, empresas de otros sectores (tales como fabricantes de juguetes, ocio, electrodomésticos) han ido incorporándola a sus productos, debido a sus características de bajo coste y consumo.
Bluetooth se utiliza para la creación de pequeñas redes de comunicación inalámbrica, llamadas **piconets**, con lo que cualquier equipo móvil podrá intercambiar información con otro. Pero además de cámaras, PDAs, fijos y portátiles, sus características de bajo coste y consumo la hacen adecuada para un gran número de equipos de otros sectores (automóviles, juguetes, maquinaria...), con lo que se abre un horizonte de posibilidades muy amplio.
En un documento a parte en la web de la asignatura, podrás encontrar más información acerca de esta tecnología.

Objetivos de la práctica

En esta práctica se quieren llevar a cabo experimentos con dispositivos Bluetooth USB en Linux, para poder conocer directamente el uso de esta tecnología y sus posibilidades.

BlueZ es la pila de protocolos Bluetooth oficial de Linux. Fue desarrollada por Qualcomm y actualmente está disponible para uso público bajo licencia GPL. BlueZ puede descargarse desde **http://bluez.sourceforge.net**. BlueZ ofrece las siguientes funcionalidades:

- Arquitectura flexible, eficiente y modular.

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

Instalación y compilación

BlueZ se distribuye en un conjunto de paquetes, aunque el núcleo depende de la distribución del núcleo Linux que estemos usando. El uso de la Suse 8.2 o superior, evita utilizar algún tipo de parche, pues incluye la versión del núcleo 2.4.20, con la funcionalidad Bluetooth ya incorporada. Para versiones anteriores existen parches en la página web de Marcel Holtmanns (http://www.holtmann.org/linux/kernel).
Además del soporte del núcleo, los paquetes que pueden usarse en función de las necesidades finales del usuario son:

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

La versión actual KUBUNTU del laboratorio ya debe de tener las librerías instaladas.

Con el fin de tener los privilegios necesarios para llevar a cabo todos los ejercicios de la práctica, ejecuta en primer lugar desde una consola:

```
Linux:~ $ sudo su
```

Proporcionando como contraseña la correspondiente a tu cuenta de usuario.

Podemos ahora utilizar el dispositivo Bluetooth. Insértalo en un puerto USB de la parte frontal de tu ordenador.

Ejecuta ahora:

```
Linux:~ # lsmod | grep blue
```

obtendrás algo parecido a

```
Bluetooth      27108   1   [hci_usb]
```

El dispositivo Bluetooth, que en nuestro caso es un adaptador USB de clase 2, ya debería estar activado. Para comprobarlo ejecuta:

```
Linux:~ # hciconfig
```

O bien, en caso de no ser root: `Linux:~ $ sudo hciconfig`
Proporcionando la contraseña de la cuenta de usuario empleada cuando se solicite.

aparecerá algo del tipo:

```
hci0:   Type: USB
BD Address: 00:02:72:40:25:B5 ACL MTU: 192:8 SCO MTU: 64:8
UP RUNNING PSCAN ISCAN
RX bytes:69 acl:0 sco:0 events:8 errors:0
TX bytes:27 acl:0 sco:0 commands:7 errors:0
```

Observa que dice “UP RUNNING”, si por el contrario dice “DOWN” puedes activarlo mediante el commando:

```
Linux:~ # hciconfig hci0 up
```

O bien, en caso de no ser root: `Linux:~ $ sudo hciconfig hci0 up`

Herramientas de Bluez

Los paquetes instalados ofrecen las siguientes herramientas: /sbin/**hcid**, /sbin/**hciconfig**, /bin/**hcitool**, /bin/**l2ping** y el programa **hcidump**.
A continuación se presenta una descripción de las diferentes opciones de las diferentes herramientas:

hcid

Este es demonio que se encarga de gestionar los dispositivos Bluetooth. En /etc/bluetooth encontraras el fichero **hcid.conf** que permite definir los parámetros básicos de los dispositivos.

