한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Internet of Things (IoT) Systems

Week 9

Raspberry Pi Programming

Ikram Syed, Ph.D.
Associate Professor
Department of Information and Communication Engineering
Hankuk University of Foreign Studies (HUFS)

Spring – 2025

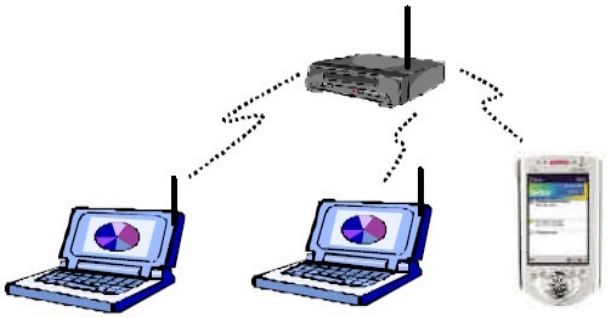
Why Wireless ?

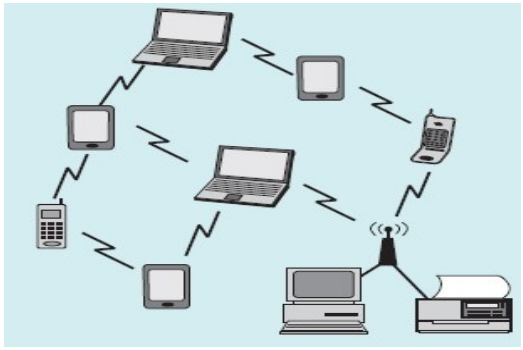
- IoT devices often need to send data without cables.
- Wireless makes devices mobile, flexible, and easy to install.
- Many wireless technologies exist: WiFi, Cellular, Zigbee, Bluetooth.

Wireless Networks

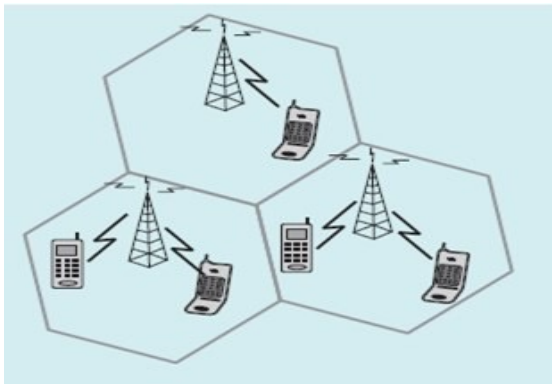
- Any type of computer network that utilizes some form of wireless network connection
- Infrastructure-based wireless networks
  - Centralized base station or access point
  - Communication via base station or access point
  - Require planning, installation, and management
- Ad-hoc wireless networks
  - A decentralized type of wireless networks
  - Ad hoc because it does not rely on a pre-existing network infrastructure such as routers or access points
    - Remote areas
    - Disaster relief
    - Military operations

