

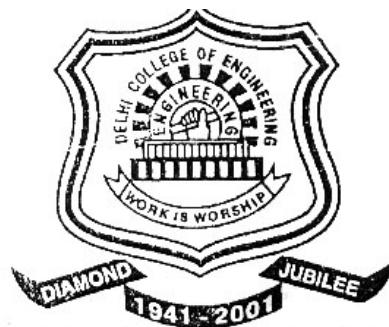
BLUETOOTH- COMMUNICATING AND EXCHANGING DATA WITH ANOTHER DEVICE, SECURITY

*A dissertation submitted in partial fulfillment of the requirements
for the award of the degree
of*

MASTER OF ENGINEERING
in
ELECTRICAL ENGINEERING
(With Specialization in Control and Instrumentation)

Submitted by
PRAVEEN DARSHANAM
(College Roll No. 12/ C&I/ 04)
(Delhi University Roll No. 8670)

Under the guidance of
Dr. PARMOD KUMAR
Professor and Head of the Department
Department of Electrical Engineering



DELHI COLLEGE OF ENGINEERING
(UNIVERSITY OF DELHI)
BAWANA ROAD, DELHI -110042
INDIA

JUNE 2006

CERTIFICATE

This is to certify that the work being presented in this dissertation entitled “**BLUETOOTH-COMMUNICATING AND EXCHANGING DATA WITH ANOTHER DEVICE, SECURITY**”, in partial fulfillment of the requirement for the award of the degree of Master of Engineering in Electrical Engineering with specialization in Control and Instrumentation submitted by **PRAVEEN DARSHANAM (8670)** to the Department of Electrical Engineering, Delhi College of Engineering, is the record of the student’s own work carried out under my supervision and guidance.

Dr. PARMOD KUMAR
Prof. and Head of Department,
Dept. of Electrical Engineering,
Delhi College of Engineering,
Delhi - 110042

ACKNOWLEDGEMENT

This report, as we see today is an outcome of persistent effort and a great deal of dedication and it has drawn intellectual and moral support from various people within the institution. I am extremely indebted to my honorable guide **Dr. PARMOD KUMAR, Prof. and Head of the Department, Department of Electrical Engineering** for his invaluable support and guidance throughout the development of this project. It is a matter of great pride for me to have worked under him, which in itself was a source of inspiration for me to complete the project with great enthusiasm, energy and determination. I also give extra special thanks to him for dedicating his valuable time whenever I needed to discuss project related work.

I take this opportunity to thank all members of Electrical Engineering Department for their valuable help in this project.

I am also thankful to all my friends who continuously helped and motivated me during the course of this dissertation.

PRAVEEN DARSHANAM
M.E. (Control & Instrumentation)
Roll No. 12/ C &I/ 04
University Roll No. 8670
Department of Electrical Engineering,
Delhi College of Engineering

CONTENTS

CERTIFICATE	i
ACKNOWLEDGEMENT	ii
CONTENTS	iii
LIST OF FIGURES	vi
LIST OF TABLES	viii
ACRONYMS	ix
ABSTRACT	x
CHAPTER I- INTRODUCTION	
1.1 Introduction	1
1.2 Objective	2
1.3 Proposed solution	2
1.4 Features of Wireless Communication	2
1.5 Bluetooth Overview	
1.5.1 Bluetooth Name and History	2
1.5.2 What is Bluetooth?	3
1.5.3 Bluetooth Special Interest Group (SIG)	4
1.6 Dissection of Dissertation	4
CHAPTER II- REVIEW WORK	
2.1 Introduction	6
2.2 Bluetooth	6
2.3 Linux Bluetooth stacks: OpenBT and BlueZ	7
2.4 Linux	8
2.5 Conclusion	9
CHAPTER III- BLUETOOTH TECHNOLOGY BASICS	
3.1 Introduction	10
3.2 Working of Bluetooth	
3.2.1 Frequency Hopping	10
3.2.2 Power Consumption	11
3.2.3 Security	12
3.3 Bluetooth Architecture	
3.3.1 The Bluetooth Protocol Stack	16
3.3.2 The Bluetooth Profiles- A Hierarchy of Groups	17
3.4 Bluetooth Specification	21
3.5 Conclusion	22
CHAPTER IV- THE BLUETOOTH PROTOCOL STACK	
4.1 Introduction	23
4.2 Bluetooth Module-Lower Protocols of the Transport Group	

4.2.1	Radio	23
4.2.2	Baseband	
4.2.2.1	Bluetooth Device Address	25
4.2.2.2	Masters, Slaves, Piconets and Scatternets	26
4.2.2.3	System Timing	26
4.2.2.4	Physical Links: SCO and ACL	27
4.2.2.5	Bluetooth Packet Structure	28
4.2.2.6	Packet Types	29
4.2.2.7	Bitstream Processing	32
4.2.3	Link Controller	32
4.2.4	Link Manager	33
4.2.5	Host Controller Interface	34
4.3	The Bluetooth Host- Upper Protocols of the Transport Group	
4.3.1	The L2CAP Layer	34
4.3.2	The RFCOMM Middleware Protocols	36
4.3.3	SDP Middleware Protocols	36
4.4	Conclusion	37

CHAPTER V- MAKING BLUETOOTH ENABLED PERSONAL COMPUTER

5.1	Introduction	38
5.2	Setting up BlueZ	
5.2.1	Obtaining BlueZ	38
5.2.2	Requirements	39
5.2.3	Compilation and Installation	39
5.2.4	Loading BlueZ Modules	40
5.2.5	Device Initialization	41
5.2.6	Debugging the BlueZ Driver	42
5.3	Conclusion	43

CHAPTER VI- ESTABLISHING A LINK BETWEEN PC AND MOBILE

6.1	Introduction	44
6.2	Dongles	44
6.3	Transport protocol	45
6.4	Port numbers and the Service Discovery Protocol	47
6.5	Establishing network connections	51
6.6	Conclusion	53

CHAPTER VII- EXPERIMENTATION RESULTS AND DISCUSSION 54

CHAPTER VIII- CONCLUSION AND FURTHER WORK 64

REFERENCES	65
APPENDIX I	67
APPENDIX II	70

LIST OF FIGURES

Figure 1.1 Bluetooth- Wire replacements Technology

Figure 3.1 The Bluetooth protocol stack

Figure 3.2 Cross-section of the Bluetooth protocol stack

Figure 3.3 Dependencies of the Bluetooth profiles

Figure 3.4 Bluetooth Headset

Figure 4.1 Format of BD_ADDR.

Figure 4.2 Bluetooth Clock

Figure 4.3 Bluetooth Packet Structure

Figure 4.4 Packet Header format

Figure 4.5 Format of the FHS payload

Figure 4.6 LMP PDU payload body

Figure 4.7 The HCI Layer

Figure 4.8 Interoperability with existing protocols & applications

Figure 5.1 BlueZ Overview Diagram

Figure 5.2 a Starting Bluetooth Services

Figure 5.2 b Starting Bluetooth Services

Figure 5.3 List of modules starting with b, h, l and r

Figure 6.1 Bluetooth USB Dongle Name

Figure 6.2 Inquiring

Figure 6.3 Paging

Figure 6.4 Shows the Connection

Figure 7.1 Name of the Bluetooth Device (Mobile)

Figure 7.2 Technical Information of the Bluetooth Device (Mobile)

Figure 7.3 Different Packets in which data is sent

Figure 7.4 Pinging Devices

Figure 7.5 HCI Packet Analyzer

Figure 7.6 Hcitol

Figure 7.7 Bluetooth USB Dongle is not connected

Figure 7.8 Repetition of up and down cycles

Figure 7.9 Starting Bluetooth

Figure 7.10 hciconfig -a

Figure 7.11 Dial-Up Networking (DUN)

LIST OF TABLES

Table 1 Bluetooth Device Class

Table 2 Radio Parameters

Table 3 Description of the FHS payload

Table 4 Comparison of the protocols

Table 5 Port numbers and their terminology for various protocols

ACRONYMS

ACL	Asynchronous connection-oriented
CAC	Channel Access Code
DAC	Device Access Code
DH	Data-High Rate
DM	Data - Medium Rate
eSCO	extended Synchronous connection-oriented
HCI	Host Controller Interface
IAC	Inquiry Access Code
IEEE	Institute of Electrical and Electronic Engineers
L2CAP	Logical Link Control and Adaptation Protocol
LCP	Link Control Protocol
LM	Link Manager
LMP	Link Manager Protocol
LSB	Least Significant Bit
MAC	Medium Access Control
Mbps	Million (Mega) bits per second
MTU	Maximum Transmission Unit
OBEX	OBject EXchange protocol
PPP	Point-to-Point Protocol
RSSI	Received Signal Strength Indicator
SCO	Synchronous connection-oriented
SDP	Service Discovery Protocol
UART	Universal Asynchronous Receiver- Transmitter
UUID	Universally Unique Identifiers
WLAN	Wireless Local Area Network

ABSTRACT

Bluetooth is a newly proposed protocol for local wireless communication and has become a de facto standard for short-range ad hoc radio connections. This report provides a study on the various protocol stacks. This thesis will also bring up the different layers in the BlueZ Bluetooth Stack. After an overview of the general Bluetooth protocols, we discussed how to enable a normal Desktop PC to a Bluetooth enabled PC and establish a link between different devices. Security concern is one of the most important problems delaying the mass adoption of Bluetooth. Some weaknesses of the Bluetooth security strategies are analyzed, together with potential risks and possible attacks against the vulnerabilities. Corresponding counter measures are also proposed in order to improve the Bluetooth security.

1.1 Introduction

IrDA (Infrared Data Association) [14], [15] specifies a cable replacement technology for point-to-point communication between ad hoc data access points, with a maximum distance of 3 feet. This technology requires Line Of Sight (LOS) communication within a 30° narrow angle cone at rates between 9600 bps and 16Mbps. IrDA transceivers operate at a transmit power of 100mW and are identified by 32-bit physical addresses. The main applications for IrDA are simple data transfer and synchronization between static devices at a short range. An example application is the exchange of business cards between two handheld computers that are equipped with IrDA ports.

- IrDA devices must be very close, no more than about 1 meter apart.
- The communicating devices must have a direct line of sight to each other. Because it relies on radio waves.

Bluetooth [1], [2], [3] communication overcomes these strict requirements. This makes Bluetooth communication much more flexible and robust. It's also important to note that because Bluetooth excels at low-bandwidth data transfer, it is not intended as a replacement for high-bandwidth cabled peripherals. For high-bandwidth devices, such as external hard drives or video cameras, cables are still the best option.

Bluetooth is the name of a new short-range, low bandwidth wireless communication technology. Many people might have heard of Bluetooth but few had experienced it hands-on. This technology is designed to be small enough to include in portable devices such as mobile phones and personal digital assistants (PDAs) but many usage scenarios also involve laptops, desktops, printers, cameras and other types of devices. Several consumer accessories featuring Bluetooth support are already into the market.

1.2 Objective

This report considers how to make a Personal Computer to Bluetooth enabled. And establish a communication link between the Desktop PC and the other Bluetooth device (say mobile phone). We will also discuss about the software and hardware (Bluetooth USB Dongle) required to connect the devices.

1.3 Proposed Solution

Use Linux as operating system because it provides support to Bluetooth. Use BlueZ [20] as the Linux stack to make a Personal Computer to Bluetooth enabled. And the hardware used is Orchid USB [17] Bluetooth Dongle to establish a link between the devices. While transferring the data we have to take care that Wi-Fi [16] or Microwave ovens are not operating. These technologies affect each other because all operate in a frequency spectrum of 2.4 GHz.

1.4 Features of Wireless Communication

The two most important phenomena impacting telecommunications over the past decade have been the explosive parallel growth of both the Internet and mobile telephone services. The Internet brought the benefits of data communications to the masses with email, the Web, and eCommerce; while mobile service has enabled "follow-me anywhere/always on" telephony.

Wi-Fi and cellular networks are rapidly converging, opening the way for innovative services and usage scenarios that benefit both end users and mobile carriers. The Wi-Fi Alliance is a strong proponent of Wi-Fi Mobile Convergence (WMC) and has prioritized efforts to make the integration of Wi-Fi and mobile devices easier and to improve the performance of the voice and multimedia applications that are at the core of converged services.

1.5 Bluetooth Overview

1.5.1 Bluetooth Name and History

Harald I Bluetooth (Danish Harald Blåtand) was the King of Denmark between 940 and 985 AD. The name "Blåtand" was probably taken from two old Danish words; 'blå' meaning dark skinned and 'tan' meaning great man. He was born in 910 and by 960 he was at the height of his powers, ruling over both Denmark and

Norway. Harald conquered all of Denmark and Norway and made the Danes Christian. Harald was killed in a battle in 985 AD. Harald completed the country's unification begun by his father, converted the Danes to Christianity, and conquered Norway. Old Harald Bluetooth united Denmark and Norway, Bluetooth of today will unite the worlds of computers and telecom.

1.5.2 What is Bluetooth?

Bluetooth wireless technology is a short-range communications system intended to replace the cable(s) connecting portable and/or fixed electronic devices. The key features of Bluetooth wireless technology are robustness, low power [8], and low cost. The Bluetooth core system consists of an RF transceiver, baseband [3], and protocol stack. The system offers services that enable the connection of devices and the exchange of a variety of classes of data between these devices over relatively short ranges. WLANs and Bluetooth™ both operate under Federal Communications Commission (FCC) Part 15 rules.



Figure 1 Bluetooth- Wire replacement Technology

DEVICE CLASS	TYPE	STRENGTH	RANGE (METERS)
Class 1 Devices	High	100 mW (20 dBm)	Up to 100
Class 2 Devices	Medium	2.5 mW (4 dBm)	Up to 10
Class 3 Devices	Low	1 mW (0 dBm)	Well within 1

Table1 Bluetooth Device Class

1.5.3 Bluetooth Special Interest Group (SIG)

In 1994 Ericsson Mobile communications initiated a study to investigate the feasibility of a low-power low-cost radio interface between mobile phones and their accessories. In Feb 1998, five companies Ericsson, Nokia, IBM, Toshiba and Intel formed a Special Interest Group (SIG). The group contained the necessary business sector members - two market leaders in mobile telephony, two market leaders in laptop computing and a market leader in digital signal processing technology. It was estimated that before year 2002, Bluetooth was a built-in feature for more than 100 million mobile phones and several million communication devices ranging from handsets and portable PCs to desktop computers and notebooks.

1.6 Dissection of Dissertation

The thesis is organized as follows

Chapter I deals with the Objective of the Thesis and the proposed solution to the problem.

Chapter II describes the review work of the project viz. hardware's and software's used

Chapter III deals with the Bluetooth Technology basics. This covers the Bluetooth Modules which covers the lower parts of the system from radio up to the host interface. And the Bluetooth Host which describes the higher parts of the system which would typically exist on or in a host system that is "Bluetooth Enabled".

Chapter IV describes about the official Linux Bluetooth stack, BlueZ. BlueZ was supported by Linux even before the kernel versions of 2.4.21. But from kernels of 2.4.21 and above BlueZ has become a part of Linux.

Chapter V describes how to configure a Linux kernel so that we can make a normal Desktop PC to Bluetooth enabled PC. We also discussed about the various software's and hardware's required to configure the Linux kernel.

Chapter VI discusses how to establish a communication link between two devices. The devices used in this project are a Personal Computer (64bit AMD Athlon, 512MB Simmtronics RAM) and a mobile phone (Nokia 6600).

Chapter VII deals with experimentation results and discussion related to them.

Chapter VIII explains about the further work that can be done on Bluetooth technologies and concludes the thesis.

2.1 Introduction

Bluetooth represents a very simple proposition, obviating the need for connectivity via physical wires; this is itself the powerful added value for Bluetooth. Bluetooth is a WPAN technology. A great deal of interest, talent and energy has marshaled around this existing new technology. Until now most of the information available about Bluetooth wireless communications has been from the SIG's official website or from brief press articles or news letters. The Bluetooth SIG was formed early in 1998 by Ericsson, Intel, IBM, Nokia [19] and Toshiba to develop an open specification for globally available short-range wireless radio frequency communications.

2.2 Bluetooth

Bluetooth wireless technology was conceived by engineers at Swedish telecommunications manufacturer Telefonaktiebolaget LM Ericsson who realized the potential of global short-range wireless communication. In 1994 Ericsson Mobile Communication began a study of radio links which aren't directional and it doesn't need line of sight, so it has obvious advantages over the infra-red links to replace cables that linked its mobile phones with accessories. Out of this study was born the specification [1] for Bluetooth Wireless technology.

