

DIRECT DATA TRANSFER OVER THE HOST CONTROLLER INTERFACE OF THE BLUEZ BLUETOOTH® PROTOCOL STACK IN BAYANIHAN LINUX V2.0

William R. Cheung^{*}, Joel T. Fallorina^{*} and Janice M. Ballesteros
Advanced Science and Technology Institute
UP Technopark, Diliman, Quezon City
Department of Science and Technology
mobilesys@asti.dost.gov.ph

ABSTRACT

The Bluetooth specification defines a whole protocol stack necessary for a standardized interface of different Bluetooth devices. One of the layers in the stack is the Host Controller Interface (HCI). It serves a vital role in interfacing the higher and the lower parts of the stack. This paper discusses the implementation, setup, and testing processes involved in directly utilizing the HCI layer in sending and receiving data. An application was developed on top of Bayanihan Linux v2.0 which has built-in BlueZ, the official Linux Bluetooth protocol stack. The application shows the operations and functions of the HCI layers in transferring data. Accessing the HCI layer directly will have several advantages, one of which is the reduced overhead from higher layers. The setup was tested with a 3COM USB Adapter and a 3COM PCMCIA Card.

Index Terms - Bluetooth, Bayanihan Linux, BlueZ, Host Controller Interface

I. Introduction

Linux and Bluetooth Wireless Technology have gained popularity in the local and international market. In order to take advantage of this, an application utilizing the Bluetooth Technology and Linux was developed. Moreover, the application was developed on top of the Host Controller Interface to investigate the layer since it serves an important role in the protocol stack. The application was named BTChatter. The ASTI Bayanihan Linux (BL) v2.0 operating system was used in the development. It includes BlueZ, the official Linux protocol stack for the Bluetooth Technology. The application was developed using the HCI module of BlueZ. The enablers used were 3COM USB Adapter and 3COM PCMCIA Card.

[®]The *Bluetooth*® word mark and logos are owned by the Bluetooth SIG, Inc. and any use of such marks by ASTI is under license.
Other trademarks and trade names are those of their respective owners.

^{*}**William R. Cheung** and **Joel T. Fallorina**, undergraduate students of Computronix College in Dagupan City, worked with ASTI under the Presidential Summer Youth Workshop Program Year 2003.

^{*}DOST-ASTI is the research and development institute of the Philippine government mandated to pursue R&D in the advanced fields of Microelectronics and Communications technologies.

II. The Tools Used

2.1 *The ASTI Bayanihan Linux v2.0*

The ASTI Bayanihan Linux is an easy-to-install Linux distribution primarily developed for desktop use. It is a customized distribution based on the Red Hat Linux 8.0 Psyche release. BL is a project of the Open Source Group of ASTI

The name Bayanihan Linux was derived from the word *Bayanihan*, a Filipino tradition which signifies working together for the common good. The Open Source movement exhibits the spirit of Bayanihan, a virtual community working together to develop, produce, and maintain software that everyone can freely use.

Some features of BL version 2.0 include:

- Linux Kernel 2.4.18
- Easy to install with user-friendly interface
- KDE 3.0
- Office Suite applications
- Support for audio, video, and multimedia applications
- BlueZ Bluetooth Protocol Stack and Tools for Bluetooth applications development (The included kernel has been recompiled and customized to particularly support Bluetooth technology, the BlueZ stack, and the applications development tools.)

2.2 *The Bluetooth Wireless Technology*

Bluetooth wireless technology is a low cost, low power, short-range radio technology utilizing the 2.4 GHz ISM (Industrial, Scientific and Medical) band. It was originally developed as a cable replacement technology and evolved into a wireless personal area network (WPAN) technology. It provides a universal link to connect different devices without the hassle of connecting and disconnecting different kinds of wires and cables [1].

The technology defines a software protocol stack to enable devices to work with other devices from different manufacturers [1]. **Figure 1** shows the architecture of this protocol stack.

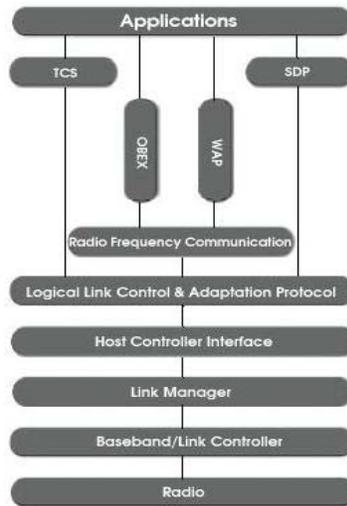


Figure 1. The Bluetooth Protocol Stack

The most common implementation of the Bluetooth protocol stack is the two processor architecture (see **Figure 2**). The stack in a two processor architecture is implemented partly in the host (higher layers of the stack, i.e. Logical Link Control and Adaptation Protocol and above) which may reside in the computer and partly in the target module (lower layers i.e. Link Manager Protocol and below).