Wireless LAN



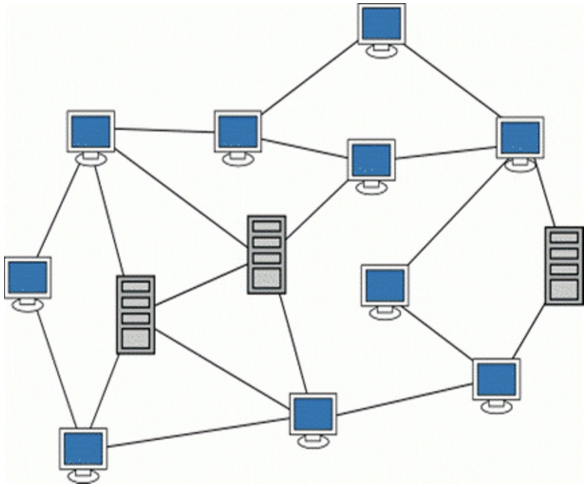


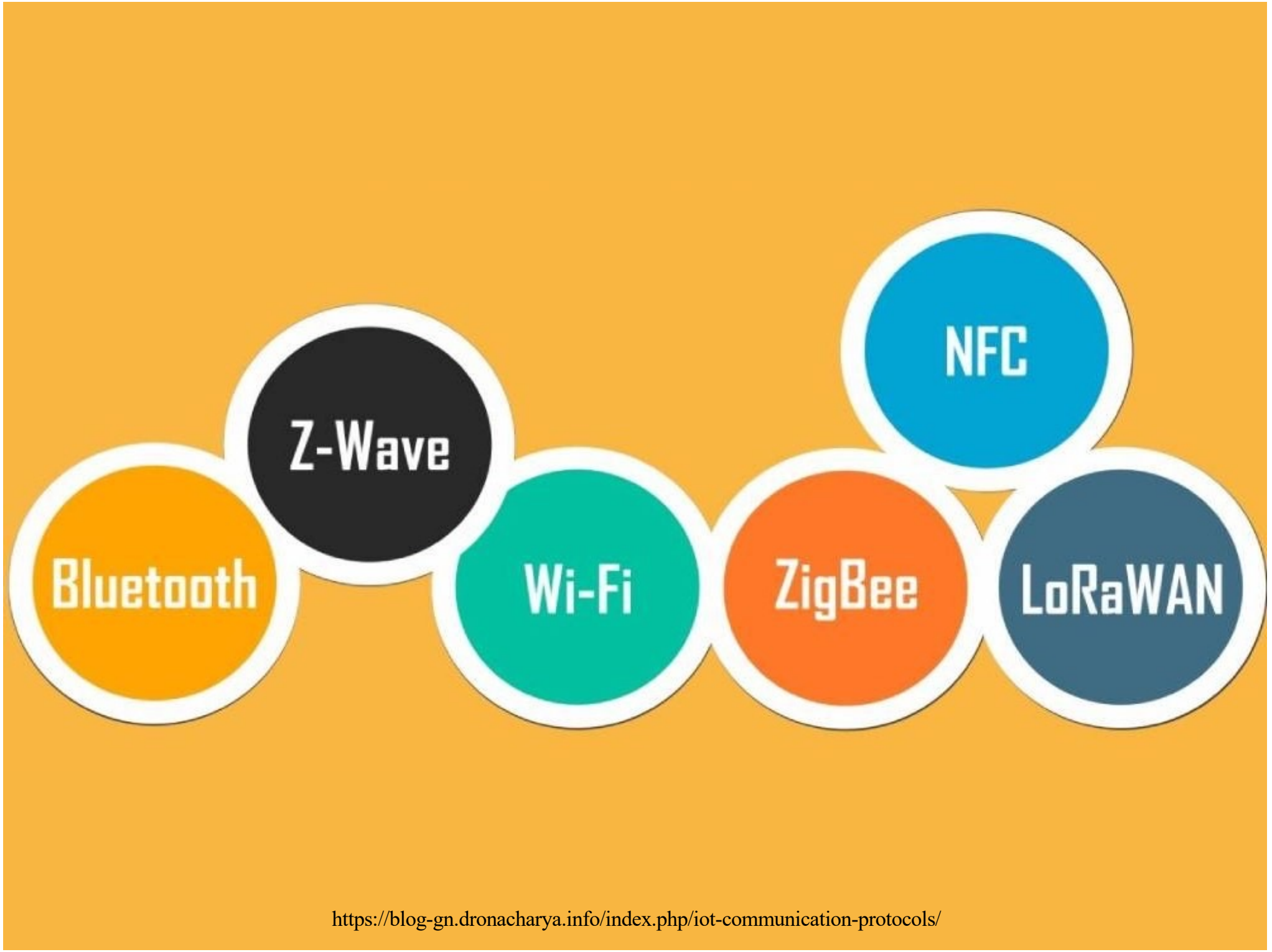
Cellular Network



Wireless Networks

- **Mobile ad hoc networking paradigms**
  - Mesh network is a network topology in which the infrastructure nodes connect directly, dynamically and non-hierarchically to as many other nodes as possible and cooperate with one another to efficiently route data
  - Sensor networks consist of spatially distributed devices communicating through wireless radio and cooperatively sensing physical or environmental conditions
  - Vehicular ad hoc network is a multi hop ad hoc network made up of vehicles.
  - Opportunistic mobile social networks are a form of mobile ad hoc networks that exploit the human social characteristics, such as similarities, daily routines, mobility patterns, and interests to perform the message routing and data sharing