hcidump

Es una herramienta que visualiza en pantalla todos los paquetes recibidos y enviados por un dispositivo especifico. Es particularmente útil cuando se quiera analizar el funcionamiento de un dispositivo o depurar a bajo nivel posible problemas de protocolos de comunicación.

```
HCIDump - HCI packet analyzer ver 1.5
Usage: hcidump [OPTION...] [filter]
  -i, --device=hci_dev       HCI device
  -p, --psm=psm              Default PSM
  -s, --snap-len=len         Snap len (in bytes)
  -r, --read-dump=file       Read dump from a file
  -w, --save-dump=file       Save dump to a file
  -a, --ascii                Dump data in ascii
```

```
  -x, --hex                 Dump data in hex
  -R, --raw                 Raw mode
  -t, --ts                  Display time stamps
  -?, --help                Give this help list
      --usage               Give a short usage message
```

hcitool

hcitool es la herramienta principal de la pila Bluez. Ofrece servicios básicos como realizar un inquiry, una conexión, obtener información sobre un dispositivo remoto y varios otros. A continuación puedes ver la ayuda en línea:

```
hcitool - HCI Tool ver 2.2
Usage:
        hcitool [options] <command> [command parameters]
Options:
        --help  Display help
        -i dev  HCI device
Commands:
        dev     Display local devices
        inq     Inquire remote devices
        scan    Scan for remote devices
        name    Get name from remote device
        info    Get information from remote device
        cmd     Submit arbitrary HCI commands
        con     Display active connections
        cc      Create connection to remote device
        dc      Disconnect from remote device
        cpt     Change connection packet type
        rssi    Display connection RSSI
        lq      Display link quality
        lst     Set/display link supervision timeout
```

hciconfig

Permite llevar a cabo todas las operaciones de configuración. Es utilizada fundamentalmente para activar y desactivar un dispositivo y para obtener o modificar todos los parámetros de funcionamiento. A continuación se reproduce la ayuda en línea:

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

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

l2ping

Permite comprobar la calidad de un enlace de la misma manera que el comando estándar ping

```
  NAME
    l2ping - Send L2CAP echo request and receive answer
  SYNOPSIS
    l2ping [ -S source addr ] [ -s size ] [ -c count ] [ -f ] <bd_addr>
  DESCRIPTION
         L2ping sends a L2CAP  echo  request  to  the  Bluetooth  MAC
         address bd_addr given in dotted hex notation.
  OPTIONS
         -S source addr
            Select address to be used as source address for the request.
         -s size
            The size of the data packets to be sent.
         -c count
            Send count number of packets then exit.
         -f
            Kind  of  flood ping. Use with care! It reduces the delay
            time between packets to 0.
         bd_addr
            The Bluetooth MAC address to be pinged in dotted hex
            Notation like 01:02:03:ab:cd:ef or 01:EF:cd:aB:02:03
```

Ejercicios:

Utilizaremos ahora las herramientas antes descritas para realizar algún experimento con los dispositivos Bluetooth. Puedes realizar los ejerciciso activando en una ventana el **hcidump** de manera que podrás ver los mensajes de bajo nivel intercambiados.

Nota: Si el hcidump no está instalado deberás instalarlo desde el “Adept Manager” (menú principal opción Sistema).

Antes de todo intenta buscar a otros dispositivos Bluetooth que estén cerca de tu máquina.

```
linux:~ # hcitool inq
Inquiring ...
00:10:60:A2:83:A9     clock offset: 0x4a0b    class: 0x000000
```

puedes ahora utilizar el comando **l2ping** para verificar la conectividad con algunos de los dispositivos detectado.

Por ejemplo:

```
linux:~ # l2ping 00:10:60:A2:83:A9
Ping: 00:10:60:A2:83:A9 from 00:10:60:A2:83:AB (data size 20)
20 bytes from 00:10:60:A2:83:A9 id 200 time 34.06ms
20 bytes from 00:10:60:A2:83:A9 id 201 time 32.81ms
20 bytes from 00:10:60:A2:83:A9 id 202 time 36.62ms
20 bytes from 00:10:60:A2:83:A9 id 203 time 33.43ms
4 sent, 4 received, 0% loss
```

Ejercicio 1:

Utilizando hciconfig deshabilita el inquiry scan y comprueba que ahora tu dispositivo es transparente para los demás.