Applications can be developed over the higher layers of the protocol stack. In this paper, we shall focus only on one layer, the Host Controller Interface.

2.3 The Host Controller Interface

The Host Controller Interface (HCI) is the interface between the host processor and its target processor [1]. This layer provides a uniform command method that enables access to Bluetooth hardware capabilities [3].

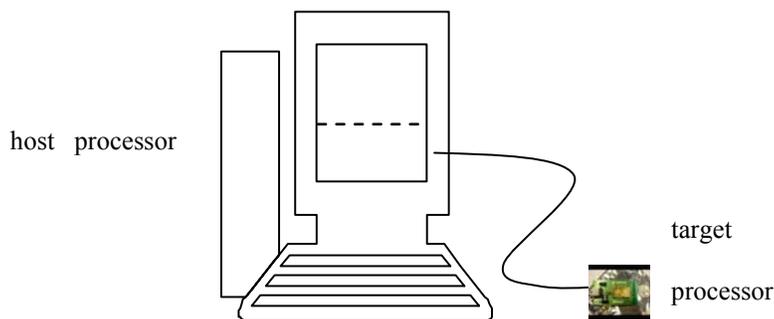


Figure 2. The two processor architecture

HCI provides the mechanisms for the actual data transfer. The HCI layer of the stack we used communicates through a Universal Serial Bus (USB) port. The initial step in the communication process is to know the packet structure of the data that the HCI layer understands [2]. This layer uses three packet types (see **Figure 3**): *commands* that go from host to module, *events* that go from module to host, and *data* that travel in both directions. Between them, these three packet types can be used to completely control a Bluetooth module and to transfer any data required [1].

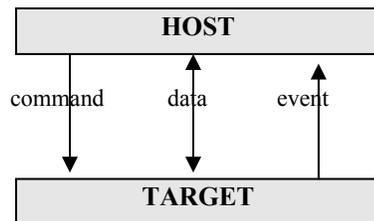


Figure 3. The HCI packets

Aside from completely controlling a Bluetooth module, HCI commands allow the host to:

- control, setup, tear down, and configure links.
- set the link's power saving modes and role switch policies.
- control many baseband features such as timeouts.
- retrieve status information on a module.
- invoke Bluetooth's test modes for factory testing and for Bluetooth qualification [1]

III. Working with the HCI layer

BlueZ as a whole consists of a set of kernel modules which implements the Bluetooth protocol stack and some programs to control the stack and to use the stack to communicate. The HCI implementation in BlueZ was used in this project.

Since the BTChatter utilizes the HCI layer, it requires HCI packet construction, sending and receiving; breakup of a large packet to smaller ones (in sending side);

combining small packets to larger ones (in receiving side), and application level flow control. **Figure 4** shows the test setup of the BTChatter application.

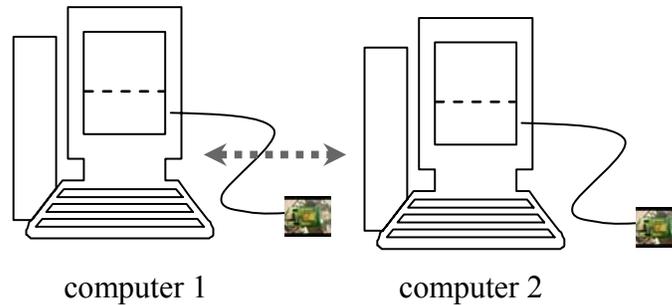


Figure 4. Communication setup of the BTChatter application

3.1 Sending ACL data packets

In constructing an Asynchronous Connectionless (ACL) packet we made use of the primitive function `writew()`. Here we put the type of packet, the header (composed of handle, flags and data length) and the payload data. The function that was used to send a packet in HCI layer was named `hci_send_acl()`. The definition of this function is listed below.

```
int hci_send_acl (
    int          dd,
    uint16_t     handle,
    uint16_t     dlen,
    void         *data,
    uint16_t     flags )
```

The largest packet that can be sent in a single `hci_send_acl()` call is hardware dependent [1]. The `hci_get_buffer_size` command is used to determine the size of the largest packet [1]. We used the `G_LOCK()` and `G_UNLOCK()` macros to prevent race conditions, since our program is a multi-threaded application [4].