A diverse set of wired and wireless devices are Bluetooth connectable, including office appliances, e.g. desktop PCs, printers, projectors, laptops, and PDAs; communication appliances, e.g. speakers, handsets, pagers, and mobile phones [19]; home appliances, e.g. DVD players, digital cameras, cooking ovens, washing machines, refrigerators, and thermostats. Bluetooth is suitable for a wide range of applications, e.g. wireless office and meeting room, smart home and vehicle, intelligent parking, electrical paying and banking.

Bluetooth adopts master-slave architecture to form an ad hoc wireless network named piconet [13]. A master in a piconet may communicate with up to seven active

slaves. Several connected piconets can further form a scatternet [10]. Bluetooth was developed by Bluetooth Special Interest Group (SIG) formed in May 1998. The founding members included Ericsson, Intel, IBM, Nokia, and Toshiba. Since then, almost all of the biggest telecommunications companies have joined the Bluetooth SIG, e.g. 3Com, Lucent, Microsoft, Motorola, etc.

Bluetooth specification is a free open standard and the first Version 1.0 came out in 1999. The next version was 1.1 released in February 2001. Version 1.2 was released in early 2004. The Bluetooth Core Specification 1.2 has been significantly restructured for better consistency and readability. The new Version 2.0 was released in November 2004 with an extra feature of Enhanced Data Rate (EDR). The Bluetooth Core Specification version 2.0 + EDR introduces Enhanced Data Rate (EDR). EDR provides a set of additional packet types that use the new 2 Mbps and 3 Mbps modes. The most important structure changes have been performed in Baseband, LMP, HCI and L2CAP. As the Bluetooth Specification continues to evolve, some features, protocols, and profiles are replaced with new ways of performing the same function. Often these changes reflect the evolution of the communications industry.

Implementations with versions 1.1 and 1.2 reach speeds of 723.1 kbit/s. Version 2.0 implementations feature *Bluetooth Enhanced Data Rate (EDR)*, and thus reach 2.1 Mbit/s. Technically version 2.0 devices have a higher power consumption, but the three times faster rate reduces the transmission times, effectively reducing consumption to half that of 1.x devices (assuming equal traffic load).

2.3 Linux Bluetooth stacks: OpenBT and BlueZ

There are two totally different Bluetooth stacks available for Linux, both of them are open source. OpenBT is mostly a contribution of Axis, and BlueZ [20] is originally a contribution of Qualcomm. BlueZ is the official Linux Bluetooth stack. BlueZ is a powerful Bluetooth communications stack with extensive APIs that allows a user to fully exploit all local Bluetooth resources, but it has no official documentation. It provides support for core Bluetooth layers and protocols. The main architectural difference between the stacks is their interfaces, both to drivers and applications. OpenBT uses a *serial* abstraction, and BlueZ uses a *network* abstraction.

The strengths of OpenBT 0.8 are:

- More mature, used in real products
- Wider set of utilities and documentation
- Support BCSP (and a wider range of hardware functions)
- Easier to hook your programs to RFCOMM

The strengths of BlueZ 1.2 are:

- Included in standard Linux kernel, better integration
- Modular design, more than one Bluetooth port per stack
- Excellent USB support
- hcidump

The additional strengths of BlueZ 2.4 are:

- Excellent PAN/BNEP support (complete and flexible)
- Excellent RFCOMM support (now as easy as OpenBT, but more flexible)
- Supported in OpenObex 1.0.0 (Obex over RFCOMM)

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, 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.

2.4 Linux

Linux is an operating system for PC platforms with many advantages for developers. The software code of the Linux kernel and most of the applications are open source and freely downloadable from the Internet at no cost. The open source community does development and maintenance of the code. Any problem regarding the software can be thrown into the community. The response out of the community is usually quick and to the point. Last but not least: Linux has proven to be a very stable

operating system. For the new Fedora Core 5 the kernel version is 2.6.15 and also it includes inbuilt Linux Bluetooth Protocol Stack, BlueZ.

2.5 Conclusion

As stated earlier, not all radio waves are the same; the faster and further a radio wave travels, the more energy [8], [9] it requires, here lies the difference between Bluetooth and Wi-Fi. Unlike Bluetooth, which emphasizes low power and short range and in turn offers a transmission speed of approximately 800 Kbps, Wi-Fi depends on a higher energy intake to offer a 100-meter range and 11 Mbps maximum transmission rate. This speed makes Wi-Fi more than 10 times as fast as Bluetooth and similar to a high-speed modem. For large file transfers and quick Internet access, Wi-Fi outperforms Bluetooth.

As Bluetooth and Wi-Fi began to capture the interest of the hi-tech industry, many understood the exciting potential of these technologies to revolutionize how people connect their devices. But negative publicity surfaced when analysts and members of the media speculated that the two technologies competed against each other. Most industry insiders and technology experts agree that the two wireless technologies do not compete but rather complement each other.

3.1 Introduction

The term *Bluetooth*[™] refers to an open specification for a technology to enable short range wireless voice and data communication. Bluetooth is a cable-replacement technology designed to wirelessly connect peripherals, such as mice and mobile phones, to your desktop or laptop computer and to each other. Bluetooth is an inexpensive, low-power, short-range radio-based [11], [12] technology. Bluetooth is not a wireless networking solution, such as AirPort. Rather, it is an alternative to the IrDA (Infrared Data Association) standard. Although the IrDA standard, too, supports wireless communication between peripherals and computers, it has two limiting requirements. First, IrDA devices must be very close, no more than about 1 meter apart. Second, the communicating devices must have a direct line of sight to each other. Bluetooth overcomes both these bottlenecks.

3.2 Working of Bluetooth

3.2.1 Frequency Hopping

Bluetooth radios transmit using a frequency hopping spread spectrum (FHSS) technique. This technique forces transmissions to switch between 79 different frequency channels in the ISM band (This frequency band is 2400 - 2483.5 MHz.) at 1600 hops per second. The master Bluetooth device sends its unique device address (similar to an Ethernet address) and the value of its internal clock as inputs to its slaves. This information is used to calculate the “pseudo-random” hopping order at which transmissions occur (“**pseudo**” meaning that the hopping order repeats after some time). Because the master device and all its slaves use the same algorithm with the same initial input, the connected devices always arrive together at the next frequency. At any given time, a Bluetooth device operating in master mode can communicate with up to seven slave devices will hop together in frequency. Therefore, if two or more piconets are within range of one another, the frequencies at which they transmit may match up during some time slots. This overlap in frequency and time will cause packet collisions and loss of throughput in these piconets. This means that Bluetooth technology will actually interfere with itself. This interference

will be referred to as “inter-piconet” interference. The Bluetooth specification [1] already describes methods to help reduce the amount of Bluetooth interference. Some of these methods include changing the data packet length, adding forward error correction, and the use of adaptive power control. Adaptive power control keeps Bluetooth devices from radiating more power than necessary.

After a Bluetooth device sends or receives a packet, it and the Bluetooth device or devices it is communicating with “hop” to another frequency before the next packet is sent. Compared with other systems operating in the same frequency band, the Bluetooth radio typically hops faster and uses shorter packets. This is because **short packages** and **fast hopping** limit the impact of microwave ovens and other sources of disturbances. Use of **Forward Error Correction** (FEC) limits the impact of random noise on long-distance links. FHSS scheme has three advantages:

- It allows Bluetooth devices to use the entirety of the available ISM band, while never transmitting from a fixed frequency for more than a very short time. This ensures that Bluetooth conforms to the ISM restrictions on transmission quantity per frequency.
- It ensures that any interference will be short-lived. Any packet that doesn't arrive safely at its destination can be resent at the next frequency.
- It provides a base level of security because it's very difficult for an eavesdropping device to predict which frequency the Bluetooth devices will use next.

3.2.2 Power Consumption

As a cable-replacement technology, it's not surprising that Bluetooth devices are usually battery-powered devices [9], such as wireless mice and mobile phones. To conserve power [8], most Bluetooth devices operate as low-power, 1 mW radios (Class 3 radio power). This gives Bluetooth devices a range of about 5–10 meters. This range is far enough for comfortable wireless peripheral communication but close enough to avoid drawing too much power from the device's power source.

Many tests have been done to prove that Bluetooth devices are too low in power to have any negative impact on health. Three low-power modes, which extend battery life by reducing activity on a connection, have been defined. These modes are called Park, Hold, and Sniff [2], [3].

Active Mode: In the active mode, the Bluetooth unit actively participates on the channel. The master schedules the transmission based on traffic demands to and from the different slaves. In addition, it supports regular transmissions to keep slaves synchronized to the channel. Active slaves listen in the master-to-slave slots for packets. If an active slave is not addressed, it may sleep until the next new master transmission.

Sniff Mode: Devices synchronized to a piconet can enter power-saving modes in which device activity is lowered. In the SNIFF mode, a slave device listens to the piconet at reduced rate, thus reducing its duty cycle. The SNIFF interval is programmable and depends on the application. It has the highest duty cycle (least power efficient) of all 3 power saving modes (sniff, hold & park).

Hold Mode: Devices synchronized to a piconet can enter power-saving modes in which device activity is lowered. The master unit can put slave units into HOLD mode, where only an internal timer is running. Slave units can also demand to be put into HOLD mode. Data transfer restarts instantly when units transition out of HOLD mode. It has an intermediate duty cycle (medium power efficient) of the 3 power saving modes (sniff, hold & park).

Park Mode: In the PARK mode, a device is still synchronized to the piconet but does not participate in the traffic. Parked devices have given up their MAC (AM_ADDR) address and occasional listen to the traffic of the master to re-synchronize and check on broadcast messages. It has the lowest duty cycle (power efficiency) of all 3 power saving modes (sniff, hold & park).

3.2.3 Security

Today's wireless world means that data is being sent among us invisibly from device to device, country to country, and person to person. This data in the form of e-mails, photos, contacts and addresses are precious and private to each of us. This private information, no longer making its way along wires in plain sight, needs to be sent securely to its intended recipient without interception.

Product developers that use *Bluetooth* wireless technology in their products have several options for implementing security. There are three modes of security [7] for *Bluetooth* access between two devices.

Security Mode1:	non-secure
Security Mode2:	service level enforced security
Security Mode3:	link level enforced security

The manufacturer of each product determines these security modes. Devices and services also have different security levels. For devices, there are two levels: "trusted device" and "untrusted device." A trusted device, having been paired with one's other device, has unrestricted access to all services. With regard to services, three security levels are defined: services that require authorization and authentication, services that require authentication only and services that are open to all devices.

The recently reported issues of advanced "hackers" gaining access to information stored on select mobile phones using *Bluetooth* functionality are due to incorrect implementation. The names bluesnarfing and bluebugging have been given to these methods of illegal and improper access to information. The questions and answers below provide users with more information about these current issues and will address their concerns for dealing with these security risks.

Bluejacking: Bluejacking allows phone users to send business cards anonymously using *Bluetooth* wireless technology. Bluejacking does NOT involve the removal or alteration of any data from the device. These business cards often have a clever or flirtatious message rather than the typical name and phone number. Bluejackers often look for the receiving phone to ping or the user to react. They then send another, more personal message to that device. Once again, in order to carry out a bluejacking, the sending and receiving devices must be within 10 meters of one another. Phone owners who receive bluejack messages should refuse to add the contacts to their address book. Devices that are set in non-discoverable mode are not susceptible to bluejacking.

Bluebugging: Bluebugging allows skilled individuals to access the mobile phone commands using *Bluetooth* wireless technology without notifying or alerting the phone's user. This vulnerability allows the hacker to initiate phone calls, send and receive text messages, read and write phonebook contacts, eavesdrop on phone conversations, and connect to the Internet. As with all the attacks, without specialized equipment, the hacker must be within a 10 meter range of the phone. This is a separate vulnerability from bluesnarfing and does not affect all of the same phones as bluesnarfing.

Bluesnarfing: Bluesnarfing allows hackers to gain access to data stored on a *Bluetooth* enabled phone using *Bluetooth* wireless technology without alerting the phone's user of the connection made to the device. The information that can be accessed in this manner includes the phonebook and associated images, calendar, and IMEI (international mobile equipment identity). By setting the device in non-discoverable, it becomes significantly more difficult to find and attack the device. Without specialized equipment the hacker must be within a 10 meter range of the device while running a device with specialized software. Only specific older *Bluetooth* enabled phones are susceptible to bluesnarfing.

Cabirworm: The cabir worm is malicious software, also known as malware. When installed on a phone, it uses *Bluetooth* technology to send itself to other similarly vulnerable devices. Due to this self-replicating behavior, it is classified as a worm. The cabir worm currently only affects mobile phones that use the Symbian series 60 user interface platform and feature *Bluetooth* wireless technology. Furthermore, the user has to manually accept the worm and install the malware in order to infect the phone. More information on the cabir worm is available from the software licensing company Symbian and on the websites of F-Secure, McAfee and Symantec.

Affect of PIN on security: The personal identification number (PIN) is a four or more digit alphanumeric code that is temporarily associated with one's products for the purposes of a one time secure pairing. During this process one or both devices need a PIN code to be entered, which is used by internal algorithms to generate a secure key, which is then used to authenticate the devices whenever they connect in the future. It is recommended that users employ at minimum an eight character or

more alphanumeric PIN when possible. Product owners must share that PIN number only with trusted individuals and trusted products for pairing. Without this PIN number, pairing cannot occur. It is always advisable to pair products in areas with relative privacy. Avoid pairing your *Bluetooth* enabled devices in public.

To perform this hack, it is necessary for the attacker to overhear the initial pairing process, which normally only happens once in a private environment and takes a fraction of a second. There are some possible methods to try and force a deletion of the security key in one of the two *Bluetooth* devices, and hence initiate a new pairing process, which we could then listen in to. To do this, we need to masquerade as the second device during a connection. The equipment needed for this process is very expensive and usually used by developers only. If this process succeeds the user will see a message on their device that asks them to re-enter a PIN code. If they do this while the attacker is present, and the PIN code they enter is sufficiently short, then the attack could theoretically succeed. If the PIN key that has been used consists of only four numeric characters, a fast PC can calculate the security key in less than one tenth of a second. As the PIN key gets longer, the time to crack the security code gets longer and longer. At eight alphanumeric characters it would take over one hundred years to calculate the PIN making this crack nearly impossible.

This is an academic analysis of *Bluetooth* security. What this analysis outlines is possible, but it is highly unlikely for a normal user to ever encounter such an attack. The attack also relies on a degree of user gullibility, so understanding the *Bluetooth* pairing process is an important defense.

Denial of service (DoS): The well known denial of service (DoS) attack, which has been most popular for attacking internet web sites and networks, is now an option for hackers of *Bluetooth* wireless technology enabled devices. This nuisance is neither original nor ingenious and is, very simply, a constant request for response from a hacker's *Bluetooth* enabled computer (with specific software) to another *Bluetooth* enabled device such that it causes some temporary battery degradation in the receiving device. While occupying the *Bluetooth* link with invalid communication requests, the hacker can temporarily disable the product's *Bluetooth* services. The

DoS attack only offers the hacker the satisfaction of temporary annoyance, but does not allow for access to the device's data or services – no information residing on the receiving device can be used or stolen by the attacker.

3.3 Bluetooth Architecture

Bluetooth is both a hardware-based radio system and a software stack that specifies the linkages between layers. This supports flexibility in implementation across different devices and platforms. It also provides robust guidelines for maximum interoperability and compatibility. In this section, we'll learn about:

- The **Bluetooth protocol stack**. The protocol stack is the core of the Bluetooth specification that defines how the technology works.
- The **Bluetooth profiles**. The profiles define how to use Bluetooth technology to accomplish specific tasks.

3.3.1 The Bluetooth Protocol Stack

The heart of the Bluetooth specification is the Bluetooth protocol stack. By providing well-defined layers of functionality, the Bluetooth specification ensures interoperability of Bluetooth devices and encourages adoption of Bluetooth technology. As we can see in Figure, these layers range from the low-level radio link to the profiles.