https://blog-gn.dronacharya.info/index.php/iot-communication-protocols/

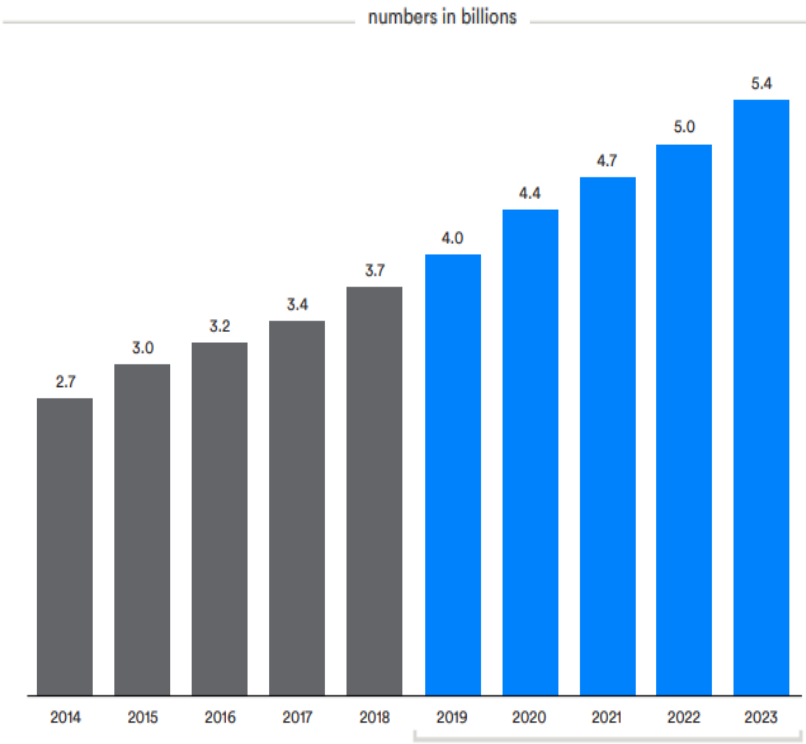
Attribute	Bluetooth® Low Energy Technology	Wi-Fi	Z-Wave	IEEE 802.15.4 (Zigbee, Thread)	LTE-M	NB-IoT	Sigfox	LoRaWAN
Range	10 m ~~– 1.5 km~~ 10m – 100m	15 m – 100 m	30 m - 50 m	30 m – 100 m	1 km – 10 km	1 km – 10 km	3 km – 50 km	2 km – 20 km
Throughput	125 kbps – 2 Mbps	54 Mbps – 1.3 Gbps	10 kbps – 100 kbps	20 kbps – 250 kbps	Up to 1 Mbps	Up to 200 kbps	Up to 100 bps	10 kbps – 50 kbps
Power Consumption	Low	Medium	Low	Low	Medium	Low	Low	Low
Ongoing Cost	One-time	One-time	One-time	One-time	Recurring	Recurring	Recurring	One-time
Module Cost	Under $5	Under $10	Under $10	$8-$15	$8-$20	$8-$20	Under $5	$8-$15
Topology	P2P, Star, Mesh, Broadcast	Star, Mesh	Mesh	Mesh	Star	Star	Star	Star
Shipments in 2019 (millions)	~3,500	~3,200	~120	~420	~7	~16	~10	~45

https://www.bluetooth.com/blog/wireless-connectivity-options-for-iot-applications/

