Bluetooth Device Driver In Linux

Bluetooth is a short-range cable-replacement technology that carries both data and voice. It supports speeds of up to 723Kbps (asymmetric) and 432Kbps (symmetric). Class 3 Bluetooth devices have a range of 10 meters, and Class 1 transmitters can communicate up to 100 meters.

Bluetooth is designed to do away with wires that constrict and clutter your environment. It can, for example, turn your wristwatch into a front-end for a bulky Global Positioning System (GPS) hidden inside your backpack. Or it can, for instance, let you navigate a presentation via your handheld. Again, Bluetooth can be the answer if you want your laptop to be a hub that can Internet-enable your Bluetooth-aware MP3 player. If your wristwatch, handheld, laptop, or MP3 player is running Linux, knowledge of the innards of the Linux Bluetooth stack will help you extract maximum mileage out of your device.

As per the Bluetooth specification, the protocol stack consists of the layers shown in Figure 16.1. The radio, link controller, and link manager roughly correspond to the physical, data link, and network layers in the OpenSystems Interconnect (OSI) standard reference model. The Host Control Interface (HCI) is the protocol that carries data to/from the hardware and, hence, maps to the transport layer. The Bluetooth Logical Link Control and Adaptation Protocol (L2CAP) falls in the session layer. Serial port emulation using Radio Frequency Communication (RFCOMM), Ethernet emulation using Bluetooth Network Encapsulation Protocol (BNEP), and the Service Discovery Protocol (SDP) are part of the feature-rich presentation layer. At the top of the stack reside various application environments called profiles. The radio, link controller, and link manager are usually part of Bluetooth hardware, so operating system support starts at the HCI layer.

A common method of interfacing Bluetooth hardware with a microcontroller is by connecting the chipset’s data lines to the controller’s UART pins. Figure 13.4 of Chapter 13, “Audio Drivers,” shows a Bluetooth chip on an MP3 player communicating with the processor via a UART. USB is
another oft-used vehicle for communicating with Bluetooth chipsets. Figure 11.2 of Chapter 11, “Universal Serial Bus,” shows a Bluetooth chip on an embedded device interfacing with the processor over USB. Irrespective of whether you use UART or USB (we will look at both kinds of devices later), the packet format used to transport Bluetooth data is HCI.

**BlueZ**

The BlueZ Bluetooth implementation is part of the mainline kernel and is the official Linux Bluetooth stack. Figure 16.2 shows how BlueZ maps Bluetooth protocol layers to kernel modules, kernel threads, user-space daemons, configuration tools, utilities, and libraries. The main BlueZ components are explained here:

1. bluetooth.ko contains the core BlueZ infrastructure. All other BlueZ modules utilize its services. It’s also responsible for exporting the Bluetooth family of sockets (`AF_BLUETOOTH`) to user space and for populating related sysfs entries.

2. For transporting Bluetooth HCI packets over UART, the corresponding BlueZ HCI implementation is hci_uart.ko. For USB transport, it’s hci_usb.ko.

3. l2cap.ko implements the L2CAP adaptation layer that is responsible for segmentation and reassembly. It also multiplexes between different higher-layer protocols.

4. To run TCP/IP applications over Bluetooth, you have to emulate Ethernet ports over L2CAP using BNEP. This is accomplished by bnep.ko. To service BNEP connections, BlueZ spawns a kernel thread called kbnepd.

5. To run serial port applications such as terminal emulators over Bluetooth, you need to emulate serial ports over L2CAP. This is accomplished by rfcomm.ko. RFCOMM also functions as the pillar that supports networking over PPP. To service incoming RFCOMM connections, rfcomm.ko spawns a kernel thread called krfcommd. To set up and maintain connections to individual RFCOMM channels on target devices, use the rfcomm utility.

6. The HID layer is implemented via hidp.ko. The user mode daemon hidd lets BlueZ handle input devices such as Bluetooth mice.

7. Audio is handled via the Synchronous Connection Oriented (SCO) layer implemented by sco.ko.
Let’s now trace the kernel code flow for two example Bluetooth devices: a Compact Flash (CF) card and a USB adapter.

**Device Example: CF Card**
The Sharp Bluetooth Compact Flash card is built using a Silicon Wave chipset and uses a serial transport to carry HCI packets. There are three different ways by which HCI packets can be transported serially:

1. H4 (UART), which is used by the Sharp CF card. H4 is the standard method to transfer Bluetooth data over UARTs as defined by the Bluetooth specification. Look at drivers/bluetooth/hci_h4.c for the BlueZ implementation.