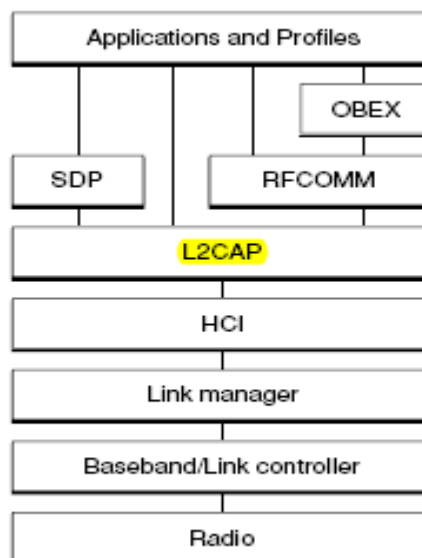


Figure 3.1 The Bluetooth protocol stack

Layer	Location
RFCOMM	Host PC (software)
L2CAP	
HCI Driver	
HCI Transport	USB/UART
Host Controller	Module (firmware)
Link Manager	
Link Controller (Baseband)	
Radio	

Figure 3.2 Cross-section of the Bluetooth protocol stack

3.3.2 The Bluetooth Profiles-A Hierarchy of Groups

The Bluetooth specification defines a wide range of profiles [1], describing many different types of tasks, some of which have not yet been implemented by any device or system. By following the profile's procedures, developers can be sure that the applications they create will work with any device that conforms to the Bluetooth specification. This section focuses on almost all profiles that different OS supports. For information on other profiles, including those still in development, see the Bluetooth specification. At a minimum, each profile specification contains information on the following topics:

- **Dependencies on other profiles.** Every profile depends on the base profile, called the generic access profile, and some also depend on intermediate profiles.
- **Suggested user interface formats.** Each profile describes how a user should view the profile so that a consistent user experience is maintained.
- **Specific parts of the Bluetooth protocol stack used by the profile.** To perform its task, each profile uses particular options and parameters at each layer of the stack. This may include an outline of the required service record, if appropriate. The Bluetooth profiles are organized into a hierarchy of groups, with each group depending upon the features provided by its predecessor. Figure 3.3 illustrates the dependencies of the Bluetooth profiles.

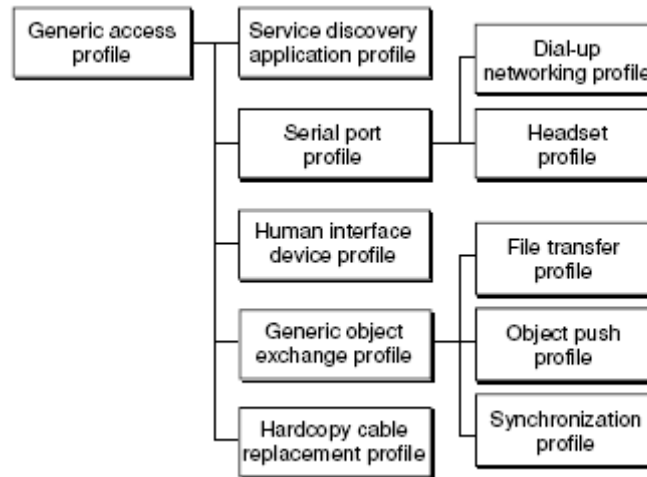


Figure 3.3 Dependencies of the Bluetooth profiles.

The Base Profile

At the base of the profile hierarchy is the generic access profile (GAP), which defines a consistent means to establish a baseband link between Bluetooth devices. In addition to this, the GAP defines:

- Which features must be implemented in all Bluetooth devices
- Generic procedures for discovering and linking to devices
- Basic user-interface terminology

All other profiles are based on the GAP. This allows each profile to take advantage of the features the GAP provides and ensures a high degree of interoperability between applications and devices. It also makes it easier for developers to define new profiles by leveraging existing definitions.

Remaining Profiles

The **service discovery application profile** describes how an application should use the SDP to discover services on a remote device. As required by the GAP, any Bluetooth device should be able to connect to any other Bluetooth device. Based on this, the service discovery application profile requires that any application be able to find out what services are available on any Bluetooth device it connects to.

The **human interface device (HID) profile** describes how to communicate with a HID class device using a Bluetooth link. It describes how to use the USB HID

[17] protocol to discover a HID class device's feature set and how a Bluetooth device can support HID services using the L2CAP layer.

As its name suggests, the **serial port profile** defines RS-232 serial-cable emulation for Bluetooth devices. As such, the profile allows legacy applications to use Bluetooth as if it were a serial-port link, without requiring any modification. The serial port profile uses the RFCOMM protocol to provide the serial-port emulation.

The **dial-up networking (DUN) profile** is built on the serial port profile and describes how a data-terminal device, such as a laptop computer can use a gateway device, such as a mobile phone or a modem to access a telephone-based network. Like other profiles built on top of the serial port profile, the virtual serial link created by the lower layers of the Bluetooth protocol stack is transparent to applications using the DUN profile. Thus, the modem driver on the data-terminal device is unaware that it is communicating over Bluetooth. The application on the data-terminal device is similarly unaware that it is not connected to the gateway device by a cable.

The **headset profile** describes how a Bluetooth-enabled headset should communicate with a computer or other Bluetooth device (such as a mobile phone). When connected and configured, the headset can act as the remote device's audio input and output interface.



Figure 3.4 Bluetooth Headset

The **hardcopy cable replacement profile** describes how to send rendered data over a Bluetooth link to a device, such as a printer. Although other profiles can be used for printing, the HCRP is specially designed to support hardcopy applications.

The **generic object exchange profile** provides a generic blueprint for other profiles using the OBEX protocol and defines the client and server roles for devices. As with all OBEX transactions, the generic object exchange profile stipulates that the client initiate all transactions. The profile does not, however, describe how applications should define the objects to exchange or exactly how the applications should implement the exchange. These details are left to the profiles that depend on the generic object exchange profile, namely the object push, file transfer, and synchronization profiles.

The **object push profile** defines the roles of push server and push client. These roles are analogous to and must interoperate with the server and client device roles the generic object exchange profile defines. The object push profile focuses on a narrow range of object formats for maximum interoperability. The most common of the acceptable formats is the vCard format. If an application needs to exchange data in other formats, it should use another profile, such as the file transfer profile.


The **file transfer profile** is also dependent on the generic object exchange profile. It provides guidelines for applications that need to exchange objects such as files and folders, instead of the more limited objects supported by the object push profile. The file transfer profile also defines client and server device roles and describes the range of their responsibilities in various scenarios. For example, if a client wishes to browse the available objects on the server, it is required to support the ability to pull from the server a folder-listing object. Likewise, the server is required to respond to this request by providing the folder-listing object.

The **synchronization profile** is another dependent of the generic object exchange profile. It describes how applications can perform data synchronization, such as between a personal digital assistant (PDA) and a computer. Not surprisingly, the synchronization profile, too, defines client and server device roles. The synchronization profile focuses on the exchange of personal information management (PIM) data, such as a to-do list, between Bluetooth-enabled devices. A typical usage of this profile would be an application that synchronizes your computer's and your PDA's versions of your PIM data. The profile also describes how an application can

support the automatic synchronization of data—in other words, synchronization that occurs when devices discover each other, rather than at a user’s command.

3.4 Bluetooth Specification

Here are some specification details:

- The devices in a piconet share a common communication data channel. The
- Channel has a total capacity of 1 megabit per second (Mbps). Headers and handshaking information consume about 20 percent of this capacity.
- In the United States and Europe, the frequency range is 2,400 to 2,483.5 MHz, with 79 1-MHz radio frequency (RF) channels. In practice, the range is 2,402 MHz to 2,480 MHz. In Japan, the frequency range is 2,472 to 2,497 MHz with 23 1-MHz RF channels.
- A data channel hops randomly 1,600 times per second between the 79 (or 23)
- RF channels.
- Each channel is divided into time slots 625 microseconds long.
- A piconet has a master and up to seven slaves. The master transmits in even time slots, slaves in odd time slots.
- Packets can be up to five time slots wide.
- Data in a packet can be up to 2,745 bits in length.
- There are currently two types of data transfer between devices: SCO (synchronous connection oriented) and ACL (asynchronous connectionless).
- In a piconet, there can be up to three SCO links of 64,000 bits per second each. To avoid timing and collision problems, the SCO links use reserved slots set up by the master.
- Masters can support up to three SCO links with one, two or three slaves.
- Slots not reserved for SCO links can be used for ACL links.
- One master and slave can have a single ACL link.
- ACL is either point-to-point (master to one slave) or broadcast to all the slaves.
- ACL slaves can only transmit when requested by the master.
- The official Bluetooth logo:  **Bluetooth**

3.5 Conclusion

The Bluetooth specification not only covers how to set up short range wireless links but also describes the Bluetooth qualification process. By putting their products through this process, companies that join the Bluetooth SIG can get a free license to use the Bluetooth wireless technology and Bluetooth brand. Bluetooth specification also includes different profiles which detail how applications should use the Bluetooth protocol stack. Bluetooth ensures that devices maintain time synchronization by repeatedly resynchronising to the Master's transmissions. Since the frequency hopping algorithm is based on the device clock, this also ensures that frequency hopping is in step.

CHAPTER IV

THE BLUETOOTH PROTOCOL STACK

4.1 Introduction

The elements of the stack (protocols, layers, applications and so on) are logically partitioned into three groups:

The Transport Protocol Group - Radio, Baseband, Link Manager and L2CAP

The Middleware Protocol Group- RFCOMM, TCS and SDP

It also supports third protocols like Internet-related protocols (PPP, IP, TCP), WAP, OBEX

The Application Group - This refers to the software that resides above the protocol stack supplied by device manufacturers, independent software vendors or others which exercises the protocol stack to accomplish some function that benefits the user of a Bluetooth device.

Each Bluetooth module vendor will determine how to measure the link quality.

4.2 Bluetooth Module-Lower Protocols of the Transport Group

At the bottom of the stack are the firmware layers that are usually implemented as part of the Bluetooth device itself. The Radio and the Link Controller perform low-level functions such as timing and error correction and cannot be accessed directly by the programmer. The link manager implements control functions such as link setup between devices and power modes. These are features that an application will want access to in order to control the operation of the device.

4.2.1 Radio

The Bluetooth radio implements the air interface for Bluetooth devices and involves circuitry for modulation and demodulation as well as amplifiers. It is at the lowest level of the protocol stack. The operating band of 83.5 MHz (2.4835-2.4465 GHz) is divided into 1MHz spaced channels, each signaling data at 1 mega symbols.

Bluetooth radio system is an ad hoc system that allows devices from lots of different manufacturers to communicate with one another when they come into range.

PARAMETERS	VALUES
MODULATION	G-FSK, $h \leq 0.35$
PEAK DATA RATE	1 Mb/s
RF BANDWIDTH	220 kHz (-3dB), 1 MHz (-20 dB)
RF BAND	2.4 GHz, ISM band
HOP CHANNELS	23/ 79
CARRIER SPACING	1 MHz
PEAK TX POWER	$\leq 20\text{dBm}$

Table 2 Radio Parameters

Bluetooth uses Gaussian-shaped frequency shift keying (GFSK) modulation with a nominal modulation index of $k=0.3$. This binary modulation was chosen for its robustness, and, with the accepted bandwidth restrictions, it can provide data rates to about 1Mbps. A noncoherent demodulation can be accomplished by a limiting FM discriminator. This simple modulation scheme allows the implementation of low-cost radio units, which is one of the main aims of the Bluetooth system.

4.2.2 Baseband

The baseband layer (resource manager) determines and instantiates the Bluetooth air- interface. In comparison with the cabled environment it might be said that the baseband determines the “shape” and “pin configuration” of the interface. It has two main functions. The baseband later defines the master and slave roles for devices. It also defines how the frequency hopping sequences used by communicating devices are formed and the rules for sharing the air interface among several devices; these rules are based upon a time division duplex (TDD) and packet-based polling scheme.

The device manager is the functional block in the baseband that controls the general behavior of the Bluetooth device. It is responsible for all operation of the Bluetooth system that is not directly related to data transport, such as inquiring for the presence of other nearby Bluetooth devices, connecting to other Bluetooth devices, or making the local Bluetooth device discoverable or connectable by other devices. The device manager requests access to the transport medium from the baseband resource controller in order to carry out its functions. The device manager also controls local device behavior implied by a number of the HCI commands, such as managing the device local name, any stored link keys, and other functionality. Key functions of the Bluetooth baseband are:

Piconet and device control functions: Connection creation, Frequency-hopping sequence selection and Timing

Modes of operation: Power control and secure operation

Medium access functions: Polling, packet types [5], packet processing and Link types

4.2.2.1 Bluetooth Device Address

Every Bluetooth chip ever manufactured is imprinted with a globally unique 48-bit Address [1], [2] which we will refer to as the Bluetooth address or device address. This is identical in nature to the MAC addresses of Ethernet or 802.11 LAN devices and both address spaces are actually managed by the same organization - the IEEE Registration Authority. These addresses are assigned at manufacture time and are intended to be unique and remain static for the lifetime of the chip. It conveniently serves as the basic addressing unit in all of Bluetooth programming. For one Bluetooth device to communicate with another, it must have some way of determining the other device's Bluetooth address. This address is used at all layers of the Bluetooth communication process, from the low-level radio protocols to the higher-level application protocols. In contrast, TCP/IP network devices that use Ethernet as their data link layer discard the 48-bit MAC address at higher layers of the communication process and switch to using IP addresses.

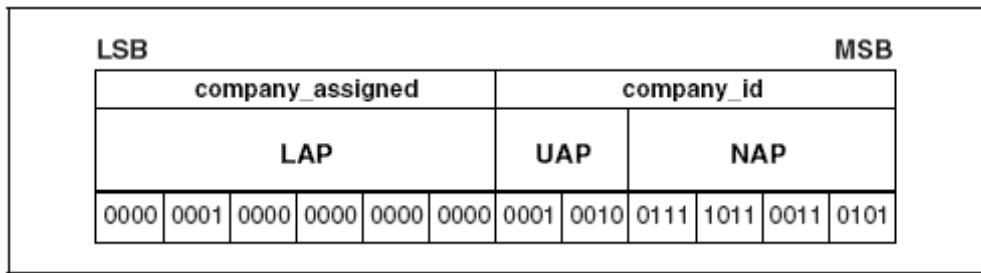


Figure 4.1 Format of BD_ADDR.

- LAP field: lower address part consisting of 24 bits
- UAP field: upper address part consisting of 8 bits
- NAP field: non-significant address part consisting of 16 bits

4.2.2.2 Masters, Slaves, Piconets and Scatternets

A Bluetooth Wireless Personal Area Network (BT-WPAN) consists of *piconets*. Each piconet is a cluster of up to eight Bluetooth devices. One device is designated as the master, and the others are the slaves. A scatternet is formed with the linking of one or more piconets. This linking is formed by the sharing of a common device called a gateway or bridge or Relay Node to form a *scatternet*. These interconnected piconets within the scatternet form a backbone for the Mobile Area Network (MANET).

A device can be both a master and a slave in different piconets. This could lead to a switch of roles between Master and Slave in the new connection. Inter-piconet communications are established over the shared unit. Time multiplexing must be used for that unit to switch between piconets. In case of **ACL** links, a unit can request to enter the **HOLD** or **PARK** mode in the current piconet, during which time it may join another piconet by just changing the channel parameters. Units in the **SNIFF** mode may have sufficient time to visit another piconet in between the sniff slots. If **SCO** links are established, other piconets can only be visited in the non-reserved slots in-between.

4.2.2.3 System Timing

Each Bluetooth device runs a 28-bit clock that is never adjusted or turned off. The clock ticks 3,200 times per second or once every 312.5 μ sec, which corresponds

to a clock rate of 3.2 KHz. The clock has an accuracy of ± 20 parts per million (ppm) but in low-power modes (e.g. standby, hold or park) the accuracy is reduced to ± 250 ppm by using a low-power oscillator. An attacker using low energy lasers (**LEL**) or electronic magnetic pulses (**EMP**) can disrupt the Bluetooth clock and disable communications between all devices. This type of attack renders any communication network inoperable. Both LEL and EMP attacks are extremely rare and there is very little risk involved by this type of attack.

