Linux BlueZ Howto

Bluetooth protocol stack for Linux

Jan Beutel *j.beutel@ieee.org*, Maksim Krasnyanskiy *maxk@qualcomm.com*

14th November 2001

1 Introduction

BlueZ is the official Linux Bluetooth stack. It provides support for core Bluetooth layers and protocols.

Bluez has many interesting features:

- Flexible, efficient and modular architecture
- Support for multiple Bluetooth devices
- Multithreaded data processing
- Hardware abstraction
- Standard socket interface to all layers

Currently BlueZ consists of (see also figure 1):

- HCI Core
- HCI UART, USB and Virtual HCI device drivers
- L2CAP protocol module
- Configuration and testing utilities

2 Setting up BlueZ

2.1 Obtaining BlueZ

You can download the BlueZ source from *http://bluez.sourceforge.net*. There is also an up to date CVS tree available there.

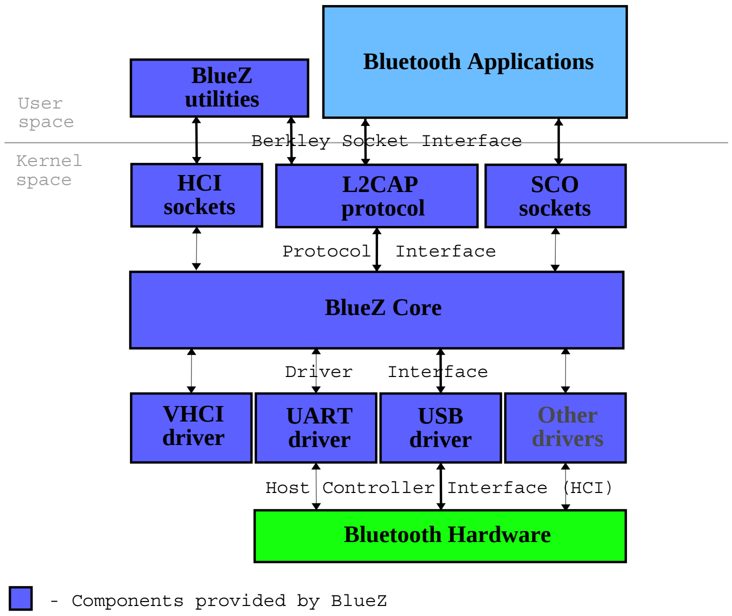

Figure 1: BlueZ Overview Diagram

2.2 Requirements

In order to use BlueZ, you need to have at least a 2.4.4 Linux kernel. The 2.4.6 kernel has BlueZ built-in. In case, if you want to use the latest version of BlueZ (see section 2.1 for download instructions), you should disable native BlueZ support.

BlueZ can be used with USB or Serial interface based Bluetooth devices. Additionally, Bluez provides Virtual HCI device (vhci) which can be used to test your Bluetooth applications. This is very useful if you do not have any real Bluetooth devices.

2.3 Compilation and Installation

To configure BlueZ run

```
./configure
```

to configure BlueZ for your kernel. The configure command automatically searches for all the required components and packages. Optionally, the configure support the following options:

`--enable-debug`	enable BlueZ debugging
`--with-kernel=<path>`	kernel source path (default is `/usr/src/linux`)

Once the Configure ran successfully, to compile and install run BlueZ, run:

```
make install
```

That's it!. Now, follow the next section to use BlueZ. See the README and configure.help for further compilation instructions including instructions for cross-compilation.

As usually it is good to check `/var/log/messages` for any output messages.

If you want to update your Linux kernel tree with the up to date CVS version run `make update` and recompile your kernel.

If you want the latest stuff don’t enable Bluetooth support in the kernel and use BlueZ - 1.2 or the latest CVS code instead. Be sure to have control of which modules you are loading.