2. H3 (RS232). Devices using H3 are hard to find. BlueZ has no support for H3.

3. BlueCore Serial Protocol (BCSP), which is a proprietary protocol from Cambridge Silicon Radio (CSR) that supports error checking and retransmission. BCSP is used on non-USB devices based on CSR BlueCore chips including PCMCIA and CF cards. The BlueZ BCSP implementation lives in drivers/bluetooth/hci_bcsp.c.

The read data path for the Sharp Bluetooth card is shown in Figure 16.3. The first point of contact between the card and the kernel is at the UART driver. As you saw in Figure 9.5 of Chapter 9, “PCMCIA and Compact Flash,” the serial Card Services driver drivers/serial/serial_cs.c, allows the rest of the operating system to see the Sharp card as if it were a serial device. The serial driver passes on the received HCI packets to BlueZ. BlueZ implements HCI processing in the form of a kernel line discipline. As you learned in Chapter 6, “Serial Drivers,” line disciplines reside above the serial driver and shape its behavior. The HCI line discipline invokes associated protocol routines (H4 in this case) for assistance in data processing. From then on, L2CAP and higher BlueZ layers take charge.

Figure 16.3. Read data path from a Sharp Bluetooth CF card.

[View full size image]
Device Example: USB Adapter

Let’s now look at a device that uses USB to transport HCI packets. The Belkin Bluetooth USB adapter is one such gadget. In this case, the Linux USB layer (drivers/usb/*), the HCI USB transport driver (drivers/bluetooth/hci_usb.c), and the BlueZ protocol stack (net/bluetooth/*) are the main players that get the data rolling. Let’s see how these three kernel layers interact.

As you learned in Chapter 11, USB devices exchange data using one or more of four pipes. For Bluetooth USB devices, each pipe is responsible for carrying a particular type of data:

1. Control pipes are used to transport HCI commands.
2. Interrupt pipes are responsible for carrying HCI events.
3. Bulk pipes transfer asynchronous connectionless (ACL) Bluetooth data.
4. Isochronous pipes carry SCO audio data.

You also saw in Chapter 11 that when a USB device is plugged into a system, the host controller driver enumerates it using a control pipe and assigns endpoint addresses between 1 and 127. The configuration descriptor read by the USB subsystem during enumeration contains information about the device, such as its class, subclass, and protocol. The Bluetooth specification defines the (class, subclass, protocol) codes of Bluetooth USB devices as (0xE, 0x01, 0x01). The HCI USB transport driver (hci_usb) registers these values with the USB core during initialization. When the Belkin USB adapter is plugged in, the USB core reads the (class, subclass, protocol) information from the device configuration descriptor. Because this information matches the values registered by hci_usb, this driver gets attached to the Belkin USB adapter. hci_usb reads Bluetooth data from the four USB pipes described previously and passes it on to the BlueZ protocol stack. Linux applications now run seamlessly over this device, as shown in Figure 16.2.

**RFCOMM**

RFCOMM emulates serial ports over Bluetooth. Applications such as terminal emulators and protocols such as PPP can run unchanged over the virtual serial interfaces created by RFCOMM.

**Device Example: Pill Dispenser**

To take an example, assume that you have a Bluetooth-aware pill dispenser. When you pop a pill out of the dispenser, it sends a message over a Bluetooth RFCOMM channel. A Linux cell phone, such as the one shown in Figure 6.5 of Chapter 6, reads this alert using a simple application that establishes an RFCOMM connection to the pill dispenser. The phone then dispatches this information to the health-care provider’s server on the Internet via its GPRS interface.

A skeletal application on the Linux cell phone that reads data arriving from the pill dispenser using the BlueZ socket API is shown in Listing 16.1. The listing assumes that you are familiar with the basics of socket programming.

**Listing 16.1. Communicating with a Pill Dispenser over RFCOMM**

```c
#include <sys/socket.h>
#include <bluetooth/rfcomm.h>  /* For struct sockaddr_rc */

void
sense_dispenser()
{
    int pillfd;
    struct sockaddr_rc pill_rfcomm;
    char buffer[1024];

    /* ... */

    /* Create a Bluetooth RFCOMM socket */
    if ((pillfd = socket(PF_BLUETOOTH, SOCK_STREAM, BPROTO_RFCOMM)) < 0) {
```
Networking

Trace down the code path from the telnet/ftp/ssh box in Figure 16.2 to see how networking is architected over BlueZ Bluetooth. As you can see, there are two different ways to network over Bluetooth:

1. By running TCP/IP directly over BNEP. The resulting network is called a personal area network (PAN).
2. By running TCP/IP over PPP over RFCOMM. This is called dialup networking (DUN).