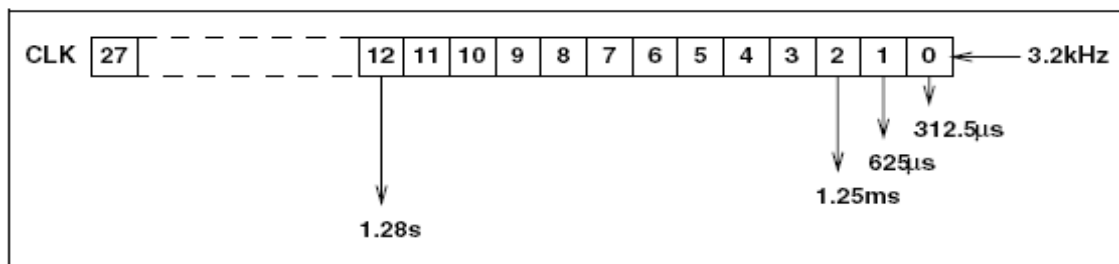


Figure 4.2 Bluetooth Clock

4.2.2.4 Physical Links: SCO and ACL

There are two basic types of physical links that can be established between a master and a slave. They are

- Synchronous Connection Oriented (SCO)
- Asynchronous Connection-Less (ACL)

An SCO link provides a symmetric link between the master and the slave, with regular periodic exchange of data in the form of reserved slots. Thus, the SCO link provides a circuit-switched connection where data are regularly exchanged, and as such it is intended for use with time-bounded information as audio. A master can support up to three SCO links to the same or to different slaves. A slave can support up to three SCO links from the same master.

An ACL link is a point-to-multipoint link between the master and all the slaves on the piconet. It can use all of the remaining slots on the channel not used for SCO links. The ACL link provides a packet-switched connection where data are exchanged sporadically, as they become available from higher layers of the stack. The traffic over the ACL link is completely scheduled by the master.

4.2.2.5 Bluetooth Packet Structure

The different types of packets [1] that are used for communicating over inter device ACL and SCO links. Packets can be breakdown into their constituent parts such as access code, packet header, payload header and payload. Every packet consists of an access code, a header and a payload.

Every packet starts with an access code. The access code is 72 or 68 bits and the header is 54 bits. The access code consists of a preamble, a sync word, and possibly a trailer. If a packet header follows, the access code is 72 bits long, otherwise the access code is 68 bits long and is known as a shortened access code. The shortened access code is used in paging, inquiry, and park. The shortened access code does not contain a trailer. This access code is used for synchronization, DC offset compensation and identification. The access code identifies all packets exchanged on a physical channel: all packets sent in the same physical channel are preceded by the same access code.

Three different access codes are defined

- device access code (DAC)
- channel access code (CAC)
- inquiry access code (IAC)

All access codes are derived from the lower address part (LAP) of a device address or an inquiry address. The device access code is used during **page**, **page scan** and **page response** sub states and shall be derived from the paged device's BD_ADDR. The channel access code is used in the **CONNECTION** state and forms the beginning of all packets exchanged on the piconet physical channel. The channel access code shall be derived from the LAP of the master's BD_ADDR. Finally, the inquiry access code shall be used in the **inquiry** sub state. There is one general IAC (GIAC) for general inquiry operations and there are 63 dedicated IACs (DIACs) for dedicated inquiry operations.

The payload ranges from zero to a maximum of 2745 bits. Different packet types have been defined. Packet may consist of:

- the shortened access code only
- the access code and the packet header
- the access code, the packet header and the payload.

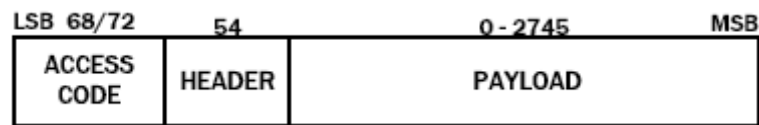


Figure 4.3 Bluetooth Packet Structure.

The header contains link control (LC) information and consists of 6 fields. They are LT_ADDR (3-bit logical transport address), TYPE (4-bit type code specifies which packet type is used and determines how many slots the current packet will occupy), FLOW (1-bit flow control), ARQN (1-bit acknowledge indication), SEQN (1-bit sequence number) and HEC (8-bit header error check). The total header, including the HEC, consists of 18 bits and is encoded with a rate 1/3 FEC resulting in a 54-bit header.

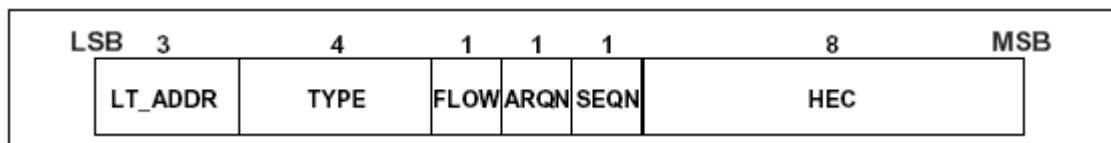


Figure 4.4 Packet Header format.

4.2.2.6 Packet Types

Packet type defines which type of traffic is carried out by this packet (SCO, ACL, NULL, POLL), the type of error correction used for the payload and how many slots the payload will last for. The packets used on the piconet are related to the logical transports they are used in. Three logical transports with distinct packet types are defined

- the SCO logical transport,
- the eSCO logical transport and
- the ACL logical transport.

For each of these logical transports, 15 different packet types can be defined. There are five common kinds of packets [1], [3] viz. NULL, POLL, FHS, DM1 and ID packet. All are common packet types but ID packet does not have a packet header.

ID packet: The identity or ID packet consists of the device access code (**DAC**) or inquiry access code (**IAC**). It has a fixed length of 68 bits. It is a very robust packet since the receiver uses a bit correlator to match the received packet to the known bit sequence of the ID packet.

NULL packet: The NULL packet has no payload and consists of the channel access code and packet header only. Its total (fixed) length is 126 bits. The NULL packet may be used to return link information to the source regarding the success of the previous transmission (ARQN), or the status of the RX buffer (FLOW). The NULL packet may not have to be acknowledged.

POLL packet: The POLL packet is very similar to the NULL packet. It does not have a payload. In contrast to the NULL packet, it requires a confirmation from the recipient. It is not a part of the ARQ scheme. The POLL packet does not affect the ARQN and SEQN fields. Upon reception of a POLL packet the slave shall respond with a packet even when the slave does not have any information to send unless the slave has scatternet commitments in that timeslot. This return packet is an implicit acknowledgement of the POLL packet. This packet can be used by the master in a piconet to poll the slaves. Slaves shall not transmit the POLL packet.

FHS packet: The FHS packet is a special control packet containing, among other things, the Bluetooth device address and the clock of the sender. The payload contains 144 information bits plus a 16-bit CRC code. The payload is coded with a rate 2/3 FEC with a gross payload length of 240 bits. The payload consists of eleven fields. The FHS packet is used in page master response, inquiry response and in role switch. The FHS packet contains real-time clock information. This clock information shall be updated before each retransmission. The retransmission of the FHS payload is different than retransmissions of ordinary data payloads where the same payload is used for each retransmission. The FHS packet is used for frequency hop synchronization before the piconet channel has been established, or when an existing piconet changes to a new piconet.

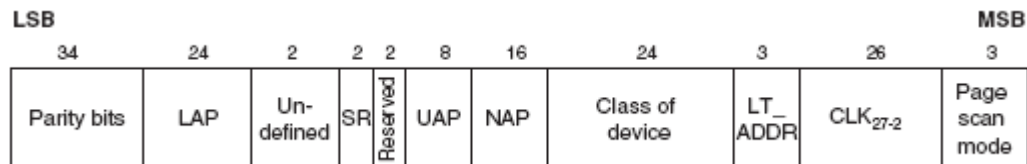


Figure 4.5 Format of the FHS payload

Parity bits	This 34-bit field contains the parity bits that form the first part of the sync word of the access code of the device that sends the FHS packet. These bits are derived from the LAP.
LAP	This 24-bit field shall contain the lower address part of the device that sends the FHS packet.
Undefined	This 2-bit field is reserved for future use and shall be set to zero.
SR	This 2-bit field is the scan repetition field and indicates the interval between two consecutive page scan windows.
Reserved	This 2-bit field shall be set to 10.
UAP	This 8-bit field shall contain the upper address part of the device that sends the FHS packet.
NAP	This 16-bit field shall contain the non-significant address part of the device that sends the FHS packet (see also section 4.2.2.1 for LAP, UAP, and NAP).
Class of device	This 24-bit field shall contain the class of device of the device that sends the FHS packet.
LT_ADDR	This 3-bit field shall contain the logical transport address the recipient shall use if the FHS packet is used at connection setup or role switch. A slave responding to a master or a device responding to an inquiry request message shall include an all-zero LT_ADDR field if it sends the FHS packet.
CLK₂₇₋₂	This 26-bit field shall contain the value of the native clock of the device that sends the FHS packet, sampled at the beginning of the transmission of the access code of this FHS packet. This clock value has a resolution of 1.25ms (two-slot interval). For each new transmission, this field is updated so that it accurately reflects the real-time clock value.
Page scan mode	This 3-bit field shall indicate which scan mode is used by default by the sender of the FHS packet.

Table3 Description of the FHS payload

DM1 packet: The DM1 packet carries data information only. The payload is between 1 and 18 information bytes (including the 1-byte payload header) plus a 16-bit CRC code. The DM1 packet occupies a single time slot. The information plus CRC bits are coded with a rate 2/3 FEC. The payload header in the DM1 packet is 1 byte long. The length indicator in the payload header specifies the number of user bytes (excluding payload header and the CRC code).

4.2.2.7 Bitstream Processing

Bluetooth units often have to contend with electro-magnetically noisy environments. Thus, there is a need for some kind of error-detection and correction. For error-detection, Bluetooth uses various checksum-calculations. When errors are detected, there are 3 error-correction schemes defined for Bluetooth. They are

1. 1/3 rate FEC (Forward Error Correction)
2. 2/3 rate FEC
3. ARQ unnumbered scheme (Automatic Repeat Request).

The purpose of the FEC scheme on the data payload is to reduce the number of re-transmissions. However, in a reasonably error-free environment, FEC gives unnecessary overhead that reduces the throughput. Therefore, the packet definitions have been kept flexible to use FEC in the payload or not, resulting in

- the **DM** and **DH** packets for the **ACL** link, and
- the HV packets for the **SCO** link.

The packet header is always protected by a 1/3 rate FEC; it contains valuable link information and should be able to sustain more bit errors.

4.2.3 Link Controller

The link controller is responsible for encoding and decoding of Bluetooth packets from the data payload [5] and parameters related to the physical channel, logical transport and logical link. The link controller carries out the link control protocol signalling (in close conjunction with the scheduling function of the resource manager), which is used to communicate flow control and acknowledgement and retransmission request signals. The interpretation of these signals is a characteristic of the logical transport associated with the baseband packet. Interpretation and control of the link control signalling is normally associated with the resource manager's scheduler. Different Link Controller states are Standby, Inquiry, Inquiry Scan, Page, Page Scan, Connection- Active, Connection- Hold, Connection- Sniff and Connection- Park. The different LC operations are

- Device Discovery and Inquiry
- Connection Establishment and Paging

- Optional Paging Scheme

4.2.4 Link Manager

The link manager is responsible for the creation, modification and release of logical links (and, if required, their associated logical transports), as well as the update of parameters related to physical links between devices. The link manager communicates with the link managers on other Bluetooth devices using the Link Management Protocol (LMP). The link manager is responsible for link set-up and control using the Link Management Protocol (LMP). The link manager assumes that the link control provides a guaranteed delivery mechanism for the LMP messages. It provides the protocol support for a number of procedures, including: authentication, encryption control link physical parameter control, e.g. power control, timing accuracy master-to-slave switching (and vice versa) [6].

Link manager communicates with its peers on other devices using the Link Management Protocol (LMP). Every LMP message begins with a flag bit which is 0 if a master initiated the transaction and 1 if the slave initiated the transaction. That bit is followed by a 7-bit Operation Code, and by the message's parameters.

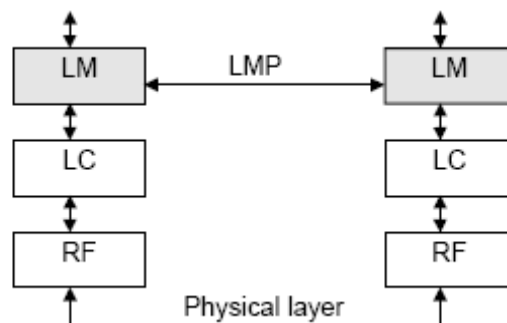


Figure 4.6 Link Manager Protocol signalling layer

When a link is first set up, it uses single-slot packets by default. Multi-slot packets make more efficient use of the band, but there are some occasions when they can't be used, for example on noisy links or if SCO links don't leave sufficient space between their slots for multi-slot packets.

LMP also provides a mechanism for negotiating encryption modes and coordinating encryption keys used by devices on both ends of the link. In addition, LMP supports messages for configuration of the quality of service on a connection. Packet types can automatically change according to the channel quality, so that the data can be transferred at a higher rate when the channel quality is good and on lower rates with more error protection if the channel quality deteriorates.

4.2.5 Host Controller Interface

All interaction between the host PC and the Bluetooth device occurs via the HCI Driver. This is because it is the abstraction layer responsible for communicating with a Bluetooth device across a variety of transport layers such as USB, UART and PCMCIA. The HCI supports ACI, SCO and HCI command and event logical channels.

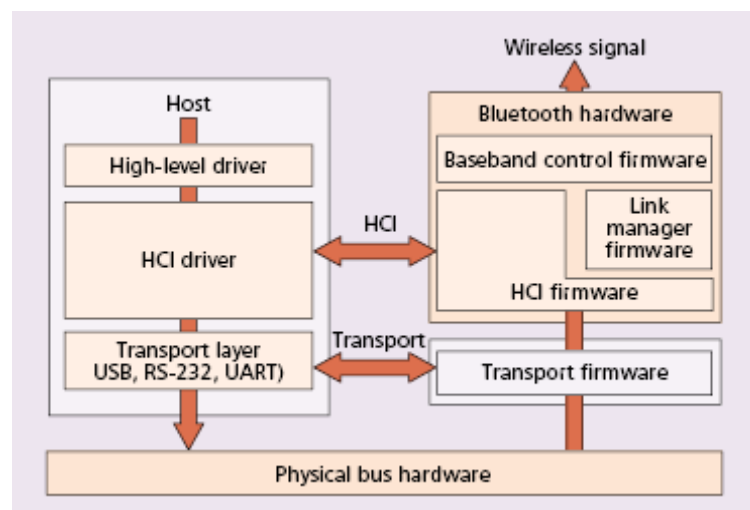


Figure 4.7 The HCI Layer

4.3 The Bluetooth Host- Upper Protocols of the Transport Group

The upper layers of the stack are usually implemented in software.

4.3.1 The L2CAP Layer

The Logical Link Control and Adaptation Protocol (L2CAP) layer is an abstraction layer that hides the complexities of the underlying transport protocol. The purpose of L2CAP is to provide connection-oriented and connectionless data services to higher layer protocols. To achieve this L2CAP provides the following functionality [6]:

- multiplexing of higher layer protocols
- establishment, maintenance and clearing of logical connections for connection-oriented services
- segmentation and re-assembly services to allow packets of **up** to 64 kilobytes to be transported between L2CAP entities.
- Additional features include support for groups

allowing existing protocols such as TCP/IP to run unmodified over Bluetooth.

L2CAP resource manager: The L2CAP resource manager block is responsible for managing the ordering of submission of PDU fragments to the baseband and some relative scheduling between channels to ensure that L2CAP channels with QoS commitments are not denied access to the physical channel due to Bluetooth controller resource exhaustion. This is required because the architectural model does not assume that the Bluetooth controller has limitless buffering, or that the HCI is a pipe of infinite bandwidth. L2CAP Resource Managers may also carry out traffic conformance policing to ensure that applications are submitting L2CAP SDUs within the bounds of their negotiated QoS settings. The general Bluetooth data transport model assumes well-behaved applications, and does not define how an implementation is expected to deal with this problem.

The channel manager is responsible for creating, managing and destroying L2CAP channels for the transport of service protocols and application data streams. The channel manager uses the L2CAP protocol to interact with a channel manager on a remote (peer) device to create these L2CAP channels and connect their endpoints to the appropriate entities. The channel manager interacts with its local link manager to create new logical links (if necessary) and to configure these links to provide the required quality of service for the type of data being transported.

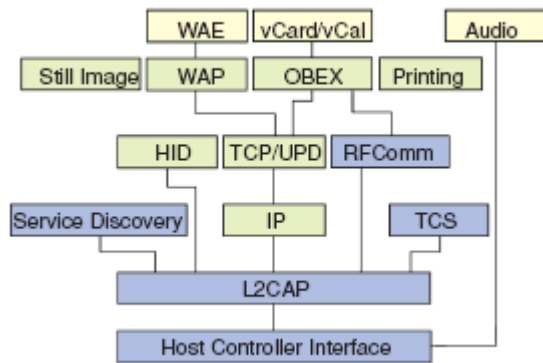


Figure 4.8 Interoperability with existing protocols & applications

4.3.2 The RFCOMM Middleware Protocols

The RFCOMM layer is a serial port (RS-232 serial ports) emulation protocol that is intended primarily to emulate serial ports over the L2CAP protocol. It is based on the E131 GSM mobile telephone specification IS 07.105. The concept is basically to allow a laptop, or other computing device, to connect to, say, a GSM phone, which is used as a radio modem for remotely accessing data services via the GSM network.