2.4 Loading BlueZ Modules

The following lines need to be present in your `/etc/modules.conf` always in order for BlueZ to work correctly:

```
alias net-pf-31    bluez
alias bt-proto-0   l2cap
```

If you want to use UART based Bluetooth devices, add the following line to your `/etc/modules.conf` in addition to the above:

```
alias tty-ldisc-15 hci_uart
```

If you want to use the Virtual HCI device, add the following line to your `/etc/modules.conf`:

```
alias char-major-10-250 hci_vhci
```

After making any of the above changes, you can run ”depmod -a” to enable auto-loading of BlueZ modules.

Manual loading of the modules can be done by:

```
modprobe bluez
modprobe hci_uart   UART support. Optional
modprobe hci_usb    USB support. Optional
modprobe l2cap
```

You should see the BlueZ modules if you run `lsmod`. If there are any errors, check your `/var/log/messages` file.

2.5 Device Intitialization

UART Devices

Make sure that your `/etc/hcid.conf` is correct (tty, speed, flow, etc). See the example provided with the package (`daemons/hcid.conf`). Start `hcid`.

To configure the UART devices you need to use the tool `hciattach`. It can be called either manually or from the PCMCIA `cardmgr` scripts. The syntax is given in section 3.

```
#
# HCI daemon configuration file.
#
# $Id: bluezhowto.tex,v 1.5.1.2 2001/11/14 12:03:10 beutel Exp $
#

# HCId options
options {
        # Automaticaly initialize new devices
        autoinit yes;
}

# Default settings for HCI devices
default {
        # Local device name
        name BlueZ;

        # Local device class
        class 0x100;

        # Default packet type
        pkt_type DH1,DM1;
}

# HCI devices with UART interface configured without the use of hciattach
#uart {
#        /dev/ttyS0 57600 flow ericsson;
#        /dev/ttyS1 57600 flow ericsson;
#}
```

USB Device

Be sure to have USB support properly installed on your system. Plug in your USB device, check that the USB stack is loaded (usb-core and uhci or usb-uhci or ohci) and do:

```
modprobe hci_usb
```

Devices get initialized when they are plugged in (USB) or on the startup of the deamon (UART). When configured correctly they should be brought up automatically. Check your kernel and system logs for error messages.

You can manually bring device up by using the `hciconfig` command:

```
hciconfig hci0 up
```

2.6 Debugging the BlueZ Driver

If things go wrong don't panic but follow these guidelines.

Check

- the system log `/var/log/messages`
- the debug output from the BlueZ driver
- for dead processes, like `hcid`
- if you are loading the right modules compiled for your current kernel from the right location.

Also please try this:

`cvs update`	get the very latest CVS code
`make distclean`	clean any changes in the code
`./configure --enable-debug`	enable debug output in the BlueZ driver
`make update`	will make sure that Bluetooth headers in the kernel-tree are uptodate
`make`	
`make install`	install the newly comiled modules and tools

Does it still hang ? If it does:

- reboot
- unplug all Bluetooth USB devices (maybe even unplug all data and power connections for a while if you are using developer hardware)
- comment out all uart devices in `/etc/hcid.conf`
- kill hcid (if it was running)
- start emulator `hciemud localhost:10`
- start hcid

If you decide to call for help please include the following information in addition to you system logs:

- What bluetooth device are you using (Ericsson, Digi, etc.)?
- GCC version?
- What do you do and when exactly does it hang?
- `cat /etc/hcid.conf`

3 Tools

hciconfig - HCI device configuration utility

```
hciconfig hciX [up               Open and initialize HCI device
               |down             Close HCI device
               |reset            Reset HCI device
               |rstat]           Reset stat conters
               |auth             Enable Authentication
               |noauth           Disable Authentication
               |encrypt          Enable Encryption
               |noencrypt        Disable Encryption
               |piscan           Set page scan and inquiry scan mode
               |noscan           Disable scan modes
               |iscan            Set inquiry scan mode only
               |pscan            Set page scan mode only
               |inq [length]     Inquiry of devices
               |ptype [type]     Set packet type
               |lm [mode]        Get/Set default link mode
               |lp [policy]      Get/Set default link policy
               |conn             Show active connections
               |features         Show features
               |name [name]      Get/Set local name
               |class [class]    Get/Set class of device
               |version          Display version information
```