The kernel implementation of Bluetooth networking is unlikely to interest the device driver writer and is not explored. Table 16.2 shows the steps required to network two laptops using PAN, however. Networking with DUN resembles this and is not examined. The laptops are respectively Bluetooth-enabled using the Sharp CF card and the Belkin USB adapter discussed earlier. You can slip the CF card into the first laptop’s PCMCIA slot using a passive CF-to-PCMCIA adapter. Look at Figure 16.2 in tandem with Table 16.2 to understand the mappings to corresponding BlueZ components. Table 16.2 uses bash-sharp> and bash-belkin> as the respective shell prompts of the two laptops.
Table 16.2. Networking Two Laptops Using Bluetooth PAN

<table>
<thead>
<tr>
<th>On the laptop with the Sharp Bluetooth CF card</th>
<th>1. Start the HCI and service discovery daemons:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. <code>bash-sharp&gt; hcid</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Because this device possesses a UART interface, you have to attach the BlueZ stack to the appropriate serial port. In this case, assume that serial_cs has allotted <code>/dev/ttyS3</code> to the card:</td>
</tr>
<tr>
<td></td>
<td><code>bash-sharp&gt; hciattach ttyS3 any</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Verify that the HCI interface is up:</td>
<td>4. <code>bash-sharp&gt; hciconfig -a</code></td>
</tr>
<tr>
<td></td>
<td>5. <code>hci0: Type: UART</code></td>
</tr>
<tr>
<td></td>
<td>6. <code>BD Address: 08:00:1F:10:3B:13 ACL MTU: 60:20 SCO MTU: 31:1</code></td>
</tr>
<tr>
<td></td>
<td>7. <code>UP RUNNING PSCAN ISCAN</code></td>
</tr>
<tr>
<td></td>
<td>8. <code>...</code></td>
</tr>
<tr>
<td></td>
<td><code>Manufacturer: Silicon Wave (11)</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Verify that basic BlueZ modules are loaded:</td>
<td>10. <code>bash-sharp&gt; lsmod</code></td>
</tr>
<tr>
<td></td>
<td>11. <code>Module                      Size  Used by</code></td>
</tr>
<tr>
<td></td>
<td>12. <code>hci_uart 16728  3</code></td>
</tr>
<tr>
<td></td>
<td>13. <code>l2cap 26144  2</code></td>
</tr>
<tr>
<td></td>
<td>14. <code>bluetooth 47684  6 hci_uart,l2cap</code></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
</tr>
<tr>
<td>15. Insert the BlueZ module that implements network encapsulation:</td>
<td><code>bash-sharp&gt; modprobe bnep</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>16. Listen for incoming PAN connections:</td>
<td><code>bash-sharp&gt; pand -s</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>On the laptop with the Belkin USB Bluetooth adapter</th>
<th>1. Start daemons, such as hcid and sdpd, and insert necessary kernel modules, such as bluetooth.ko and l2cap.ko.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Because this is a USB device, you don’t need to invoke hciattach, but make sure that the hci_usb.ko module is inserted.</td>
</tr>
<tr>
<td></td>
<td>3. Verify that the HCI interface is up:</td>
</tr>
<tr>
<td></td>
<td>4. <code>bash-belkin&gt; hciconfig -a</code></td>
</tr>
<tr>
<td></td>
<td>5. <code>hci0: Type: USB</code></td>
</tr>
<tr>
<td></td>
<td>6. <code>BD Address: 00:02:72:B0:33:AB ACL MTU: 192:8 SCO MTU: 64:8</code></td>
</tr>
<tr>
<td></td>
<td>7. <code>UP RUNNING PSCAN ISCAN</code></td>
</tr>
<tr>
<td></td>
<td>8. <code>...</code></td>
</tr>
<tr>
<td></td>
<td><code>Manufacturer: Cambridge Silicon Radio (10)</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Search and discover devices in the neighborhood:</td>
<td>10. <code>bash-belkin&gt; hcitool -i hci0 scan --flush</code></td>
</tr>
<tr>
<td></td>
<td>11. <code>Scanning.... 08:00:1F:10:3B:13 bash-sharp</code></td>
</tr>
</tbody>
</table>
Table 16.2. Networking Two Laptops Using Bluetooth PAN