RFCOMM provides the same major attributes of TCP. The biggest difference between TCP and RFCOMM from a network programmer's perspective is the choice of port number. Whereas TCP supports up to 65535 open ports on a single machine, RFCOMM only allows for 30. This has a significant impact on how to choose port numbers for server applications, and is discussed shortly.

4.3.3 SDP Middleware Protocols

The Service Discovery Protocol (SDP) allows Bluetooth devices to discover what services are available on a device. It has a client-server architecture that uses the Service Discovery Database at the server. The Service Discovery Database (or SDP server) contains a number of Service Records, and each Service Record contains attributes of the service. One of the attributes is the Service Record Handle, which uniquely identifies each service record within the SDP server. The SDP supports both searching for services and browsing. Searching allows a client to search for a specific service, which is subsequently identified by the Service Record Handle attribute. Browsing allows a client to discover what services are supported; these are identified

by the service attributes, including the Service Record Handle. Once a service has been identified, its attributes can be requested using the Service Record Handle. The attributes include information on how to connect to the service via the protocol stack, e.g. the RFCOMM server channel number with which the service is registered. The Service Discovery Protocol is run over L2CAP.

4.4 Conclusion

The general overview of the upper and lower protocols of the transport group was discussed here. With the knowledge of the Bluetooth Module and Bluetooth Host we can implement any given task related to Bluetooth with ease. From this chapter it is clear that what the exact purpose of each layer and protocol.

CHAPTER V

MAKING BLUETOOTH ENABLED PERSONAL COMPUTER

5.1 Introduction

BlueZ is a powerful Bluetooth communications stack with extensive APIs that allows a user to fully exploit all local Bluetooth resources, but it has no official documentation. Furthermore, there is very little unofficial documentation as well. Novice developers are told to figure out the API by reading through the BlueZ source code. This is a time consuming process which reveals small pieces of information at a time, and is quite often enough of an obstacle to deter many potential developers. This chapter presents a short introduction to developing Bluetooth applications in C with BlueZ.

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 5.1):
 - HCI Core
 - HCI UART, USB and Virtual HCI device drivers
 - L2CAP protocol module
 - Configuration and testing utilities

5.2 Setting up BlueZ

5.2.1 Obtaining BlueZ

BlueZ [20] can be directly downloaded. There is also an up to date CVS tree available there.

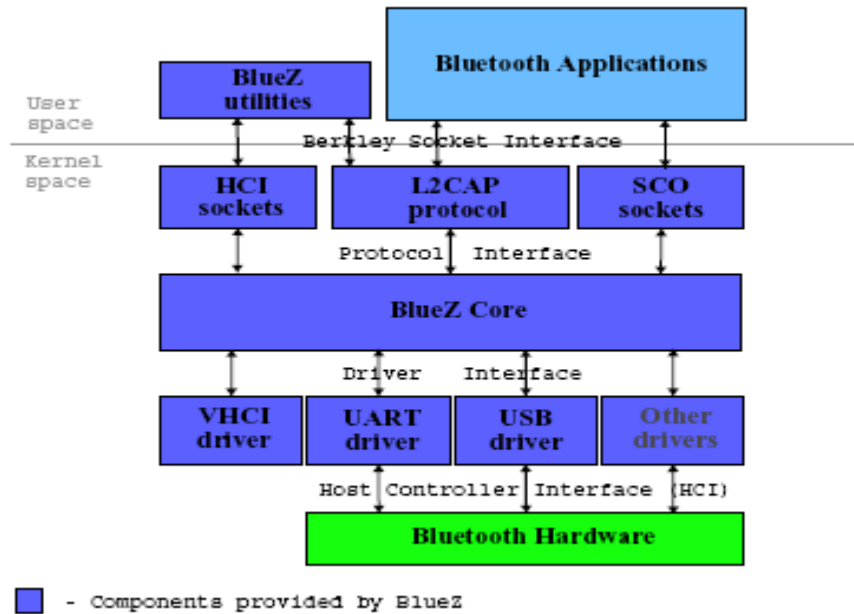


Figure 5.1 BlueZ Overview Diagram

5.2.2 Requirements

In order to use BlueZ, atleast a 2.4.4 Linux kernel is required. The 2.4.6 kernel has BlueZ built-in. In case, if the latest version of BlueZ is needed, disable native BlueZ support on the present PC. 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 Bluetooth applications. This is very useful if we do not have any real Bluetooth devices.

5.2.3 Compilation and Installation

To configure BlueZ for the kernel run

```
./configure
```

The `configure` command automatically searches for all the required components and packages in the directory. Optionally, the `configure` support the following options:

`--enable-debug` enables BlueZ debugging

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

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

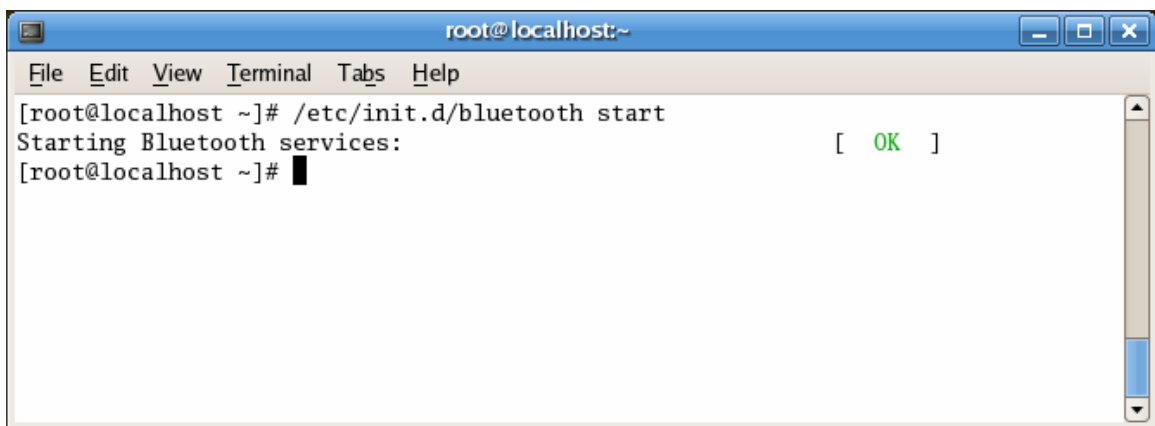
make install

Installation of BlueZ is complete! Now, 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.

To update Linux kernel tree with the up to date CVS version run

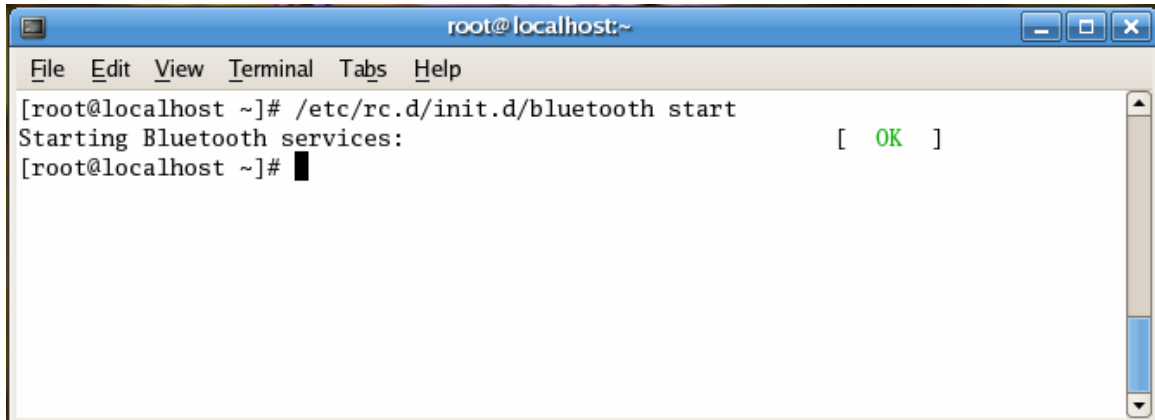
make update

and recompile your kernel.



```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# /etc/init.d/bluetooth start  
Starting Bluetooth services: [ OK ]  
[root@localhost ~]# █
```

Figure 5.2a Starting Bluetooth Services



```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# /etc/rc.d/init.d/bluetooth start  
Starting Bluetooth services: [ OK ]  
[root@localhost ~]# █
```

Figure 5.2b Starting Bluetooth Services

5.2.4 Loading BlueZ Modules

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

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

To use UART based Bluetooth devices, add the following line to the file /etc/modules.conf in addition to the above

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

To use the Virtual HCI device, add the following line to the file /etc/modules.conf

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

After making any of the above changes, you can run the below command to enable auto-loading of BlueZ modules.

```
depmod -a
```

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
```

To see the BlueZ modules, run

```
lsmod
```

If there are any errors, check your /var/log/messages file.

5.2.5 Device Initialization

UART Devices: Make sure that /etc/hcid.conf is correct (tty, speed, flow, etc). Start hcid. To configure the UART devices, use the tool hciattach. It can be called either manually or from the PCMCIA cardmgr scripts.

USB Device: Be sure to have USB support properly installed on the Desktop PC that is being used. Plug in the 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 daemon (UART). When configured correctly they should be brought up automatically. Check your kernel and system logs for error messages.

5.2.6 Debugging the BlueZ Driver

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

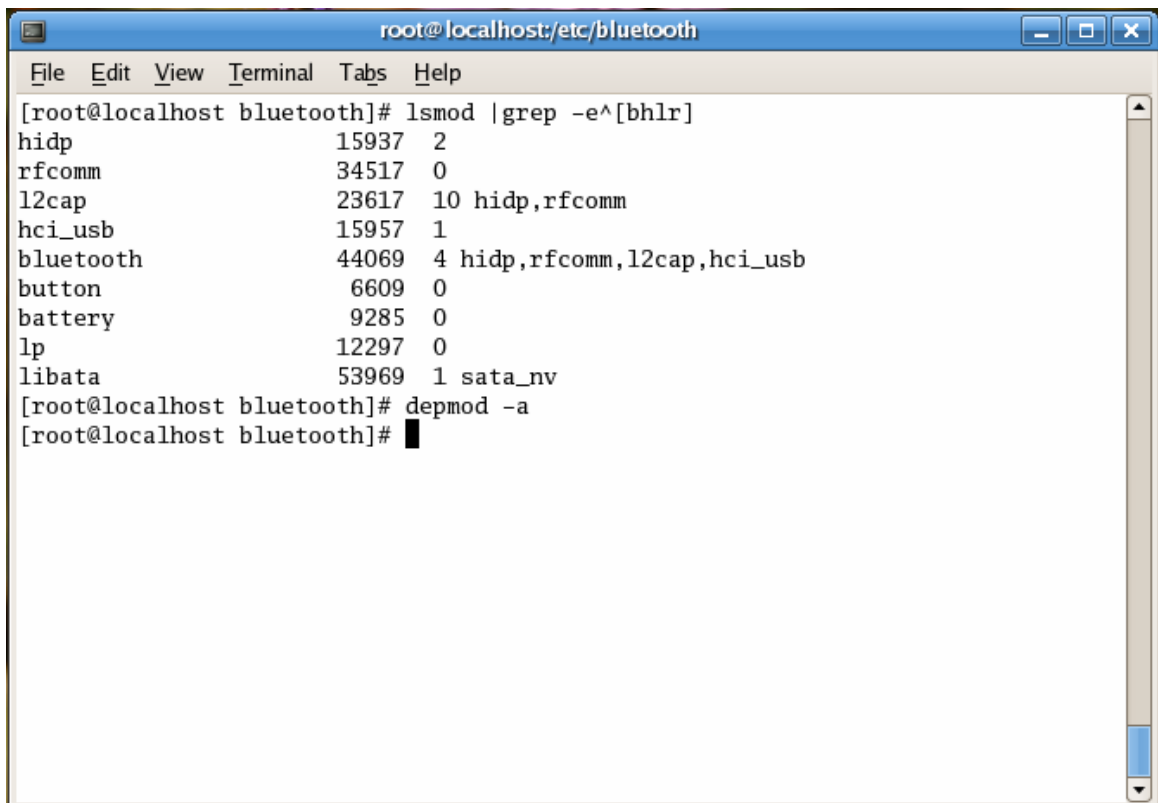
- check the system log in /var/log/messages
- check the debug output from the BlueZ driver
- check for dead processes, like hcid
- check whether loading the right modules compiled for current kernel from the right location.

Also try:

cvns 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 up-to-date
make	
make install	install the newly compiled modules and tools

If it still hangs

- 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



```
root@localhost:/etc/bluetooth
File Edit View Terminal Tabs Help
[root@localhost bluetooth]# lsmod | grep -e^[bhlr]
hidp                15937  2
rfcomm              34517  0
l2cap               23617  10 hidp,rfcomm
hci_usb            15957  1
bluetooth          44069  4 hidp,rfcomm,l2cap,hci_usb
button              6609  0
battery             9285  0
lp                 12297  0
libata              53969  1 sata_nv
[root@localhost bluetooth]# depmod -a
[root@localhost bluetooth]#
```

Figure 5.3 List of modules starting with b, h, l and r

5.3 Conclusion

After the successful completion of the compilation and installation of BlueZ the computer we had now is Bluetooth enabled. To increase the services provided by the PC we can bring various software's and install them. Now a Bluetooth USB Dongle is used as HCI layer which should be inserted in the USB port present in the CPU.

CHAPTER VI

ESTABLISHING A LINK BETWEEN PC AND MOBILE

6.1 Introduction

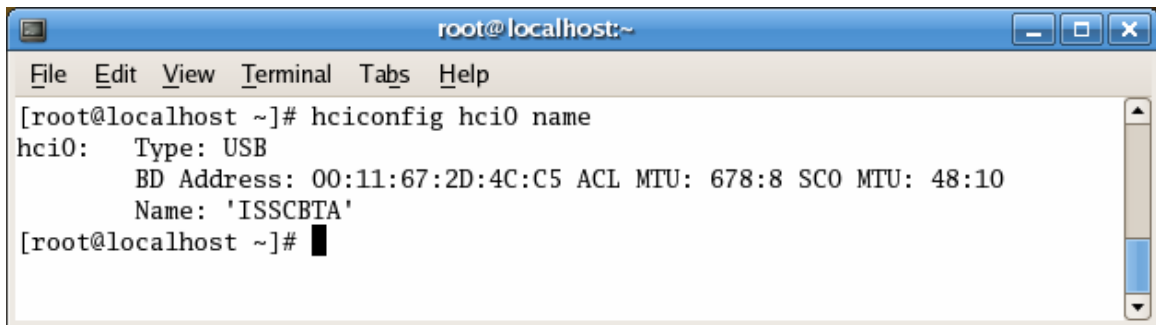
Although Bluetooth was designed from the ground up, independently of the Ethernet and TCP/IP protocols, it is quite reasonable to think of Bluetooth programming in the same way as Internet programming. Fundamentally, they have the same principles of one device communicating and exchanging data with another device. The different parts of network programming can be separated into several components

- Choosing a device with which to communicate
- Figuring out how to communicate with it
- Making an outgoing connection
- Accepting an incoming connection
- Sending data
- Receiving data

Some of these components do not apply to all models of network programming. In a connectionless model, for example, there is no notion of establishing a connection. Some parts can be trivial in certain scenarios and quite complex in another. If the numerical IP address of a server is hard-coded into a client program, for example, then choosing a device is no choice at all. In other cases, the program may need to consult numerous lookup tables and perform several queries before it knows its final communication endpoint.

6.2 Dongles

A Bluetooth dongle can be loosely defined as a small device that is externally attached to a system in order to provide Bluetooth functionality. Dongles consist of the same type of ASSP as a development kit although they do not necessarily implement any of the optional parts of the Bluetooth specification. Usually they provide only one HCI transport layer option and come bundled with a software stack and some utility programs.