To query the current default packet type:

```
hciconfig hci0 ptype
```

To set the new packet type:

```
hciconfig hci0 ptype <types list separated by comma>
```

hciattach - HCI UART driver initialization utility

```
hciattach <tty> <type | id> [speed] [flow]
```

A simple utility that initializes a given serial port. It can be called either manually or from the PCMCIA `cardmgr` scripts. It is also working hot-plug for UART based PCMCIA devices. Using this tool you can add/delete UART devices without restarting `HCId`.

```
hciattach ttyS0 xircom 115200 flow
hciattach ttyS1 ericsson 115200 flow
hciattach ttyS2 any 57600
```

The PCMCIA `cardmgr` calls it like:

```
        hciattach ttyS1 0x0123,0x4567
```

l2ping - L2CAP ping

```
l2ping [-S source addr] [-s size] [-c count] [-f] <bd_addr>
```

l2test - L2CAP testing

```
l2test <mode> [-b bytes] [-P psm] [-I imtu] [-O omtu] [bd_addr]
```

Modes:	`-d`	Dump (server)
	`-c`	Reconnect (client)
	`-m`	Multiple connects (client)
	`-r`	Receive (server)
	`-s`	Send (client)

Options:	`-I`	Incoming MTU that we accept
	`-O`	Minimum outgoing MTU that we need
	`-b`	Size of the data chunks in kb
	`-P`	Use this PSM

If you have several devices on one box this may be useful:

```
   -S <Source BD address>
```

A simple throughput test using `l2test`:

Server:	`l2test -I 2000 -r`
Client:	`l2test -O 2000 -s <bd_addr>`

scotest - SCO testing

```
scotest <mode> [-b bytes] [bd_addr]
```

Modes:	`-d`	Dump (server)
	`-c`	Reconnect (client)
	`-m`	Multiple connects (client)
	`-r`	Receive (server)
	`-s`	Send (client)

3.1 Additional Tools

hcidump - HCI packet analyzer

```
hcidump <-i hciX> [-h]
```

hcitool - Generic writing and monitoring to the HCI interface

```
hcitool [-i hciX] OGF OCF param...
```

where `OGF` Is the OpCode Group Field (00-3F),
`OCF` is the OpCode Command Field (0000-03FF),
`param...` are parameters.

Each parameter is a sequence of bytes. Bytes are entered in hexadecimal form without spaces, most significant byte first. The size of each parameter is determined based on the number of bytes entered. An example to do an inquiry using LAP 0x9E8B33 for 10 × 1.28 sec and unlimited responses is:

```
hcitool -i hci0 01 0001 33 8b 9e 10 00
```