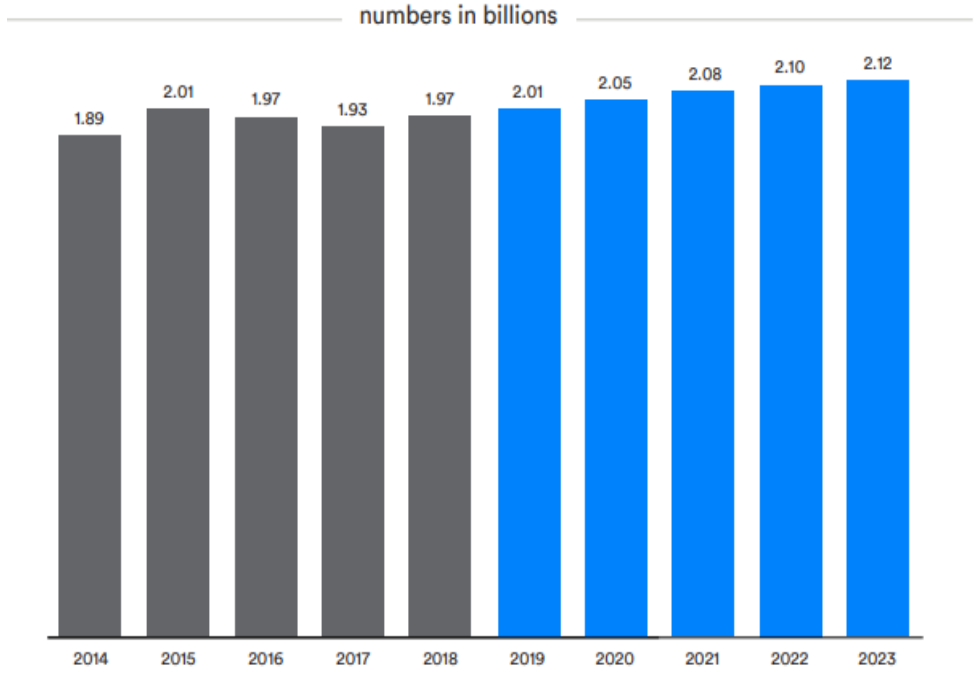
Bluetooth Technology

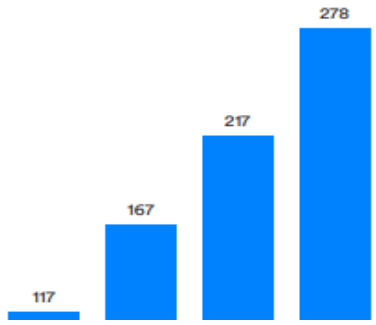
- Bluetooth is a wireless LAN technology used for exchanging data between fixed and mobile devices over short distances
- A Bluetooth LAN is an ad hoc network
- Low-cost and low-power
- Operates at 2.4 GHz ISM band
- IEEE 802.15.1 (WPANs)
- This technology was invented by Ericson in 1994
- It provides a communication platform between a wide range of devices

Total Annual Bluetooth Device Shipments

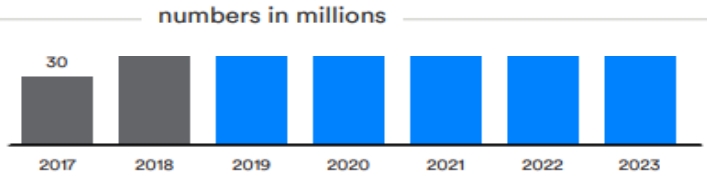


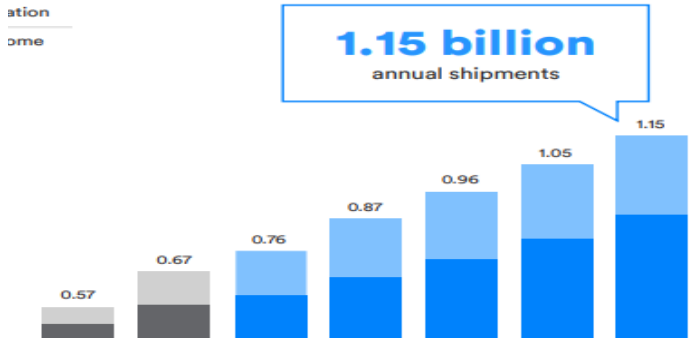
Annual Bluetooth Phone, Tablet, PC Device Shipments