Ejercicio 2:
Utilizando y comparando los mensajes obtenidos con el hcidump determina a que secuencia de ordenes hcitool corresponde las orden "hcitool scan"

Ejercicio 3:
Estudia la configuración del fichero "/etc/bluetooth/hcid.conf" y modifica alguno de los parámetros, por ejemplo el nombre del dispositivo, para evaluar como se modifica el comportamiento del dispositivo. Acuérdate de re-ejecutar el hcid cada vez que modificas su fichero de configuración (hcid restart)**.**

Ejercicio 4:
Modifica el tamaño del paquete en 256 y 512 bytes y comprueba como varían los tiempos.

Ejercicio 5:
Intenta sincronizarte con tus compañeros cercano para repetir el ejercicio anterior de forma que haya solamente una transmisión a la vez o que todos estén intentando transmitir a la vez. ¿Observas algún cambio? ¿Por qué?

Ejercicio 6:
Repite el ejercicio 4 pero primero con un dispositivo cercano y luego con un dispositivo que este a cerca de 10 metros de ti. ¿Observas algún cambio? ¿Por qué?

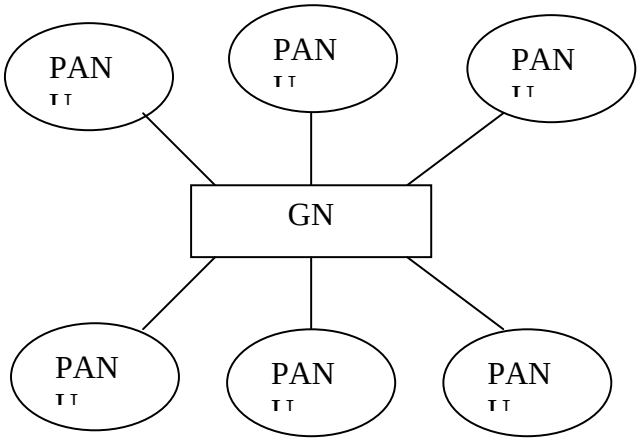
Uso del profile PAN

Versión adaptada del documento http://bluez.sourceforge.net/contrib/HOWTO-PAN*, autor Michael Schmidt, University of Siegen, Germany (schmidt@nue.et-inf.uni-siegen.de)*

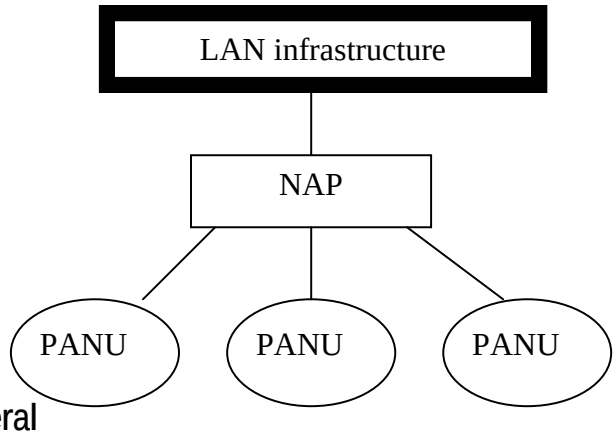
El *profile* PAN ofrece, entre otras, la posibilidad de soportar IP sobre L2CAP. Los participantes de una PAN puede actuar como:

1) usuario PAN (**PANU**): son los clientes de un NAP o de un GN, explicados a continuación.

2) controlador de una *Group ad-hoc Network* (**GN**): nodo que se encarga de hacer *forwarding* en una red de tipo *peer-to-peer* (una piconet). Permite interconectar hasta 7 dispositivos activos (PANUs) en una red *peer-to-peer* real.



3) Network Access Point (**NAP**): actúa como un proxy, router o puente entre una red fija (una LAN) y hasta 7 PANUs activos.



Setup general

Comprueba que el demonio HCI ('hcid') está ejecutando y que el dispositivo está activo ("up"). Tienes ahora que cargar el modulo bnep:

```
linux:~ # modprobe bnep
linux:~ # lsmod | grep blue
bluetooth                 29548   1  [bnep hci_usb]   (o algo
parecido)
```

- Ahora hay que activar el demonio PAN ('pand').