and to stop the inquiry:

```
hcitool -i hci0 01 0002
```

hciemud - HCI Emulation daemon

```
hciemud [-n] local_address
```

3.2 Tools Examples

Trace 1: DualPPro SMP machine. 2 Bluetooth devices (Ericsson AppTK) connected via UART.

```
bluetooth11:/> uname -a
Linux bluetooth11.qualcomm.com 2.4.4 #4 SMP Mon Apr 30 18:55:18 PDT 2001

bluetooth11:/> cat /etc/modules.conf
alias eth0 tlan
alias char-major-108    ppp_async
alias char-major-10-200 tun

# BlueZ
alias net-pf-31    bluez
alias bt-proto-0   l2cap
alias tty-ldisc-14 hci_uart

bluetooth11:/> cat /etc/hcid.conf
#
# HCI daemon configuration file.
#
# $Id: bluezhowto.tex,v 1.5.1.2 2001/11/14 12:03:10 beutel Exp $
#

# HCI devices with UART interface
uart {
        /dev/ttyS0 115200 flow ericsson;
        /dev/ttyS1 115200 flow ericsson;
        #/dev/ttyS0 57600 flow;
}

bluetooth11:/> lsmod
Module                  Size  Used by
tlan                   25056   1  (autoclean)

bluetooth11:/> hcid

bluetooth11:/> hciconfig
hci0:   Type: UART
         BD Address: 00:D0:B7:03:4B:F0 ACL MTU: 672:10  SCO: MTU 0:0
         UP RUNNING NORMAL PSCAN ISCAN
         RX bytes:62 acl:0 sco:0 events:7 errors:0
         TX bytes:36 acl:0 sco:0 commands:7 errors:0

hci1:   Type: UART
         BD Address: 00:D0:B7:03:4B:85 ACL MTU: 800:10  SCO: MTU 0:0
         UP RUNNING NORMAL PSCAN ISCAN
         RX bytes:62 acl:0 sco:0 events:7 errors:0
         TX bytes:36 acl:0 sco:0 commands:7 errors:0

bluetooth11:/> lsmod
Module                  Size  Used by
hci_uart                4656   2  (autoclean)
bluez                   22544   3  (autoclean) [hci_uart]

bluetooth11:/> l2ping 00:D0:B7:03:4B:85
Ping: 00:D0:B7:03:4B:85 from 00:D0:B7:03:4B:F0 (data size 20) ...
20 bytes from 00:D0:B7:03:4B:85 id 200 time 48.91ms
20 bytes from 00:D0:B7:03:4B:85 id 201 time 50.02ms
2 sent, 2 received, 0% loss

bluetooth11:/usr/src/bluez/tools> ./l2test -r -b 10 -I 2000 &
[1] 22761
l2test[22761]: Waiting for connection on psm 10 ...

bluetooth11:/usr/src/bluez/tools> ./l2test -s 00:D0:B7:03:4B:85
l2test[22763]: Connected [imtu 672, omtu 2000, flush_to 65535]
l2test[22763]: Sending ...
l2test[22764]: Connect from 00:D0:B7:03:4B:F0 [imtu 2000, omtu 672, flush_to 65535]
l2test[22764]: Receiving ...
l2test[22764]: 10240 bytes in 0.01m speed 11.12 kb
l2test[22764]: 10240 bytes in 0.01m speed 11.12 kb
l2test[22764]: 10240 bytes in 0.02m speed 11.00 kb
l2test[22764]: 10240 bytes in 0.01m speed 11.12 kb
l2test[22764]: 10240 bytes in 0.01m speed 11.12 kb
l2test[22764]: 10240 bytes in 0.01m speed 11.12 kb
<ctrl-c>
l2test[22764]: Read failed. Connection reset by peer(104)
l2test[22764]: Disconnect

bluetooth11:/> lsmod
Module                  Size  Used by
l2cap                  17520   0  (autoclean)
```

```
hci_uart                4656   2  (autoclean)
bluez                  22544   3  (autoclean) [l2cap hci_uart]
tlan                   25056   1  (autoclean)
```

Trace 2: Notebook (Compaq PIII). 1 Bluetooth device (Ericsson AppTK) connected via USB.

```
btdemo1:~>lsmod
Module                  Size  Used by
uhci                   23040   0  (unused)
eepro100               15984   1  (autoclean)
usbcore                48784   1  [uhci]

btdemo1:~>modprobe l2cap
btdemo1:~>lsmod
Module                  Size  Used by
l2cap                  15552   0  (unused)
bluez                  20624   0  [l2cap]
uhci                   23040   0  (unused)
eepro100               15984   1  (autoclean)
usbcore                48784   1  [uhci]

btdemo1:~>modprobe hci_usb

btdemo1:~>hciconfig
hci0:   Type: USB
         BD Address: 00:00:00:00:00:00 ACL MTU: 0:0  SCO: MTU 0:0
         DOWN NORMAL
         RX bytes:0 acl:0 sco:0 events:0 errors:0
         TX bytes:0 acl:0 sco:0 commands:0 errors:0

btdemo1:~>hciconfig hci0 up
btdemo1:~>hciconfig
hci0:   Type: USB
         BD Address: 00:D0:B7:03:4B:3B ACL MTU: 672:10  SCO: MTU 255:255
         UP RUNNING NORMAL PSCAN ISCAN
         RX bytes:61 acl:0 sco:0 events:8 errors:0
         TX bytes:33 acl:0 sco:0 commands:8 errors:0

btdemo1:~>l2ping aa:aa:aa:aa:aa:aa
Can't connect.: Host is down
```