```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hciconfig hci0 name  
hci0:   Type: USB  
        BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10  
        Name: 'ISSCBTA'  
[root@localhost ~]#
```

Figure 6.1 Bluetooth USB Dongle Name

6.3 Transport protocol

Once the client application has determined the address of the host machine it wants to connect to, it must determine which transport protocol to use. This section describes the Bluetooth transport protocols closest in nature to the most commonly used Internet protocols. Both Bluetooth and Internet programming involve using numerous different transport protocols, some of which are stacked on top of others. In TCP/IP, many applications use either TCP or UDP, both of which rely on IP as an underlying transport. TCP provides a connection-oriented method of reliably sending data in streams, and UDP provides a thin wrapper around IP that unreliably sends individual datagram's of fixed maximum length. There are also protocols like RTP for applications such as voice and video communications that have strict timing and latency requirements. While Bluetooth does not have exactly equivalent protocols, it does provide protocols which can often be used in the same contexts as some of the Internet protocols.

RFCOMM + TCP

The RFCOMM protocol provides roughly the same service and reliability guarantees as TCP. Although the specification explicitly states that it was designed to emulate RS-232 serial ports (to make it easier for manufacturers to add Bluetooth capabilities to their existing serial port devices), it is quite simple to use it in many of the same scenarios as TCP. In general, applications that use TCP are concerned with having a point-to-point connection over which they can reliably exchange streams of data. If a portion of that data cannot be delivered within a fixed time limit, then the connection is terminated and an error is delivered. The biggest difference between TCP and RFCOMM from a network programmer's perspective is the choice of port

number. Whereas TCP supports up to 65535 open ports on a single machine, RFCOMM only allows for 30.

L2CAP + UDP

UDP is often used in situations where reliable delivery of every packet is not crucial, and sometimes to avoid the additional overhead incurred by TCP. Specifically, UDP is chosen for its best-effort, simple datagram semantics. These are the same criteria that L2CAP satisfies as a communications protocol.

L2CAP, by default, provides a connection-oriented protocol that reliably sends individual datagram's of fixed maximum length. Being fairly customizable, L2CAP can be configured for varying levels of reliability. To provide this service, the transport protocol that L2CAP is built on employs a transmit/acknowledgement scheme, where unacknowledged packets are retransmitted. There are three policies an application can use

- never retransmit
- retransmit until total connection failure (the default)
- drop a packet and move on to queued data if a packet hasn't been acknowledged after a specified time limit (0-1279 milliseconds). This is useful when data must be transmitted in a timely manner.

Although Bluetooth does allow the application to use best-effort communication instead of reliable communication, several caveats are in order. The reason for this is that adjusting the delivery semantics for a single L2CAP connection to another device affects all L2CAP connections to that device. If a program adjusts the delivery semantics for an L2CAP connection to another device, it should take care to ensure that there are no other L2CAP connections to that device. Additionally, since RFCOMM uses L2CAP as a transport, all RFCOMM connections to that device are also affected. While this is not a problem if only one Bluetooth connection to that device is expected, it is possible to adversely affect other Bluetooth applications that also have open connections.

The limitations on relaxing the delivery semantics for L2CAP aside, it serves as a suitable transport protocol when the application doesn't need the overhead and streams-based nature of RFCOMM, and can be used in many of the same situations

that UDP is used in. Given this suite of protocols and different ways of having one device communicate with another, an application developer is faced with the choice of choosing which one to use. In doing so, we will typically consider the delivery reliability required and the manner in which the data is to be sent. As shown above we will usually choose RFCOMM in situations where we would choose TCP and L2CAP when we would choose UDP.

Requirement	Internet	Bluetooth
Reliable, streams-based	TCP	RFCOMM
Reliable, datagram	TCP	RFCOMM or L2CAP with infinite retransmit
Best-effort, datagram	UDP	L2CAP (0-1279 ms retransmit)

Table 4 Comparison of the protocols.

6.4 Port numbers and the Service Discovery Protocol

Once a numerical address and transport protocol are known, choose the port number to communicate with a remote machine. Almost all Internet transport protocols in common usage are designed with the notion of port numbers, so that multiple applications on the same host may simultaneously utilize a transport protocol. Bluetooth is no exception, but uses slightly different terminology. In L2CAP, ports are called Protocol Service Multiplexers (PSM), and can take on odd-numbered values between 1 and 32767. In RFCOMM, channels 1-30 are available for use. These differences aside, both protocol service multiplexers and channels serve the same purpose that ports do in TCP/IP. L2CAP, unlike RFCOMM, has a range of reserved port numbers (1-1023) that are not to be used for custom applications and protocols. This information is summarized in Table 5. The word port is used in place of PSM and channel for clarity.

Protocol	Terminology	Reserved/well-known ports	Dynamically assigned ports
TCP	port	1-1024	1025-65535
UDP	port	1-1024	1025-65535
RFCOMM	channel	none	1-30
L2CAP	PSM	odd numbered 1-4095	odd numbered 4097 -32765

Table 5 Port numbers and their terminology for various protocols

In Internet programming, server applications traditionally make use of well known port numbers that are chosen and agreed upon at design time. Client applications will use the same well known port number to connect to a server. The main disadvantage to this approach is that it is not possible to run two server applications which both use the same port number. Due to the relative large number port numbers to choose from, this has not yet become a serious issue. The Bluetooth transport protocols, however, were designed with few port numbers, which means we cannot choose an arbitrary port number at design time. Although this problem is not as significant for L2CAP, which has around 15,000 unreserved port numbers, RFCOMM has only 30 different port numbers. A consequence of this is that there is a greater than 50% chance of port number collision with just 7 server applications. In this case, the application designer clearly should not arbitrarily choose port numbers. The Bluetooth answer to this problem is the Service Discovery Protocol (SDP). Instead of agreeing upon a port to use at application design time, the Bluetooth approach is to assign ports at runtime and follow a publish-subscribe model. The host machine operates a server application, called the SDP server that uses one of the few L2CAP reserved port numbers. Other server applications are dynamically assigned port numbers at runtime and register a description of themselves and the services they provide (along with the port numbers they are assigned) with the SDP server. Client applications will then query the SDP server (using the well defined port number) on a particular machine to obtain the information they need.

This raises the question of how do clients know which description is the one they are looking for. The standard way of doing this in Bluetooth is to assign a 128-bit

number, called the Universally Unique Identifier (UUID) [1], at design time. Following a standard method of choosing this number guarantees choosing a UUID unique from those chosen by anyone else following the same method. Thus, a client and server application both designed with the same UUID can provide this number to the SDP server as a search term.

As with RFCOMM and L2CAP, only a small portion of SDP has been described here - those parts most relevant to a network programmer. Among the other ways SDP can be used are to describe which transport protocols a server is using, to give information such as a human-readable description of the service provided and who is providing it, and to search on fields other than the UUID such as the service name. Another point worth mentioning is that SDP is not even required to create a Bluetooth application. It is perfectly possible to revert to the TCP/IP way of assigning port numbers at design time and hoping to avoid port conflicts, and this might often be done to save some time. In controlled settings such as the computer science laboratory, this is quite reasonable. Ultimately, however, to create a portable application that will run in the greatest number of scenarios, the application should use dynamically assigned ports and SDP.

The various services offered by my Bluetooth enabled PC are

```
Inquiring ...
Browsing 00:0E:6D:DE:F2:42 ...
Service Name: Fax
Service RecHandle: 0x10000
Service Class ID List:
  "Fax" (0x1111)
  "Generic Telephony" (0x1204)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
  Channel: 1
Language Base Attr List:
  code_ISO639: 0x656e
  encoding: 0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Fax" (0x1111)
  Version: 0x0100

Service Name: Dial-up Networking
Service RecHandle: 0x10001
```

Service Class ID List:
"Dialup Networking" (0x1103)
"Generic Networking" (0x1201)
Protocol Descriptor List:
"L2CAP" (0x0100)
"RFCOMM" (0x0003)
Channel: 1
Language Base Attr List:
code_ISO639: 0x656e
encoding: 0x6a
base_offset: 0x100
Profile Descriptor List:
"Dialup Networking" (0x1103)
Version: 0x0100

Service Name: Bluetooth Serial Port
Service Description: Bluetooth Serial Port
Service Provider: Symbian Ltd.
Service RecHandle: 0x10002
Service Class ID List:
"Serial Port" (0x1101)
Protocol Descriptor List:
"L2CAP" (0x0100)
"RFCOMM" (0x0003)
Channel: 2
Language Base Attr List:
code_ISO639: 0x656e
encoding: 0x6a
base_offset: 0x100

Service Name: OBEX Object Push
Service RecHandle: 0x10003
Service Class ID List:
"OBEX Object Push" (0x1105)
Protocol Descriptor List:
"L2CAP" (0x0100)
"RFCOMM" (0x0003)
Channel: 9
"OBEX" (0x0008)
Language Base Attr List:
code_ISO639: 0x656e
encoding: 0x6a
base_offset: 0x100
Profile Descriptor List:
"OBEX Object Push" (0x1105)
Version: 0x0100

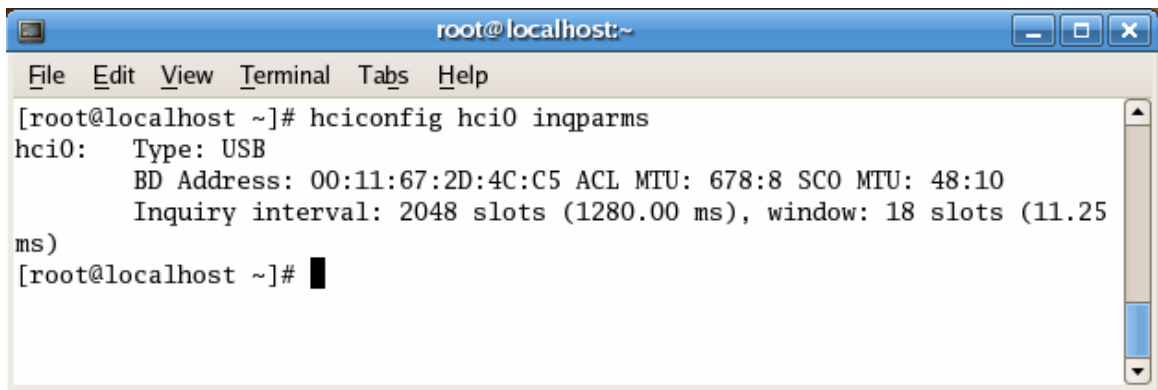
Service Name: OBEX File Transfer
Service RecHandle: 0x10004
Service Class ID List:

"OBEX File Transfer" (0x1106)
Protocol Descriptor List:
 "L2CAP" (0x0100)
 "RFCOMM" (0x0003)
 Channel: 10
 "OBEX" (0x0008)
Language Base Attr List:
 code_ISO639: 0x656e
 encoding: 0x6a
 base_offset: 0x100
Profile Descriptor List:
 "OBEX File Transfer" (0x1106)
 Version: 0x0100

Service Name: Handsfree Audio Gateway
Service RecHandle: 0x10005
Service Class ID List:
 "Handfree Audio Gateway" (0x111f)
 "Generic Audio" (0x1203)
Protocol Descriptor List:
 "L2CAP" (0x0100)
 "RFCOMM" (0x0003)
 Channel: 3
Language Base Attr List:
 code_ISO639: 0x656e
 encoding: 0x6a
 base_offset: 0x100
Profile Descriptor List:
 "Handsfree" (0x111e)
 Version: 0x0101

6.5 Establishing network connections

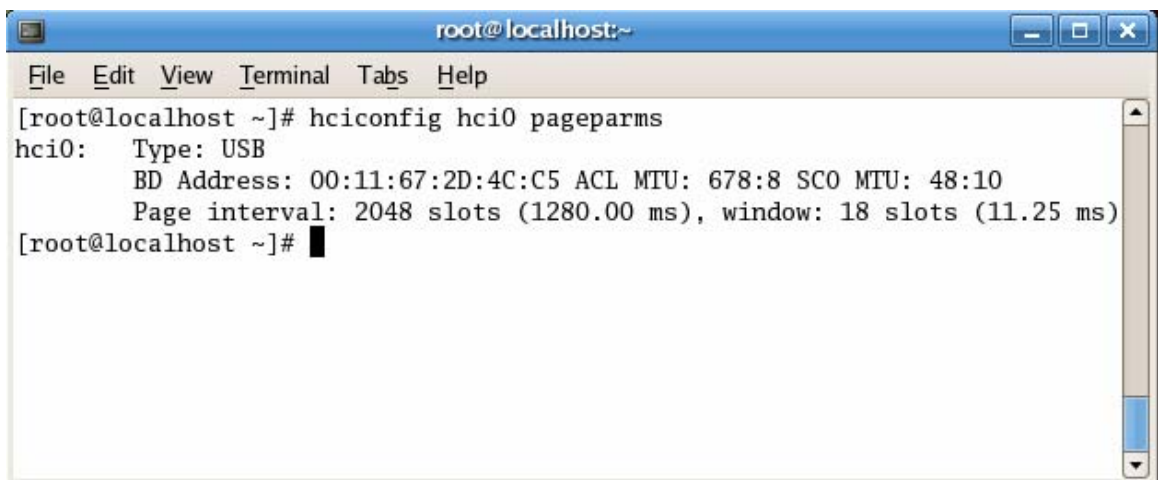
In order to establish new connections the procedures **inquiry** and **paging** are used. The inquiry procedure enables a unit to discover which units are in range, and what their device addresses and clocks are. With the paging procedure, an actual connection can be established. Only the Bluetooth device address is required to set up a connection. Knowledge about the clock will accelerate the setup procedure. A unit that establishes a connection will carry out a page procedure and will automatically become the master of the connection.



```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hciconfig hci0 inqparms  
hci0:  Type: USB  
       BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10  
       Inquiry interval: 2048 slots (1280.00 ms), window: 18 slots (11.25  
ms)  
[root@localhost ~]#
```

Figure 6.2 Inquiring

For the paging process, several paging schemes can be applied. There is one mandatory paging scheme which has to be supported by each Bluetooth device. This mandatory scheme is used when units meet for the first time, and in case the paging process directly follows the inquiry process. Two units, once connected using a mandatory paging/scanning scheme, may agree on an optional paging/scanning scheme.



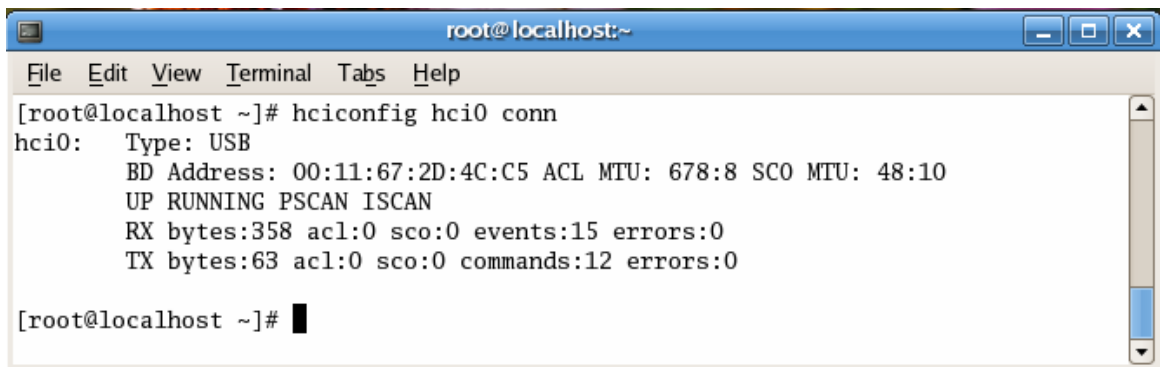
```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hciconfig hci0 pageparms  
hci0:  Type: USB  
       BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10  
       Page interval: 2048 slots (1280.00 ms), window: 18 slots (11.25 ms)  
[root@localhost ~]#
```

Figure 6.3 Paging

The Connection Establishment

After the paging procedure, the master must poll the slave by sending POLL or NULL packets, to which the slave responds. LMP procedures that do not require any interactions between the LM and the host at the paged unit's side can then be carried out. When the paging device wishes to create a connection involving layers above LM, it sends LMP_host_connection_req. When the other side receives this

message, the host is informed about the incoming connection. The remote device can accept or reject the connection request by sending LMP_accepted or LMP_not_accepted.

A terminal window titled 'root@localhost:~' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal content shows the command '[root@localhost ~]# hciconfig hci0 conn' and its output: 'hci0: Type: USB', 'BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10', 'UP RUNNING PSCAN ISCAN', 'RX bytes:358 acl:0 sco:0 events:15 errors:0', and 'TX bytes:63 acl:0 sco:0 commands:12 errors:0'. The prompt '[root@localhost ~]#' is followed by a cursor.