La idea es que 'pand' se tiene que activar como server en un lado del canal, utilizando el parámetro '-s'. Hay que observar que en el lado servidor el dispositivo tiene que poder funcionar como master. Por eso, hay que habilitar la conmutación master/slave en el fichero de configuración '/etc/bluetooth/hcid.conf':

```
lm accept, master
```

Un nodo puede ser configurado como master también de forma directa:

```
linux:~ # pand -s
```

Para establecer una conexión entre un PANU y un GN o un NAP hay que ejecutar, en el otro nodo:

```
linux:~ # pand -c <destination BT device address>
```

Ejemplo y pasos a seguir:

para conectar un PANU (00:37:5C:67:D3:01) con un GN (00:37:5C:67:D3:02), los dos dispositivo tienen que estar up, el modulo bnep tiene que estar cargado y el GN tiene que poder funcionar como master.

Tiene que ejecutar en el GN (server):

```
linux:~ # pand -s
```

y en el PANU:

```
linux:~ # pand -c 00:37:5C:67:D3:02
```

Eventuales problemas se reflejarían en los ficheros de log /var/log/messages o /var/log/warn.

Una vez establecida la conexión, generaremos un interfaz virtual “bnep0” en ambos dispositivos, por ejemplo en el nodo maestro ejecutaremos:

```
linux:~ # ifconfig bnep0 192.168.0.1 netmask 255.255.255.0
linux:~ # ifconfig bnep0 up
```

y en el esclavo:

```
linux:~ # ifconfig bnep0 192.168.0.2 netmask 255.255.255.0
linux:~ # ifconfig bnep0 up
```

creando una red IP privada en el rango de direcciones 192.168.0.x. Ahora es posible utilizar el ping para comprobar la calidad del enlace.

El interfaz que hemos creado puede ser configurado como cualquier otro dispositivo de red utilizando el comando “ifconfig”.

Ejercicio 7
Conecta y configura dos ordenadores utilizando PAN y el procedimiento arriba descrito.

Ejercicio 8:
Calcula el throughput del canal usando el comando ping, compáralo con los valores obtenidos con el comando l2ping en los ejercicios anteriores. ¿Hay alguna diferencia?

Desconecta ahora el cable de red del nodo PANU. Haz ahora un ping al router de la UPV (dirección IP 158.42.1.10) desde dicho nodo.

Ejercicio 9(a):
El ping no funciona. ¿Sabrías justificar el por qué?

El GN tiene que ser utilizado como gateway por el dispositivo PANU. Ejecuta en el GN la siguiente secuencia de comandos:

```
linux:~ # echo "1" > /proc/sys/net/ipv4/ip_forward
linux:~ # iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Esta secuencia activa en el GN los servicios de forwading y de NAT (Network Address Translation).

Intenta otra vez con el ping al router de la UPV (dirección IP 158.42.1.10).

Ejercicio 9(b):
El ping sigue sin funcionar. ¿Sabrías decir por qué y hacerlo funcionar? (una pista: usa el comando route de linux)

Ejercicio de programación

En el fichero **RALFI_P5.tar.gz** puedes encontrar el file simple_l2test_vBASE.c. Este fichero contiene una versión simplificada de la herramienta l2test. Esta permite la transmisión de datos usando sockets L2CAP entre dos dispositivos. Permite cambiar la MTU y otros parámetros relativos a la conexión, como se muestra en la tabla a continuación.

```
l2test <mode> [-b bytes] [-P psm] [-I imtu] [-O omtu] [bd_addr]
  <mode>    Se aceptan las siguientes opciones:
       -r : Receive (Servidor)
       -s : Send (Cliente)
  -I : MTU de entrada que aceptamos
  -O : Mínima MTU de salida que necesitamos
  -b : Tamaño de los trozos de datos en kbytes
  -P : PSM a utilizar
```

Para probarlo tenéis que antes compilarlo ejecutando el comando:

```
Linux:~ # cc –L/usr/lib –lbluetooth simple_l2test.c –o sl2
```

Por ejemplo en un ordenador podéis ejecutar:

```
Linux:~ # ./sl2 –r
```

y en otro ordenador:

```
Linux:~ #./sl2 –s direccion_maquina_anterior
```

Este programa se estructura básicamente en 4 procedimientos:

- **int do_connect(char *svr)**: que establece una conexión utilizando un L2CAP socket con la maquina con dirección MAC este contenida el 'svr' en el formato: xx:xx:xx:xx:xx:xx.
- **void do_listen(void (*handler)(int sk))**: que espera conexiones entrantes y que ejecuta la función 'handler' pasada como parámetro cuando establece una.
- **void recv_mode(int s)**: que se encarga de recibir datos utilizando el descriptor 's'
- **void send_mode(int s)**: que envía datos utilizando el descriptor 's'

Ejercicio 10:
Antes de todo tenéis que estudiar la estructura y el funcionamiento del código. El objetivo final es que lo modifiquéis para que sea posible utilizarlo como un sencillo programa para la transferencia de ficheros.
Al ejecutar vuestro programa con la opción '-s' el programa tendrá que conectarse a la maquina indicada en la línea de comandos, y una vez establecida la conexión tendrá que pedir el nombre de un fichero para su envío.
En la parte *receiver*, el programa habrá arrancado con la opción –r, y lo que se pretende es que todo lo que llega por el canal Bluetooth sea reenviado a stdout.

A continuación podéis encontrar el código del file simple_l2test.c original.

simple_l2test.c

```
/*
   BlueZ - Bluetooth protocol stack for Linux
   Copyright (C) 2000-2001 Qualcomm Incorporated

   Written 2000,2001 by Maxim Krasnyansky <maxk@qualcomm.com>

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License version 2 as
   published by the Free Software Foundation;

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS.
   IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) AND AUTHOR(S) BE LIABLE FOR ANY CLAIM,
   OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER
   RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
   NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE
   USE OR PERFORMANCE OF THIS SOFTWARE.

   ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY PATENTS, COPYRIGHTS,
   TRADEMARKS OR OTHER RIGHTS, RELATING TO USE OF THIS SOFTWARE IS DISCLAIMED.
*/

/*
 * $Id: l2test.c,v 1.3 2003/02/04 15:32:21 jscrane Exp $
 */

/*
 * simple_l2test.c 1.0 2003/OCT/1 pmanzoni
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/time.h>
#include <unistd.h>
#include <syslog.h>
#include <string.h>
#include <errno.h>
#include <signal.h>
#include <sys/select.h>

#include <netinet/in.h>
#include <arpa/inet.h>
#include <resolv.h>
#include <netdb.h>
#include <sys/socket.h>

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

/* Test modes */
enum {
   SEND,
   RECV
};

unsigned char *buf;

/* Default mtu */
int imtu = 672;
int omtu = 0;

/* Default data size */
long data_size = 672;

/* Default addr and psm */
bdaddr_t bdaddr;
unsigned short psm = 10;

int master = 0;
int auth = 0;
int encrypt = 0;
int socktype = SOCK_SEQPACKET;
```

```
float tv2fl(struct timeval tv)
{
   return (float)tv.tv_sec + (float)(tv.tv_usec/1000000.0);
}

int do_connect(char *svr)
{
   struct sockaddr_l2 rem_addr, loc_addr;
   struct l2cap_options opts;
   int s, opt;

   if( (s = socket(PF_BLUETOOTH, socktype, BTPROTO_L2CAP)) < 0 ) {
      syslog(LOG_ERR, "Can't create socket. %s(%d)", strerror(errno), errno);
      return -1;
   }

   memset(&loc_addr, 0, sizeof(loc_addr));
   loc_addr.l2_family = AF_BLUETOOTH;
   loc_addr.l2_bdaddr = bdaddr;
   if( bind(s, (struct sockaddr *) &loc_addr, sizeof(loc_addr)) < 0 ) {
      syslog(LOG_ERR, "Can't bind socket. %s(%d)", strerror(errno), errno);
      exit(1);
   }

   /* Get default options */
   opt = sizeof(opts);
   if( getsockopt(s, SOL_L2CAP, L2CAP_OPTIONS, &opts, &opt) < 0 ) {
      syslog(LOG_ERR, "Can't get default L2CAP options. %s(%d)", strerror(errno),
errno);
      return -1;
   }