Trace3: Setting packet types on Ericsson AppTK

```
bluez:/usr/src>hciconfig hci0
hci0:   Type: USB
         BD Address: 00:D0:B7:03:4B:3D ACL MTU: 672:10  SCO: MTU 255:255
         UP RUNNING NORMAL PSCAN ISCAN
         RX bytes:55 acl:0 sco:0 events:7 errors:0
         TX bytes:29 acl:0 sco:0 commands:7 errors:0

bluez:/usr/src>hciconfig hci0 ptype
hci0:   Type: USB
         Default packet type: DM1 DM3 DM5 DH1 DH3 DH5
```

```
bluez:/usr/src>hciconfig hci0 ptype DH1,DH5
bluez:/usr/src>hciconfig hci0 ptype
hci0:   Type: USB
         Default packet type: DH1 DH5
```

3.3 Performance

This will give you a rough guideline at what you can expect to see on a Point to Point connection.

USB Operation Datarates as reported by `l2test` are given in kiloBytes per second.

Packet Type	Datarate
DH1	20 kBps
DH3	50 kBps
DH5	65-80 kBps

UART Operation The UART HCI Transport Layer will be the limiting factor to any connection.

Baudrate	Packet Type	Datarate
57600	any	5 kBps
112500	any	5 kBps

Modifications to the MTU or packet sizes did not make any significant difference.

4 Setting up RFCOMMd and PPP on Top of BlueZ

You will need a kernel with ppp support enabled. To configure and install RFCOMMd run:

```
./configure
make install
```

Server: `rfcommd <-s> [-f file] [-P port]`
Client: `rfcommd [-f file] [-P port] [-L local address] [-p]`
`[-t timeout] <host> <server adress>`

Using the `-n` option RFCOMMd won't detach from the terminal and you should see information and error messages there.

4.1 Setting up a PPP Link

Here is an example of the `rfcommd.conf` files for client and server.

Server side:

```
options {
   psm 3;        # Listen on this psm.
   ppp            /usr/sbin/pppd;
   ifconfig       /sbin/ifconfig;
   route          /sbin/route;
   firewall       /sbin/ipchains;
}

# Network Access
na {
   channel 1;
   up {
         ppp "noauth 10.0.0.1:10.0.0.2";
   };
}
```

Start the server using ip 10.0.0.1 with:

```
 rfcommd -s na
```

Client side:

```
options {
   psm 3;         # Listen on this psm.

   ppp            /usr/sbin/pppd;
   ifconfig       /sbin/ifconfig;
   route          /sbin/route;
   firewall       /sbin/ipchains;
}

# Network Access
na {
   channel 1;
   up {
         ppp "noauth";
   };
}
```

Start the client using ip 10.0.0.2 with:

```
 rfcommd na server_bd_addr
```

This will give you a ppp link between 10.0.0.1 (ppp server) and 10.0.0.2 (ppp client). Try ping.

To debug you may want to add the options `debug` and `record /tmp/pppd.log`. To see the PPP exchange, run `pppdump /tmp/pppd.log`.

4.2 Setting up an Internet Gateway

Now you will want to be able to route packets to and from your regular internet connection. This will make your linux box into a Bluetooth access point. We assume this is available on `eth0` on the ppp server. You will need a kernel configured with Netfilters for this operation.

On the server side check your routes and iptables:

```
[beutel@tec-pc-jg]# ifconfig
```

```
eth0      Link encap:Ethernet  HWaddr 00:50:DA:46:E3:20
          inet addr:129.132.119.47  Bcast:129.132.119.63  Mask:255.255.255.192
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:20370 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10303 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:10 Base address:0xb800

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0

ppp0      Link encap:Point-to-Point Protocol
          inet addr:10.0.0.1  P-t-P:10.0.0.2  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3

[beutel@tec-pc-jg]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.2        0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
129.132.119.0   *               255.255.255.192 U     0      0        0 eth0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         rou-etx-1-tik-t 0.0.0.0         UG    0      0        0 eth0

[beutel@tec-pc-jg]# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Now we will set up Network Address Translation (NAT) for all packets leaving `eth0`:

```
 iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

and enable ip forwarding:

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

now your NAT table will have the following entry:

```
[beutel@tec-pc-jg]# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  anywhere             anywhere
```

```
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

On the client side you will need to define your default route as follows:

```
 route add default gw 10.0.0.1
```

Delete all other routes that might still be present on the client.