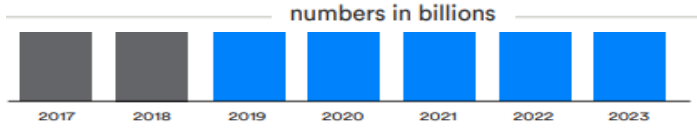


Annual Bluetooth Smart Industry Device Shipments





Annual Bluetooth Smart Home Device Shipments



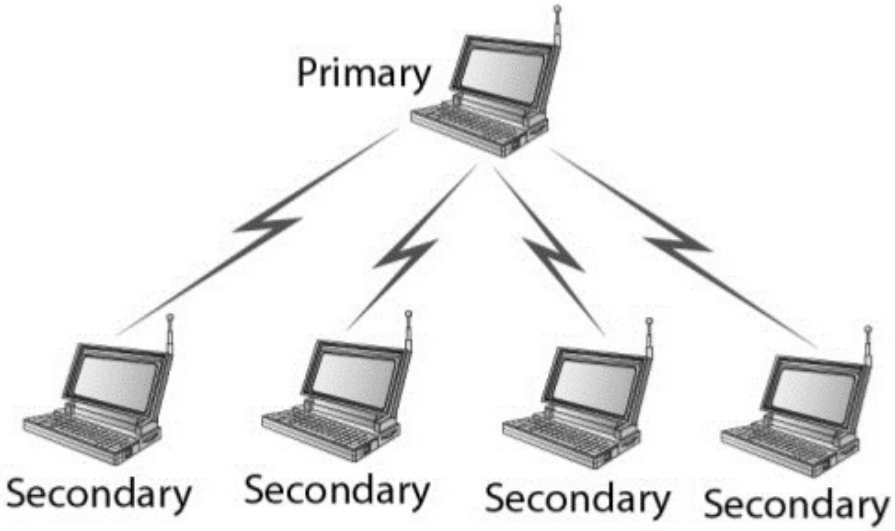
https://www.bluetooth.com/wp-content/uploads/2018/04/2019-Bluetooth-Market-Update.pdf

Bluetooth Technology

- Bluetooth is meeting market demand for
  - Audio Streaming
  - Data Transfer
  - Location Services
  - Device Networks
- Common Bluetooth Use Cases:
  - Smart watches
  - Wireless headphones
  - Health sensors (e.g., heart rate monitors)
  - Home automation (locks, lights)