   /* Set new options */
   opts.omtu = omtu;
   opts.imtu = imtu;
   if( setsockopt(s, SOL_L2CAP, L2CAP_OPTIONS, &opts, opt) < 0 ) {
      syslog(LOG_ERR, "Can't set L2CAP options. %s(%d)", strerror(errno), errno);
      return -1;
   }

   memset(&rem_addr, 0, sizeof(rem_addr));
   rem_addr.l2_family = AF_BLUETOOTH;
   baswap(&rem_addr.l2_bdaddr, strtoba(svr));
   rem_addr.l2_psm = htobs(psm);
   if( connect(s, (struct sockaddr *)&rem_addr, sizeof(rem_addr)) < 0 ){
      syslog(LOG_ERR, "Can't connect. %s(%d)", strerror(errno), errno);
      close(s);
      return -1;
   }

   opt = sizeof(opts);
   if( getsockopt(s, SOL_L2CAP, L2CAP_OPTIONS, &opts, &opt) < 0 ){
      syslog(LOG_ERR, "Can't get L2CAP options. %s(%d)", strerror(errno), errno);
      close(s);
      return -1;
   }

   syslog(LOG_INFO, "Connected [imtu %d, omtu %d, flush_to %d]\n",
          opts.imtu, opts.omtu, opts.flush_to);

   return s;
}

void do_listen( void (*handler)(int sk) )
{
   struct sockaddr_l2 loc_addr, rem_addr;
   struct l2cap_options opts;
   int  s, s1, opt;
   bdaddr_t ba;

   if( (s = socket(PF_BLUETOOTH, socktype, BTPROTO_L2CAP)) < 0 ) {
      syslog(LOG_ERR, "Can't create socket. %s(%d)", strerror(errno), errno);
      exit(1);
   }

   loc_addr.l2_family = AF_BLUETOOTH;
```

```
   loc_addr.l2_bdaddr = bdaddr;
   loc_addr.l2_psm    = htobs(psm);
   if( bind(s, (struct sockaddr *) &loc_addr, sizeof(loc_addr)) < 0 ) {
      syslog(LOG_ERR, "Can't bind socket. %s(%d)", strerror(errno), errno);
      exit(1);
   }

   /* Set link mode */
   opt = 0;
   if (master)
       opt |= L2CAP_LM_MASTER;

   if (auth)
       opt |= L2CAP_LM_AUTH;

   if (encrypt)
       opt |= L2CAP_LM_ENCRYPT;

   if (setsockopt(s, SOL_L2CAP, L2CAP_LM, &opt, sizeof(opt)) < 0) {
      syslog(LOG_ERR, "Can't set L2CAP link mode. %s(%d)", strerror(errno), errno);
      exit(1);
   }

   /* Get default options */
   opt = sizeof(opts);
   if (getsockopt(s, SOL_L2CAP, L2CAP_OPTIONS, &opts, &opt) < 0) {
      syslog(LOG_ERR, "Can't get default L2CAP options. %s(%d)", strerror(errno),
errno);
      exit(1);
   }

   /* Set new options */
   opts.imtu = imtu;
   if (setsockopt(s, SOL_L2CAP, L2CAP_OPTIONS, &opts, opt) < 0) {
      syslog(LOG_ERR, "Can't set L2CAP options. %s(%d)", strerror(errno), errno);
      exit(1);
   }

   if (socktype == SOCK_DGRAM) {
      handler(s);
      return;
   }

   if( listen(s, 10) ) {
      syslog(LOG_ERR,"Can not listen on the socket. %s(%d)", strerror(errno), errno);
      exit(1);
   }

   syslog(LOG_INFO,"Waiting for connection on psm %d ...", psm);

   while(1) {
      opt = sizeof(rem_addr);
      if( (s1 = accept(s, (struct sockaddr *)&rem_addr, &opt)) < 0 ) {
         syslog(LOG_ERR,"Accept failed. %s(%d)", strerror(errno), errno);
         exit(1);
      }
      if( fork() ) {
         /* Parent */
         close(s1);
         continue;
      }
      /* Child */

      close(s);

      opt = sizeof(opts);
      if( getsockopt(s1, SOL_L2CAP, L2CAP_OPTIONS, &opts, &opt) < 0 ) {
         syslog(LOG_ERR, "Can't get L2CAP options. %s(%d)", strerror(errno), errno);
         exit(1);
      }

      baswap(&ba, &rem_addr.l2_bdaddr);
      syslog(LOG_INFO, "Connect from %s [imtu %d, omtu %d, flush_to %d]\n",
             batostr(&ba), opts.imtu, opts.omtu, opts.flush_to);