```
[beutel@tec-pc-jb]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.1        *               255.255.255.255 UH    0      0        0 ppp0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         10.0.0.1        0.0.0.0         UG    0      0        0 ppp0
```

5 Piconet Role Switch

For using the Role Switch function you will need hardware that supports this feature. Be sure to have the right hardware before continuing in this section.

Devices have a default `link_mode` setting (In BlueZ this is called `link_mode` because it is somewhat different from `link_policy`). The default `LM` can be changed via `hciconfig hciX lm <flags>`

Valid LM flags are:

`ACCEPT` Accept an incoming connection event if upper layers (L2CAP, SCO) didn’t accept it. If this flag is not set and L2CAP/SCO don’t have listening sockets we reject connection.

`MASTER` Always be the `MASTER` i.e. do the role switch on incoming connections and don’t accept Role Switch on outgoing connections. If this flag is not set we accept Role Switch on outgoing connections and don’t do the Role Switch on incoming connections.

`NONE` Just clears all flags.

Examples of Role Switch:

```
champ:/tmp#hciconfig hci0 lm accept,master
champ:/tmp#hciconfig hci0 lm
hci0:   Type: USB
        BD Address: 00:80:37:14:42:45 ACL MTU: 672:10  SCO: MTU 255:255
        Default link mode: ACCEPT MASTER

champ:/tmp#hciconfig hci0 lm none
champ:/tmp#hciconfig hci0 lm
hci0:   Type: USB
        BD Address: 00:80:37:14:42:45 ACL MTU: 672:10  SCO: MTU 255:255
        Default link mode:
```

You can also set the default `LM` in hcid.conf using the same syntax.

Also `LM` can be set per application (currently on servers only) Applications can set `LM_MASTER` flags on the listening socket. On incoming connections we check if listeners need a Role Switch and if they do we request it. `l2test` also has an option `-M` which sets the `LM_MASTER` flag.

5.1 Point to Multipoint Connections

In order to set up a point-to-multipoint connection you will need to use the following syntax in the `rfcommd.conf` file:

```
session_X {
        channel X;
}

session_Y {
        channel Y;
}
```

and then start a server for each session:

```
rfcommd session_X server
rfcommd session_Y server
```

6 Service Discovery Protocol

In general I think it should look like DNS Unix implementation e.g: SDPd - SDP server (like BIND) SDPclient - SDP client library (like Resolver library)

Things we need: - SDPlib - Generic library for building/parsing SDP packets. This be used internally by the server daemon and client library to construct/parse SDP messages.

- SDPd - SDP server Daemon that listens on L2CAP and Unix sockets. This guy should respond on incoming SDP queries, maintain SDP database, allow record registration via unix socket, etc Also it needs nice config file where you can specify predefined services (if you think rfcommd config is ok you can easily reuse that code). Daemon will use SDPlib.

- SDPclient - SDP client library This thing should provide nice high level client API. It will use L2CAP socket and SDPlib.

I was thinking about creating ’sdp’ module in BlueZ CVS.

I guess the SDP server side will be implemented as a TCP/IP server, i.e. accepting connections on the SDP port of the L2CAP layer (instead of a TCP port over IP). The SDP will run in user mode. Right ?

Yes. I’d recommend to read early archives of bluez-users mailing list. We discussed SDP design with Gordon there. In short SDP implementation is somewhat similar to DNS implementation e.g. SDP server - named, SDP lib - resolver.

7 Programming Using BlueZ

BlueZ provides standard socket interface to all its layers (hci,l2cap..). Have a look at bluez-1.2/tools/l2test.c bluez-1.2/tools/l2ping.c.

Adding Bluez support to any existing socket based programs is very easy. For eg. You would use `AF_BLUETOOTH` instead of `AF_INET`(ip) when you make ”socket” call. You would use `sockaddr_l2` instead of `sockaddr_in`. `SOCK_SEQ_PACKET` instead of `SOCK_STREAM` and so on. Only few new data structures and constants. setsockopt and getsockopt take new data structure and constants.

The BlueZ protocol stack is interfacing to the Linux socket layer, providing a new address family. Through the raw HCI socket interface and hcilib one has a function `sendcmd`. An application can thus send any HCI command to a device and receive events in return.

There is no point of implementing ioctls for every HCI command. So, we'll have only important set of ioctls like `DEVUP`, `DOWN`, etc. Everything else should be done via the raw sockets.

Operations like initializing a `local_name` and `class_of_device` is supposed to go into hcid.

7.1 HCI Packet Filters

Filter allows you to specify packet types and events your HCI application is interested in. It's also an optimization feature in HCI core that makes sure that we don't send junk to the apps and therefore don't waste memory and don't wake up processes unnecessary. For example in case of a simple HCI app that sends one command and expects one event back. Without socket filter we would send a copy of _every_ single packet that is sent/received on this device.

Filter API:

```
struct hci_filter {
       __u32 type_mask;       // Packet type mask
       __u32 event_mask[2];  // Event mask
} flt;

/* Set filter */
flt.type_mask  = ((1<<HCI_ACLDATA_PKT) | (1<<HCI_EVENT_PKT));
                             // Event and ACL data packets
flt.event_mask[0] = ~0L;      // All events
flt.event_mask[1] = ~0L;
setsockopt(s, SOL_HCI, HCI_FILTER, &flt, sizeof(flt));

/* Get filter */
len = sizeof(flt);
getsockopt(s, SOL_HCI, HCI_FILTER, &flt, &len);
```

Default filter is set to ”event packets only / all events”, which is what most HCI apps want. So, if you need SCO data or something else don't forget to set a new filter

8 Setting up VTun on top of BlueZ

VTun is the easiest way to create Virtual Tunnels over TCP/IP networks with traffic shaping, compression, and encryption. It supports IP, Ethernet, PPP and other tunnel types. VTun is easily and highly configurable. It can be used for various networks tasks:

- VPN
- Mobile IP
- Shaping
- and others

Starting from version 2.4, VTun supports Virtual Tunnels over L2CAP. This can be used to run almost all IP-based programs such as Apache web server.

8.1 Compilation and Installation

To setup VTun over L2CAP, obtain a copy of VTun from the BlueZ source *http://bluez.sourceforge.net*. You can also obtain it from *http://vtun.sourceforge.net*. The compilation instructions can be found in the `vtun/README` file.

Before installing VTun, make sure that you have enabled the following kernel option:

```
CONFIG_TUN
```

If you don't have this option enabled, you need to either re-compiler your kernel with this option as a module or alternatively, you can obtain Universal TUN/TAP from *h*ttp://vtun.sourceforge.net/tun/index.html.

Add the folllowing line to your `/etc/modules.conf`

```
#VTun
alias char-major-10-200 tun
```

Create a tun device as follows:

`mknod /dev/tun c 10 200`	if you are using 2.4.2 or below
`mknod /dev/net/tun c 10 200`	if you are using kernel version 2.4.4 or greater

Now, load the tun module as bellow:

```
modprobe tun
```

Make sure, you see the tun module when you run lsmod command.

8.2 Configuration

VTun can be configured for a complex network setup. This section will show how to setup a simple host and client network over L2CAP. For various other configuration options, read the `Readme.Setup` file from VTun.

Date	Section	Changes
Aug. 2, 2001	1,2,3	Initial Version
Aug. 7, 2001	5,6	Added filters, checked tool section
Aug. 9, 2001	3,4,5	Added rfcommd and ppp, added packet type, started programming section, l2test new, trying to get sdp setup going
Aug. 21, 2001	2	Changed module loading and compiling instructions

Table 1: Revision History