```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hciconfig hci0 conn  
hci0:  Type: USB  
      BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10  
      UP RUNNING PSCAN ISCAN  
      RX bytes:358 acl:0 sco:0 events:15 errors:0  
      TX bytes:63 acl:0 sco:0 commands:12 errors:0  
[root@localhost ~]# █
```

Figure 6.4 Shows the Connection

When a device does not require any further link set-up procedures, it will send LMP_setup_complete. The device will still respond to requests from the other device. When the other device is also ready with link set-up, it will send LMP_setup_complete. After this, the first packet on a logical channel different from LMP can then be transmitted.

6.6 Conclusion

In this chapter we saw the similarities between the various transport layers of TCP/IP (Transport Control Protocol/Internet Protocol) and Bluetooth Transport protocols. Also we established link between Bluetooth enabled Desktop PC and Mobile.

CHAPTER VII

EXPERIMENTATION RESULTS AND DISCUSSION

Before establishing a link make sure that Desktop PC is Bluetooth enabled and the Mobile phone used is in discoverable mode (show to all).

The principle remains the same, however, in that the unique identifying address of the target device must be known to communicate with it. In both cases, the client program will often not have advance knowledge of these target addresses. In Internet programming, the user will typically supply a host name, such as dce.edu or dceonline.com, which the client must translate to a physical IP address using the Domain Name System (DNS). In Bluetooth, the user will typically supply some user-friendly name, such as “My Phone” (here “PRAVEEN DARSHANAM”), and the client translates this to a numerical address by searching nearby Bluetooth devices.

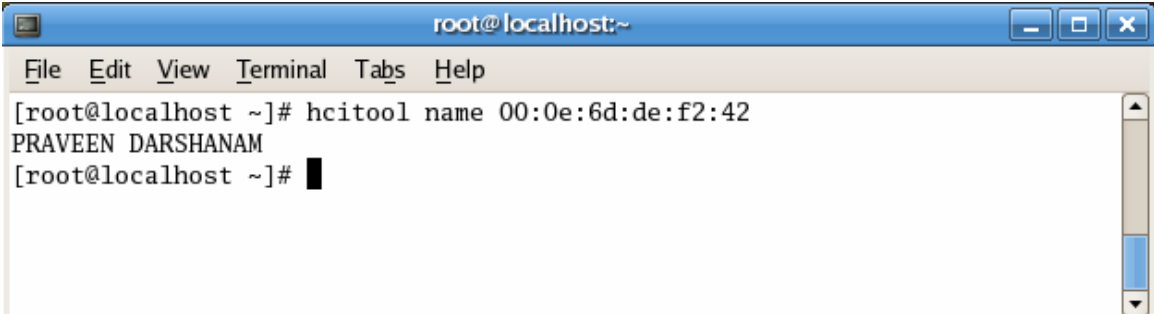
hciconfig - HCI device configuration utility

hciconfig hciX	[up	Open and initialize HCI device
	down	Close HCI device
	reset	Reset HCI device
	rstat	Reset stat counters
	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

Device Name

Since humans do not deal well with 48-bit numbers like 0x000EED3D1829 (in much the same way we do not deal well with numerical IP addresses like 192.68.161.104), Bluetooth devices will almost always have a user-friendly name. This name is usually shown to the user in lieu of the Bluetooth address to identify a device, but ultimately it is the Bluetooth address that is used in actual communication. For many machines, such as cell phones and desktop computers, this name is configurable and the user can choose an arbitrary word or phrase. There is no requirement for the user to choose a unique name, which can sometimes cause confusion when many nearby devices have the same name. When sending a file to someone's phone, for example, the user may be faced with the task of choosing from 5 different phones, each of which is named "My Phone". Although names in Bluetooth differ from Internet names in that there is no central naming authority and names can sometimes be the same, the client program still has to translate from the user-friendly names presented by the user to the underlying numerical addresses. In TCP/IP, this involves contacting a local name server, issuing a query, and waiting for a result. In Bluetooth, where there are no name servers, a client will instead broadcast inquiries to see what other devices are nearby and query each detected device for its user-friendly name. The client then chooses whichever device has a name that matches the one supplied by the user.



```

root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# hcitool name 00:0e:6d:de:f2:42
PRAVEEN DARSHANAM
[root@localhost ~]#

```

Figure 7.1 Name of the Bluetooth Device (Mobile)

```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hcitool info 00:0e:6d:de:f2:42  
Requesting information ...  
    BD Address: 00:0e:6d:de:f2:42  
    Device Name: PRAVEEN DARSHANAM  
    LMP Version: 1.1 (0x1) LMP Subversion: 0x248  
    Manufacturer: Nokia Mobile Phones (1)  
    Features: 0xbf 0x28 0x21 0x00 0xc0 0x00 0x00 0x00  
              <3-slot packets> <5-slot packets> <encryption> <slot offset>  
              <timing accuracy> <role switch> <sniff mode> <SCO link>  
              <HV3 packets> <CVSD>  
[root@localhost ~]#
```

Figure 7.2 Technical Information of the Bluetooth Device (Mobile)

To query for the default packet type

```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hciconfig hci0 ptype  
hci0:  Type: USB  
       BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10  
       Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3  
[root@localhost ~]#
```

Figure 7.3 Different Packets in which data is sent

l2ping - L2CAP ping

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

```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hcitool scan  
Scanning ...  
[root@localhost ~]# hcitool inq  
Inquiring ...  
[root@localhost ~]# hcitool inq  
Inquiring ...  
[root@localhost ~]# hcitool scan  
Scanning ...  
00:0E:6D:DE:F2:42 PRAVEEN DARSHANAM  
[root@localhost ~]# l2ping 00:0E:6D:DE:F2:42  
Ping: 00:0E:6D:DE:F2:42 from 00:11:67:2D:4C:C5 (data size 44) ...  
0 bytes from 00:0E:6D:DE:F2:42 id 0 time 85.79ms  
0 bytes from 00:0E:6D:DE:F2:42 id 1 time 22.88ms  
0 bytes from 00:0E:6D:DE:F2:42 id 2 time 23.77ms  
0 bytes from 00:0E:6D:DE:F2:42 id 3 time 24.71ms  
0 bytes from 00:0E:6D:DE:F2:42 id 4 time 22.69ms  
0 bytes from 00:0E:6D:DE:F2:42 id 5 time 22.64ms  
0 bytes from 00:0E:6D:DE:F2:42 id 6 time 23.53ms  
0 bytes from 00:0E:6D:DE:F2:42 id 7 time 23.44ms  
0 bytes from 00:0E:6D:DE:F2:42 id 8 time 23.45ms  
0 bytes from 00:0E:6D:DE:F2:42 id 9 time 24.40ms  
0 bytes from 00:0E:6D:DE:F2:42 id 10 time 24.34ms  
0 bytes from 00:0E:6D:DE:F2:42 id 11 time 32.05ms
```

Figure 7.4 Pinging Devices

l2test - L2CAP testing

`l2test <mode> [-b bytes] [-P psm] [-I mtu] [-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>`

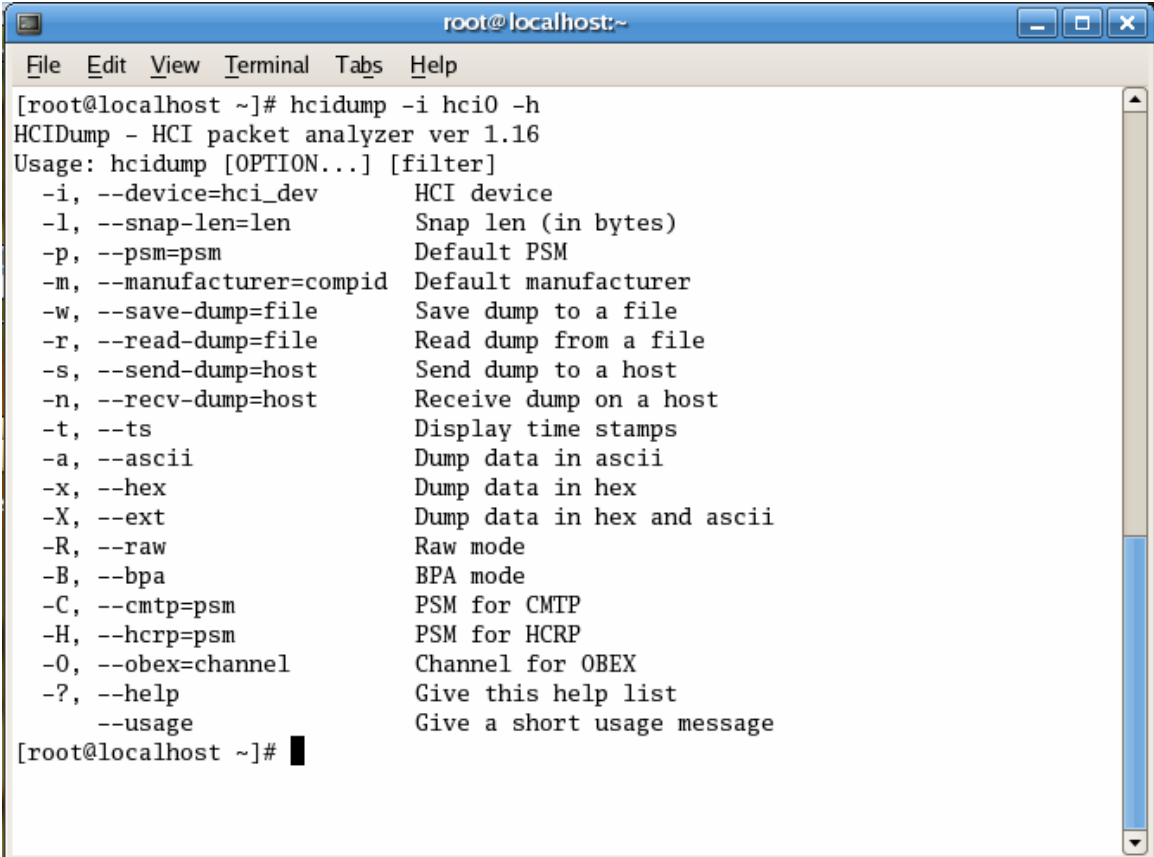
scotest - SCO testing

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

Modes:

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

hcidump - HCI packet analyzer



```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hcidump -i hci0 -h  
HCIDump - HCI packet analyzer ver 1.16  
Usage: hcidump [OPTION...] [filter]  
-i, --device=hci_dev      HCI device  
-l, --snap-len=len        Snap len (in bytes)  
-p, --psm=psm             Default PSM  
-m, --manufacturer=compid Default manufacturer  
-w, --save-dump=file       Save dump to a file  
-r, --read-dump=file       Read dump from a file  
-s, --send-dump=host       Send dump to a host  
-n, --recv-dump=host       Receive dump on a host  
-t, --ts                  Display time stamps  
-a, --ascii                Dump data in ascii  
-x, --hex                  Dump data in hex  
-X, --ext                  Dump data in hex and ascii  
-R, --raw                  Raw mode  
-B, --bpa                  BPA mode  
-C, --cmtpp=psm           PSM for CMTTP  
-H, --hcrpp=psm           PSM for HCRP  
-O, --obex=channel         Channel for OBEX  
-?, --help                 Give this help list  
--usage                    Give a short usage message  
[root@localhost ~]# █
```

Figure 7.5 HCI Packet Analyzer

hctool - Generic writing and monitoring to the HCI interface

```

root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# hcitool
hcitool - HCI Tool ver 2.25
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
    sr       Switch master/slave role
    cpt      Change connection packet type
    rssi     Display connection RSSI
    lq       Display link quality
    tpl      Display transmit power level
    afh      Display AFH channel map
    lst      Set/display link supervision timeout
    auth     Request authentication
    enc      Set connection encryption
    key      Change connection link key
    clkoff   Read clock offset
    clock    Read local or remote clock

For more information on the usage of each command use:
    hcitool <command> --help
[root@localhost ~]# █

```

Figure 7.6 Hcitol

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
```

Personal Computer (AMD Athlon, 64 bit Technology) and Bluetooth device (Nokia 6600) connected via USB.

```
[root@localhost Bluetooth]# lsmod
```

Module	Size	Used by
uhci	23040 0	(unused)
eeepro	100 15984 1	(autoclean)
usbcore	48784 1	[uhci]

The list of all modules present in the Linux operating system is given in the Appendix II.

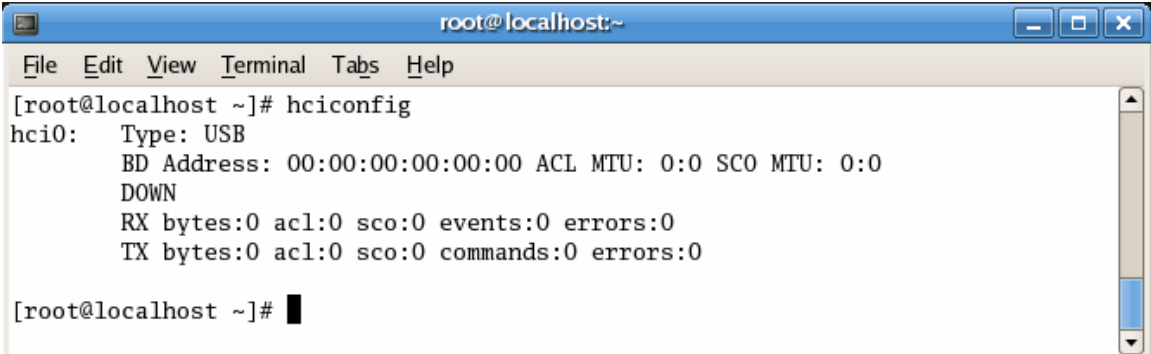
```
[root@localhost Bluetooth]# modprobe l2cap
```

```
[root@localhost Bluetooth]# lsmod
```

Module	Size	Used by
l2cap	15552 0	(unused)
bluez	20624 0	[l2cap]
uhci	23040 0	(unused)
eeepro	100 15984 1	(autoclean)
usbcore	48784 1	[uhci]

```
[root@localhost Bluetooth]# modprobe hci_usb
```

```
[root@localhost Bluetooth]# hciconfig
```

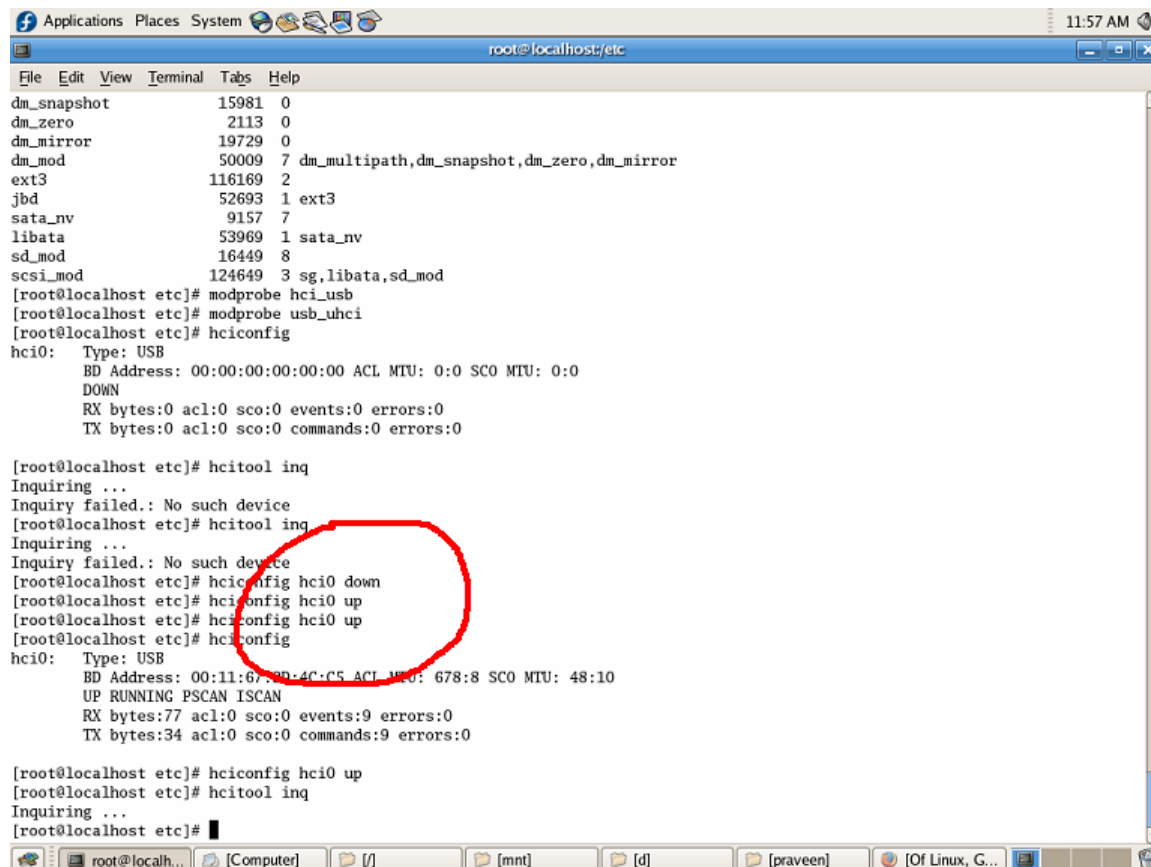


```
root@localhost:~  
File Edit View Terminal Tabs Help  
[root@localhost ~]# hciconfig  
hci0:  Type: USB  
      BD Address: 00:00:00:00:00:00 ACL MTU: 0:0 SCO MTU: 0:0  
      DOWN  
      RX bytes:0 acl:0 sco:0 events:0 errors:0  
      TX bytes:0 acl:0 sco:0 commands:0 errors:0  
[root@localhost ~]# █
```