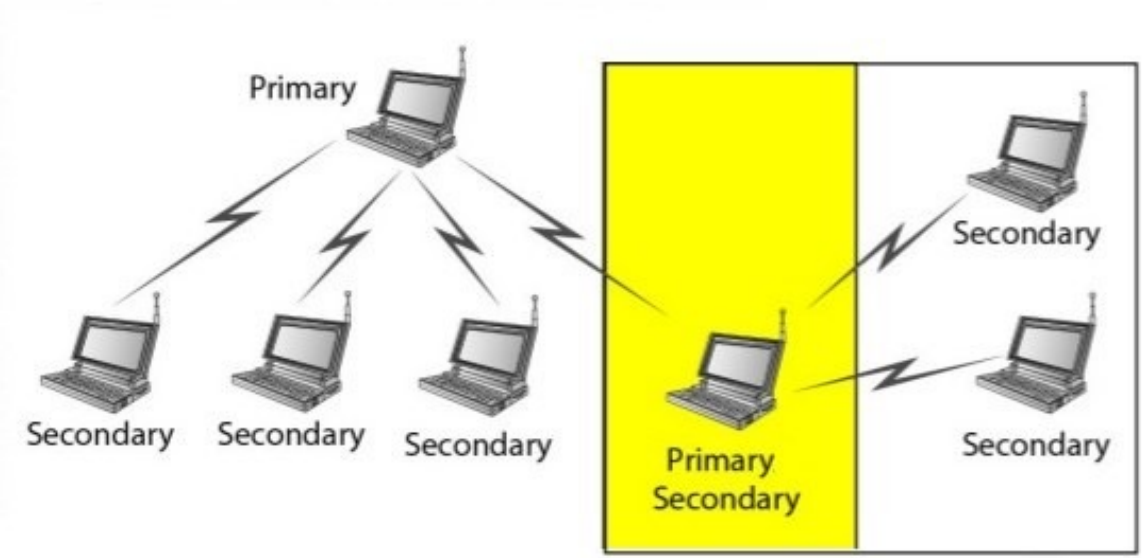
Bluetooth Technology

- **Bluetooth defines two types of networks**
  - **Piconet**
  - **Scatternet**
- A Bluetooth network is called a Piconet
  - A piconet can have up to eight stations
  - Secondary stations synchronize their clocks and hopping sequence with primary
  - The communication between primary and secondary can be
    - one-to-one
    - one-to-many



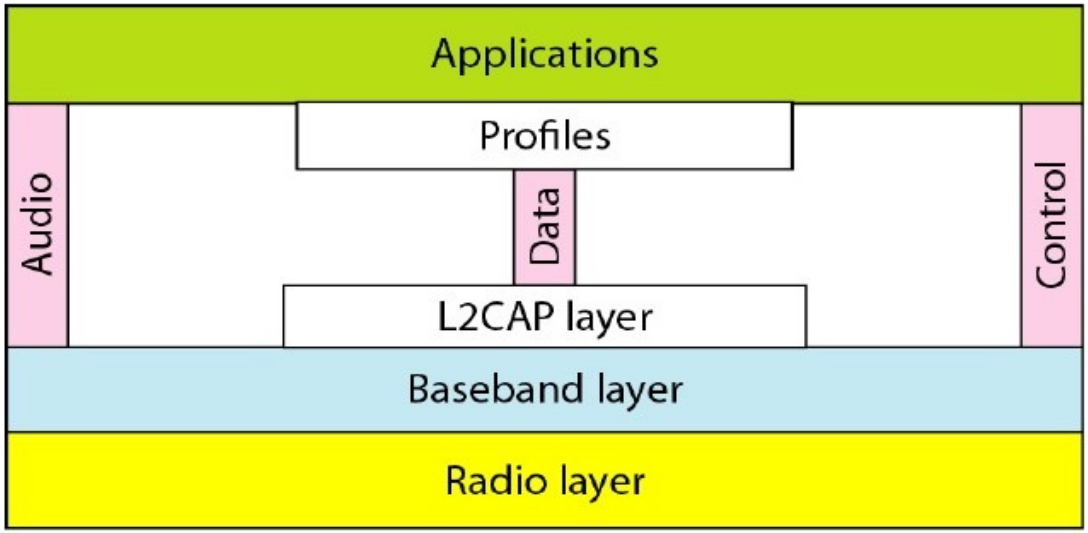
Bluetooth Technology

- Piconets can be combined to form what is called a Scatternet
  - Multiple interconnected Piconets
  - A secondary station in one piconet can be the primary in another piconet
  - This station can receive messages from the primary in the first piconet and deliver them to secondary stations in the second piconet



Bluetooth Technology

- Bluetooth Architecture
  - The Bluetooth protocol stack
    - The protocol stack defines how technology works
  - The Bluetooth profiles
    - The profiles define how to use Bluetooth technology to accomplish specific tasks