On the laptop with the Sharp Bluetooth CF card

1. Start the HCI and service discovery daemons:

   bash-sharp> hcid

Because this device possesses a UART interface, you have to attach the BlueZ stack to the appropriate serial port. In this case, assume that serial_cs has allotted /dev/ttyS3 to the card:

   bash-sharp> hciattach ttyS3 any

3. Verify that the HCI interface is up:

   4. bash-sharp> hciconfig -a
   5. hci0:   Type: UART
   6. BD Address: 08:00:1F:10:3B:13 ACL MTU: 60:20  SCO MTU: 31:1
   7. UP RUNNING PSCAN ISCAN
   8. ...
          Manufacturer: Silicon Wave (11)

9. Verify that basic BlueZ modules are loaded:

   10. bash-sharp> lsmod

   11. Module                  Size  Used by
       12. hci_uart               16728  3
       13. l2cap                  26144  2
       14. bluetooth              47684  6 hci_uart,l2cap
           ...

15. Insert the BlueZ module that implements network encapsulation:

   bash-sharp> modprobe bnep

16. Listen for incoming PAN connections:[1]

   bash-sharp> pand -s

12. Establish a PAN with the first laptop. You can get its Bluetooth address (08:00:1F:10:3B:13) from its hciconfig output:

   bash-belkin> pand -c 08:00:1F:10:3B:13

If you now look at the ifconfig output on the two laptops, you will find that a new interface named bnep0 has made an appearance at both ends. Assign IP addresses to both interfaces and get ready to telnet and FTP!

[1] A useful command-line option to pand is --persist, which automatically attempts to reconnect when a connection drops. Dig into the man pages for more invocation options.

Human Interface Devices
Look at sections “USB and Bluetooth Keyboards” and “USB and Bluetooth Mice” in Chapter 7, “Input Drivers,” for a discussion on Bluetooth human interface devices.

Audio

Let’s take the example of an HBH-30 Sony Ericsson Bluetooth headset to understand Bluetooth SCO audio. Before the headset can start communicating with a Linux device, the Bluetooth link layer on the latter has to discover the former. For this, put the headset in discover mode by pressing the button earmarked for device discovery. In addition, you have to configure BlueZ with the headset’s personal identification number (PIN) by adding it to /etc.bluetooth/pin. An application on the Linux device that uses BlueZ SCO APIs can now send audio data to the headset. The audio data should be in a format that the headset understands. The HBH-30 uses the A-law PCM (pulse code modulation) format. There are public domain utilities for converting audio into various PCM formats.

Bluetooth chipsets commonly have PCM interface pins in addition to the HCI transport interface. If a device supports, for instance, both Bluetooth and GSM, the PCM lines from the GSM chipset may be directly wired to the Bluetooth chip’s PCM audio lines. You might then have to configure the Bluetooth chip to receive and send SCO audio packets over its HCI interface instead of its PCM interface.

Debugging

There are two BlueZ tools useful for debugging:

1. hcidump taps HCI packets flowing back and forth, and parses them into human-readable form. Here’s an example dump while a device inquiry is in progress:
   1. bash> hcidump -i hci0
   2. HCIDump - HCI packet analyzer ver 1.11
   3. device: hci0 snap_len: 1028 filter: 0xffffffff
   4. HCI Command: Inquiry (0x01|0x0001) plen 5
   5. HCI Event: Command Status (0x0f) plen 4
   6. HCI Event: Inquiry Result (0x02) plen 15
   7. ...
   8. ...
   9. HCI Event: Inquiry Complete (0x01) plen 1 < HCI Command:
   10. Remote Name Request (0x01|0x0019) plen 10
   11. ...

11. The virtual HCI driver (hci_vhci.ko), as shown in Figure 16.2, emulates a Bluetooth interface if you do not have actual hardware.

Looking at the Sources

Look inside drivers/bluetooth/ for BlueZ low-level drivers. Explore net/bluetooth/ for insights into the BlueZ protocol implementation.

Bluetooth applications fall under different profiles based on how they behave. For example, the cordless telephony profile specifies how a Bluetooth device can implement a cordless phone. We discussed profiles for PAN and serial access, but there are many more profiles out there such as fax profile, General Object Exchange Profile (GOEP) and SIM Access Profile (SAP). The bluez-utils package, downloadable from www.bluez.org, provides support for several Bluetooth profiles.
The official Bluetooth website is [www.bluetooth.org](http://www.bluetooth.org). It contains Bluetooth specification documents and information about the Bluetooth Special Interest Group (SIG).