Figure 7.7 Bluetooth USB Dongle is not connected

If the Dongle is not connected repeat the commands to invoke the device. That is shown in the figure below. We can manually bring device up by using the hciconfig command

hciconfig hci0 up



```
dm_snapshot      15981  0
dm_zero          2113  0
dm_mirror        19729  0
dm_mod           50009  7 dm_multipath,dm_snapshot,dm_zero,dm_mirror
ext3             116169  2
jbd              52693  1 ext3
sata_nv          9157  7
libata           53969  1 sata_nv
sd_mod           16449  8
scsi_mod         124649  3 sg,libata,sd_mod
[root@localhost etc]# modprobe hci_usb
[root@localhost etc]# modprobe usb_uhci
[root@localhost etc]# hciconfig
hci0:  Type: USB
      BD Address: 00:00:00:00:00:00 ACL MTU: 0:0 SCO MTU: 0:0
      DOWN
      RX bytes:0 acl:0 sco:0 events:0 errors:0
      TX bytes:0 acl:0 sco:0 commands:0 errors:0

[root@localhost etc]# hcitool inq
Inquiring ...
Inquiry failed.: No such device
[root@localhost etc]# hcitool inq
Inquiring ...
Inquiry failed.: No such device
[root@localhost etc]# hciconfig hci0 down
[root@localhost etc]# hciconfig hci0 up
[root@localhost etc]# hciconfig hci0 up
[root@localhost etc]# hciconfig
hci0:  Type: USB
      BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10
      UP RUNNING PSCAN ISCAN
      RX bytes:77 acl:0 sco:0 events:9 errors:0
      TX bytes:34 acl:0 sco:0 commands:9 errors:0

[root@localhost etc]# hciconfig hci0 up
[root@localhost etc]# hcitool inq
Inquiring ...
[root@localhost etc]#
```

Figure 7.8 Repetition of up and down cycles

After the repetition of the cycles try to invoke the device as shown in the figure below.


```
root@localhost:/mnt/g/ENTERTAINMENT/WALLPAPERS/praveen
File Edit View Terminal Tabs Help
[root@localhost praveen]# /etc/rc.d/init.d/bluetooth start
Starting Bluetooth services: [ OK ]
[root@localhost praveen]# hciconfig
hci0:  Type: USB
      BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10
      UP RUNNING PSCAN ISCAN
      RX bytes:2090 acl:0 sco:0 events:42 errors:0
      TX bytes:161 acl:0 sco:0 commands:28 errors:0

[root@localhost praveen]#
```

Figure 7.9 Starting Bluetooth

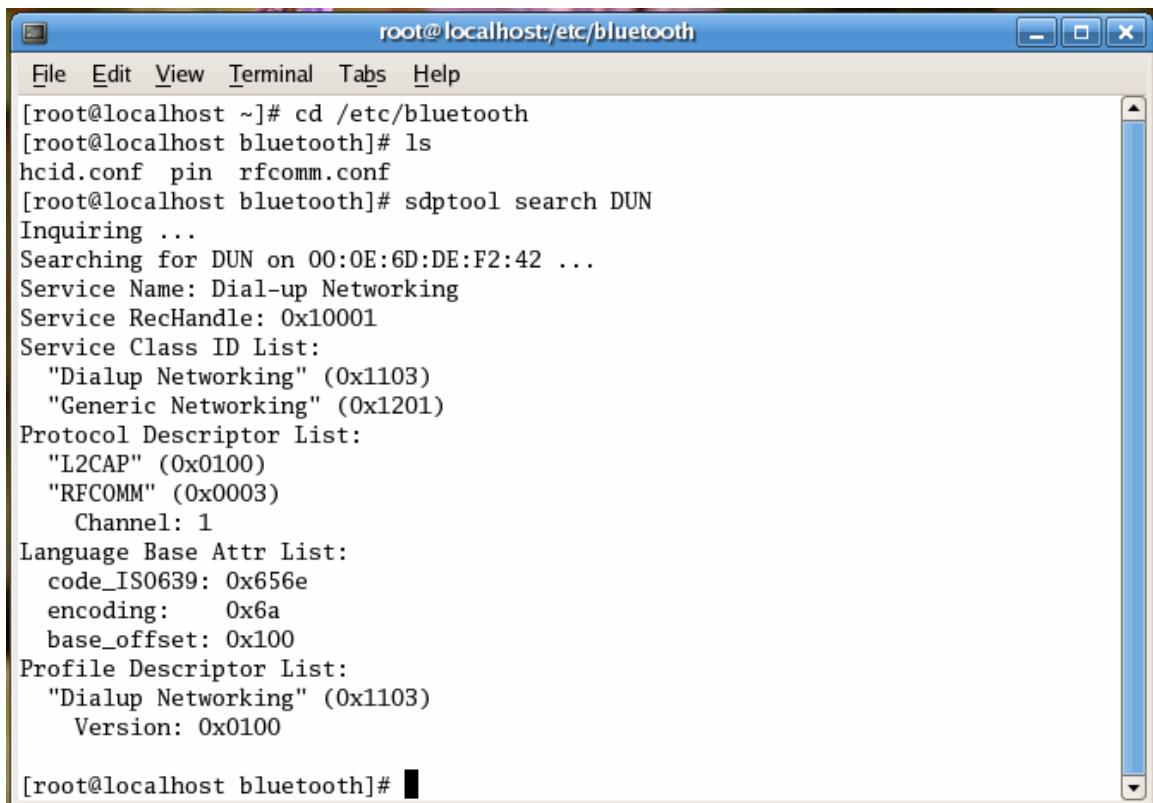
```
root@localhost:/mnt/g/ENTERTAINMENT/WALLPAPERS/praveen
File Edit View Terminal Tabs Help
hci0:  Type: USB
      BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10
      UP RUNNING PSCAN ISCAN
      RX bytes:354 acl:0 sco:0 events:12 errors:0
      TX bytes:43 acl:0 sco:0 commands:12 errors:0

[root@localhost praveen]# hciconfig -a
hci0:  Type: USB
      BD Address: 00:11:67:2D:4C:C5 ACL MTU: 678:8 SCO MTU: 48:10
      UP RUNNING PSCAN ISCAN
      RX bytes:354 acl:0 sco:0 events:12 errors:0
      TX bytes:43 acl:0 sco:0 commands:12 errors:0
      Features: 0xbf 0xfe 0x8d 0x78 0x08 0x18 0x00 0x00
      Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
      Link policy:
      Link mode: SLAVE ACCEPT
      Name: 'ISSCBTA'
      Class: 0x000000
      Service Classes: Unspecified
      Device Class: Miscellaneous,
      HCI Ver: 1.2 (0x2) HCI Rev: 0x1fe LMP Ver: 1.2 (0x2) LMP Subver: 0x1fe
      Manufacturer: Integrated System Solution Corp. (57)

[root@localhost praveen]#
```

Figure 7.10 hciconfig -a

The BlueZ stack includes an application named `rfcomm` that is used to connect two Bluetooth devices at the RFCOMM layer. Once the connection has been established `pppd` can be launched to place a new route for IP traffic between the two hosts, assuming that PPP is correctly configured.

A terminal window titled "root@localhost:/etc/bluetooth" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal output shows the user navigating to /etc/bluetooth, listing files (hcid.conf, pin, rfcmm.conf), and running "sdptool search DUN". The output displays details for the "Dial-up Networking" service, including its RecHandle (0x10001), Service Class ID List (Dialup Networking 0x1103, Generic Networking 0x1201), Protocol Descriptor List (L2CAP 0x0100, RFCOMM 0x0003), Language Base Attr List (code_IS0639: 0x656e, encoding: 0x6a, base_offset: 0x100), and Profile Descriptor List (Dialup Networking 0x1103, Version: 0x0100).

```
root@localhost:/etc/bluetooth
File Edit View Terminal Tabs Help
[root@localhost ~]# cd /etc/bluetooth
[root@localhost bluetooth]# ls
hcid.conf  pin  rfcmm.conf
[root@localhost bluetooth]# sdptool search DUN
Inquiring ...
Searching for DUN on 00:0E:6D:DE:F2:42 ...
Service Name: Dial-up Networking
Service RecHandle: 0x10001
Service Class ID List:
  "Dialup Networking" (0x1103)
  "Generic Networking" (0x1201)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
Language Base Attr List:
  code_IS0639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Dialup Networking" (0x1103)
    Version: 0x0100
[root@localhost bluetooth]#
```

Figure 7.11 Dial-Up Networking (DUN)

CHAPTER VIII

CONCLUSION AND FURTHER WORK

The thesis discussed the origins of the SIG and presented an overview of wireless communication concepts leading to the development of the Bluetooth technology and specification. Future versions of the specification are expected to provide higher data rates for Bluetooth (between 2 and 10 Mbps) and to provide the multimedia distribution facilities which are becoming a key requirement for the future of information technology. As Bluetooth and Wi-Fi began to capture the interest of the hi-tech industry, many understood the exciting potential of these technologies to revolutionize how people connect their devices. But negative publicity surfaced when analysts and members of the media speculated that the two technologies competed against each other. Most industry insiders and technology experts agree that the two wireless technologies do not compete but rather complement each other.

The scope for future work is huge. Going into the depths of Bluetooth protocol stack may help in performing practical setups like running Bluetooth robots, Bluetooth remote for TV's. Also mobile phones can used as remote for running music on MP3 players or on PC. The scope of Bluetooth is so vast that in future it may be part of every human beings life.

REFERENCES

- [1] Bluetooth Special Interest Group, Specification of the Bluetooth System, v2.0, <http://www.bluetooth.com>, November 2004.
- [2] B. A. Miller and C. Bisdikian, “Bluetooth Revealed: The Insider’s Guide to an Open Specification for Global Wireless Communications”, Prentice Hall, 2001.
- [3] J. Bray and C. F. Sturman, “Bluetooth: Connect Without Cables”, Prentice Hall 2001.
- [4] KVSSSS Sairam, N.Gunasekaran, S.Rama Reddy, “Bluetooth in Wireless Communication”, IEEE Communications Magazine, June 2002.
- [5] Chatschik Bisdikian, “An Overview of the Bluetooth Wireless Technology”, IEEE Communications Magazine, pages 86- 94, December 2001.
- [6] R. Shepherd, “Bluetooth wireless technology in the home”, Electronics & Communication Engineering Journal, pages 195- 203, October 2001.
- [7] Gregory Lamm, Gerlando Falauto, Jorge Estrada, Jag Gadiyaram, “Bluetooth Wireless Networks Security Features”, Proceedings of the IEEE, pages 265-272, June 2001.
- [8] C. Schurgers, V.Raghunathan, M. Srivastava, “Power management of energy-aware communication systems”, ACM Trans. Embedded CompSystems, vol. 2, iss.3, pages 431-447, August 2003.
- [9] E. Shih, “Wake on Wireless: An event driven energy saving strategy for battery operated devices”, Proc. MOBICOM, pages 160-171, 2002.

- [10] G. Tan, A. Miu, J. Gutang and H. Balakrishna, "Forming scatternets from Bluetooth personal area networks", MIT Technical Report, MITLCS-TR-826, October 2001.
- [11] Haartsen, J.C. and Zurbes, S., "Frequency hop selection in the Bluetooth radio system", IEEE Seventh International Symposium on Spread Spectrum Techniques and Applications, pages 83-87, Sep. 2002.
- [12] Zan Li, Yilin Chang and Lijun Jin, "A Novel Family of Frequency Hopping Sequences for Multi-hop Bluetooth Networks", IEEE Transactions on Consumer Electronics, Vol. 49, No. 4, pages 1084- 1089, November 2003
- [13] Ivan Howitt, "Mutual Interference Between Independent Bluetooth Piconets", IEEE Transactions On Vehicular Technology, vol. 52, no. 3, pages 708- 718, May 2003.
- [14] Ron Weinstein, "RFID: A Technical Overview and Its Application to the Enterprise", IEEE Computer Society-IT Pro, pages 27- 33, June 2005.
- [15] Jeremy Landt, "The history of RFID", IEEE POTENTIALS, pages 8- 11, November 2005.
- [16] IEEE 802.11b Working Group. (<http://grouper.ieee.org/groups/802/11>)
- [17] <http://www.usb.org>.
- [18] <http://www.bluetooth.org>
- [19] <http://www.forums.nokia.com>
- [20] <http://www.bluez.com>

To configure the UART devices you need to use the tool hciattach. It can be called either manually or from the PCMCIA cardmgr scripts.

```
#
# HCI daemon configuration file.
# $Id: bluezhowto.tex, v 1.5.1.2 2001/11/14 12:03:10 beutel Exp $
#
# HCId options
options {
    # Automatically 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 orchid;
    # /dev/ttyS1 57600 flow orchid;
    #/dev/ttyS0 57600 flow;
}
```

Setting up a PPP Link

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 10.4.42.15:192.168.50.1";
};
}
```

Start the server using ip 10.4.42.15 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";  
};  
}
```

Setting up a point-to-multipoint connection

Use 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
```


APPENDIX II

Module	Size	Used	by
autofs4	19013	1	
i2c_dev	8773	0	
i2c_core	20673	1	i2c_dev
rfcomm	34517	0	
l2cap	23617	10	hidp,rfcomm
sunrpc	136573	1	
ip_contrack_netbios_ns	3009	0	
xt_state	2241	9	
ip_contrack	49261	2	ip_contrack_netbios_ns,xt_state
nfnetlink	6489	1	ip_contrack
xt_tcpudp	3265	11	
iptable_filter	3137	1	
ip_tables	11657	1	iptable_filter
x_tables	1261	4	ipt_REJECT,xt_state,xt_tcpudp,ip_tables
hci_usb	15957	1	
bluetooth	44069	4	hidp,rfcomm,l2cap,hci_usb
vfat	11969	5	
fat	47709	1	vfat
dm_multipath	18121	0	
button	6609	0	
battery	9285	0	
ac	4933	0	
parport_pc	25445	1	
parport	34313	3	ppdev,lp,parport_pc
nvr	8393	0	
ohci1394	31749	0	
ieee1394	288665	1	ohci1394
ehci_hcd	29005	0	
ohci_hcd	19805	0	
snd_hda_intel	17233	1	
snd_hda_codec	112065	1	snd_hda_intel
forcedeth	22213	0	
snd_seq_device	8909	3	snd_seq_dummy,snd_seq_oss,snd_seq
snd_pcm_oss	45009	0	
snd_mixer_oss	16449	1	snd_pcm_oss
snd_pcm	76869	3	
			snd_hda_intel,snd_hda_codec,snd_pcm_oss
snd_timer	22597	2	snd_seq,snd_pcm
snd	50501	11	
			snd_hda_intel,snd_hda_codec,snd_seq_oss,snd_seq,snd_seq_device,snd_pcm_oss,snd_mixer_oss,snd_pcm,snd_timer
soundcore	9377	1	snd
snd_page_alloc	10441	2	snd_hda_intel,snd_pcm
dm_snapshot	15981	0	

dm_mod	50009	7	
dm_multipath, dm_snapshot, dm_zero, dm_mirror			
jbd	52693	1	ext3
sata_nv	9157	7	
libata	53969	1	sata_nv
scsi_mod	124649	3	sg, libata, sd_mod