3.2 Receiving ACL packets

To be able to receive ACL data packets we used the primitive `read()` function. It also has a timeout parameter so that `flag` and `error` can be used if packets don't arrive in time. It uses the `hci_data` structure for convenience. The HCI data structure is:

```
struct hci_data{
    uint16_t     handle;
    uint16_t     dlen;
    void         *data;
    uint16_t     flags;
};
```

The receiving function is `hci_recv_data()`. This function receives the single packet coming from the module. The definition of this function is listed below.

```
int hci_recv_data(
    int          dd,
    struct hci_data *d,
    int          to)
```

The command `poll()` is used to determine if any packets or data have arrived. If there is data, there is a function (`IsdataIn()`) that determines if the packet is going out (from `hci_send_acl()`) or coming in (from the opposite side of connection). Any packets that are going out are ignored. The packets that are coming in are read and the structure is filled.

3.3 Fragmenting the packets

The largest data that can be sent in a single `hci_send_acl()` call is hardware dependent. In the module it is 128 bytes. This can be determined by using the `hci_read_buffer_size` command. The mere 128 bytes are not enough for the maximum size of the message that the program should be able to send. Therefore, a large packet has to be broken down into small ones, that can be transferred in a single call. A header has also been created (aside from ACL header) so that the packet can be reconstructed reliably. The header is constructed as follows:

```
typedef struct {
    uint16_t  type;
    uint8_t   num;
    uint16_t  dlen;
}strm_hdr;
```

The implementation first determines the buffer size of the hardware. It then constructs the bigger packet from the header and data. Then in a loop it sends each fragment until all fragments are sent. If the number of fragments is more than the module can buffer, the function waits for the `hci_number_of_completed_packets_event` before continuing sending to prevent buffer overflow. The number of packets that can be sent without waiting for a `hci_number_of_completed_packets_event` is also determined from the `hci_read_buffer_size` command. When all the fragments are sent, the function now waits for a reply. This is the acknowledging scheme to make the transfer reliable. If no reply is received, the function will send the packets again up to several times until `HCI_NUM_OF_TRY` is reached. After which, the following messages will be issued: `connection timed out` and the function returns `(-1)` indicating an error. Once a reply is received, the function exits with a return value of zero `(0)`.

3.4 Combining the fragments

The fragments are then combined after receiving the packets. The header is read to determine the length of the packets. The application continues reading until the maximum length is reached and returns the value zero (0) upon success and (-1) upon failure or timed out.

The implementation first sets the filters so that only ACL packets are received. It reads the data from the socket, and checks whether this is the start or the continuation of packets. If it is the start of the packets, the header is copied. The header gives important information such as the length of the payload data. Any arriving continuation packet will be concatenated to the previous packet. When the payload length is reached, the program returns. By now, it is apparent that the function did not send any acknowledgment or reply to the sender. This is because in a multi-threading environment one thread might receive the packet but the other might not. The receiving thread has the responsibility of acknowledging the packet and indicating that it has really received those that are bound to it.

IV. Results

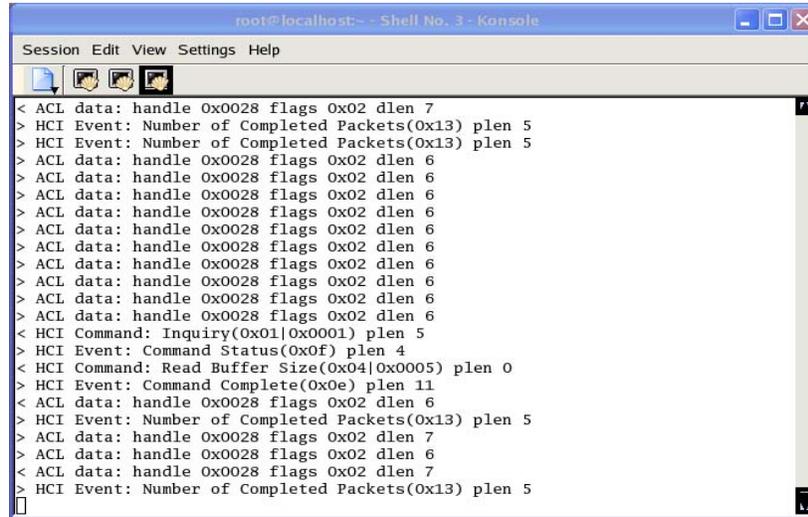
Figure 5 shows the chat and file sending action of the main window of the BTChatter. All messages sent by both the sending and receiving ends will be displayed in the Chatter View. Clicking the Send File button will launch the File selector window. If no file is chosen, an error messages is generated. After choosing a file, the receiving end will be notified and will be asked to save or discard the file. If the receiving end accepts and saves the file, transfer then starts.