      handler(s1);
```

```
      syslog(LOG_INFO, "Disconnect\n");
      exit(0);
   }
}


void recv_mode(int s)
{
   struct timeval tv_beg,tv_end,tv_diff;
   long total;
   uint32_t seq;

   syslog(LOG_INFO,"Receiving ...");

   seq = 0;
   while (1) {
      gettimeofday(&tv_beg,NULL);
      total = 0;
      while (total < data_size) {
         uint32_t sq;
         uint16_t l;
         int i,r;

         if ((r = recv(s, buf, data_size, 0)) <= 0) {
               if (r < 0)
                     syslog(LOG_ERR, "Read failed. %s(%d)",
                                    strerror(errno), errno);
               return;
         }

         /* Check sequence */
         sq = btohl(*(uint32_t *)buf);
         if (seq != sq) {
               syslog(LOG_INFO, "seq missmatch: %d -> %d", seq, sq);
               seq = sq;
         }
         seq++;

         /* Check length */
         l = btohs(*(uint16_t *)(buf+4));
         if (r != l) {
               syslog(LOG_INFO, "size missmatch: %d -> %d", r, l);
               continue;
         }

         /* Verify data */
         for (i=6; i < r; i++) {
               if (buf[i] != 0x7f)
                     syslog(LOG_INFO, "data missmatch: byte %d 0x%2.2x", i, buf[i]);
         }

         total += r;
      }
      gettimeofday(&tv_end,NULL);

      timersub(&tv_end,&tv_beg,&tv_diff);

      syslog(LOG_INFO,"%ld bytes in %.2f sec, %.2f kB/s",total,
             tv2fl(tv_diff), (float)(total / tv2fl(tv_diff) ) / 1024.0);
   }
}

void send_mode(int s)
{
   uint32_t seq;
   int i;

   syslog(LOG_INFO,"Sending ...");

   for(i=6; i < data_size; i++)
      buf[i]='X';             /* 0x7f; */

   seq = 0;
   while (1) {
      *(uint32_t *) buf = htobl(seq++);
      *(uint16_t *)(buf+4) = htobs(data_size);
```

```
      if (send(s, buf, data_size, 0) <= 0) {
         syslog(LOG_ERR, "Send failed. %s(%d)", strerror(errno), errno);
         exit(1);
      }
   }
}


void usage(void)
{
   printf("l2test - L2CAP testing\n"
      "Usage:\n");
   printf("\tl2test <mode> [options] [bdaddr]\n");
   printf("Modes:\n"
      "\t-r listen and receive\n"
      "\t-s connect and send\n");

   printf("Options:\n"
      "\t[-b bytes] [-P psm] [-I imtu] [-O omtu]\n");
}

extern int optind,opterr,optopt;
extern char *optarg;

int main(int argc ,char *argv[])
{
   int opt, mode, s, need_addr;
   struct sigaction sa;

   mode = RECV; need_addr = 0;

   while ((opt=getopt(argc,argv,"srb:P:I:O:")) != EOF) {
      switch(opt) {
      case 'r':
         mode = RECV;
         break;

      case 's':
         mode = SEND;
         need_addr = 1;
         break;

      case 'b':
         data_size = atoi(optarg);
         break;

      case 'P':
         psm = atoi(optarg);
         break;

      case 'I':
         imtu = atoi(optarg);
         break;

      case 'O':
         omtu = atoi(optarg);
         break;

      default:
         usage();
         exit(1);
      }
   }

   if (need_addr && !(argc - optind)) {
      usage();
      exit(1);
   }

   if (!(buf = malloc(data_size))) {
      perror("Can't allocate data buffer");
      exit(1);
   }

   memset(&sa, 0, sizeof(sa));
   sa.sa_handler = SIG_IGN;
   sa.sa_flags   = SA_NOCLDSTOP;
```

```
   sigaction(SIGCHLD, &sa, NULL);

   openlog("l2test", LOG_PERROR | LOG_PID, LOG_LOCAL0);

   switch( mode ){
      case RECV:
         do_listen(recv_mode);
         break;
      case SEND:
         s = do_connect(argv[optind]);
         if (s < 0)
               exit(1);
         send_mode(s);
         break;
   }
   syslog(LOG_INFO, "Exit");

   closelog();

   return 0;
}
```