Figure 5. The file transfer in progress

The output of the application was analyzed through the output of the hcidump application. The hcidump application is the HCI packet analyser of BlueZ that

determines the direction and type of data that goes through HCI. **Figure 6** shows the output of `hcidump` while sending a 10.4 MB file. The `hcidump` shows the individual packets moving in the HCI layer. Each of the data packets shown here is either a part of the file or an acknowledgement packet.



```
root@localhost:~ - Shell No. 3 - Konsole
Session Edit View Settings Help
< ACL data: handle 0x0028 flags 0x02 dlen 7
> HCI Event: Number of Completed Packets(0x13) plen 5
> HCI Event: Number of Completed Packets(0x13) plen 5
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 6
< HCI Command: Inquiry(0x01|0x0001) plen 5
> HCI Event: Command Status(0x0f) plen 4
< HCI Command: Read Buffer Size(0x04|0x0005) plen 0
> HCI Event: Command Complete(0x0e) plen 11
< ACL data: handle 0x0028 flags 0x02 dlen 6
> HCI Event: Number of Completed Packets(0x13) plen 5
> ACL data: handle 0x0028 flags 0x02 dlen 7
> ACL data: handle 0x0028 flags 0x02 dlen 6
> ACL data: handle 0x0028 flags 0x02 dlen 7
> HCI Event: Number of Completed Packets(0x13) plen 5
```

Figure 6. The output of `hcidump`

The `hcidump` result of each module was used to verify packet directions, sizes and handles. Each packet was monitored from the sending to the receiving side to check that the connection is correct. With connection established, data transfer between computers is now operational.

V. Conclusion

Direct data transfer is possible over the Host Controller Interface of the Bluetooth Protocol Stack. The HCI layer can be directly used in data transfer applications. Development and implementation of the chat program with file transfer feature on top of the HCI layer reduces the overhead from higher layers since the control headers from the other layers will not be used. The only tradeoff is that there are extra work like fragmenting and combining packets. This project used only the HCI layer to test its capabilities and limitations, since it is essential for the proper operation of the higher layers.

It was also shown that Bluetooth applications development is possible with Bayanihan Linux v2.0 (BLv2.0). The BlueZ module in BLv2.0 was used in the implementation of the project.

VI. Recommendation

The program automatically inquires and connects to all Bluetooth modules. This consumes a lot of time. In order to save time, it is better to add the inquiry and connection code to the Bluetooth modules. In addition, the data transfer rate may be increased by improving the sending action of the messages for chat and file transfer.

The BTChatter application can be enhanced by making it a native KDE application. Also, the Bayanihan Linux version 3.0 (BL3) is already available, it is recommended that enhancements/applications be developed on top of BL3.

We also encourage developers to work on value-added applications. There are endless applications that a developer can do on top of the HCI layer such as broadcast messaging, multi-player games, image transfer, video and audio streaming, etc.

VII. References

1. Jennifer Bray and Charles F. Sturman, "BLUETOOTH: Connect Without Cables," Prentice Hall PTR, Upper Saddle River, New Jersey 07458 (2001).
2. <http://www.palowireless.com/infotooth/tutorial/hci.asp>
3. J. Ballesteros, M. Borres, et al. Developing a Windows 2000 Serial Driver for Bluetooth, "Philippine Journal of ICT and Microelectronics", Volume 1 Number 2 *July 2002).
4. <http://developer.gnome.org/doc/API/>
5. <http://www.gtk.org/api/>
6. <http://sourceforge.net/mailarchive/forum.php?forum=bluez-users>

Acknowledgement

The authors would like to thank the Mobile Systems and Applications Group (MSAG) of the Advanced Science and Technology Institute (ASTI). MSAG is composed of the following people:

Project Leader: Bienvenido H. Galang Jr.

Members: Emmanuel Balintec, Janice M. Ballesteros, Mabeth M. Borres, Lucelle C. Botardo, Anne Margrette Q. Caccam, and Billy S. Pucyutan

The authors appreciate the trust, support, and pieces of advice that the group provided in order to make this project successful.

MSAG is part of the Computer Software Division of ASTI. Its mission is to empower Filipino software developers with tools such as the Bayanihan Linux v2.0 and to conduct technology transfer activities through trainings and workshops such as the PSWYP.

®The *Bluetooth*® word mark and logos are owned by the Bluetooth SIG, Inc. and any use of such marks by ASTI is under license.
Other trademarks and trade names are those of their respective owners.

* **William R. Cheung** and **Joel T. Fallorina**, undergraduate students of Computronix College in Dagupan City, worked with ASTI under the Presidential Summer Youth Workshop Program Year 2003.

* DOST-ASTI is the research and development institute of the Philippine government mandated to pursue R&D in the advanced fields of Microelectronics and Communications technologies.