L2CAP: Logical Link Control and Adaptation Protocol

Bluetooth Technology

- Radio layer
  - The radio layer is roughly equivalent to the **physical layer** of the **Internet model**
  - The radio module in a Bluetooth device is responsible for **modulation** and **demodulation** of data into RF signals
  - Bluetooth devices operate at 2.4 GHz in the license-free, globally available ISM radio band
    - The advantage of operating in this band is worldwide availability and compatibility
    - A potential disadvantage is that Bluetooth devices must share this band with many other RF emitters such as ZigBee and WiFi
  - Physical range of 10 m
    - Bluetooth 5.0
      - 40–400 m

Bluetooth Technology

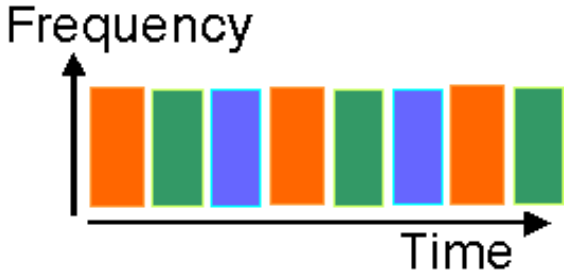
- Bluetooth uses the frequency-hopping spread spectrum (FHSS) method in the physical layer to avoid interference from other devices or networks
  - After a Bluetooth device sends or receives a packet, it and the device(s) it is communicating with *hop* to another frequency before next packet is sent.
  - This scheme has two main advantages
    - 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.

Bluetooth Technology

- There are three classes of BT devices communication
  - Class 1
    - Laptops and desktops
    - Range 100 meters
    - Power 100mW (20dBm)
  - Class 2
    - Phones and headsets
    - Range 20~50 meters
    - Power 2.5mW (4 dBm)
  - Class 3
    - Extremely low power devices
    - Range 1~10 meters
    - Power 1mW (0 dBm)

Bluetooth Technology

- Baseband layer
  - The baseband layer is roughly equivalent to the **MAC sublayer** in LANs
  - Bluetooth uses a form of TDMA
    - Time division duplex TDMA
  - The primary and secondary communicate using time slots

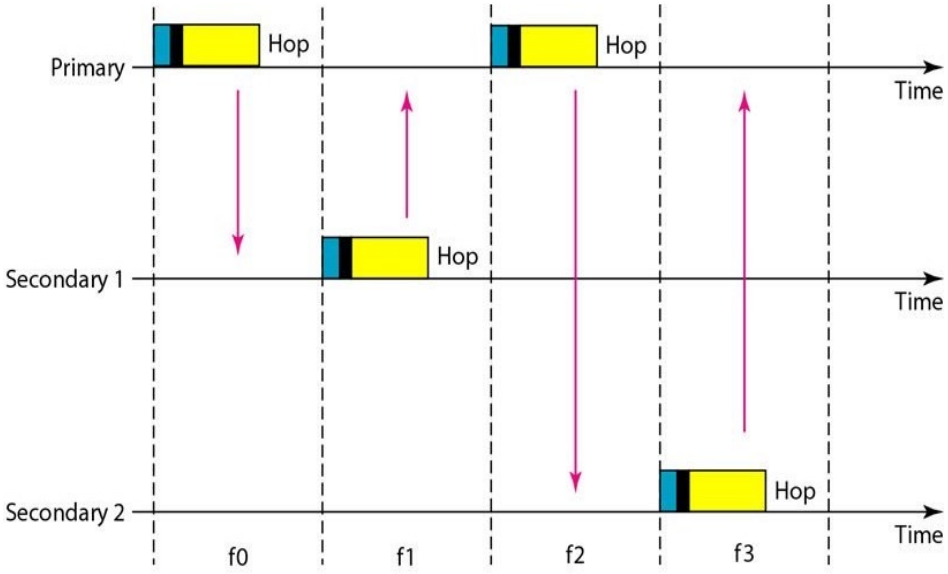


  - Single-Secondary Communication
    - The time is divided into slots of 625 μs
    - The primary uses **even numbered** slots and secondary uses **odd-numbered** slots

Bluetooth Technology

- Baseband layer
  - Multiple-Secondary Communication
    - The primary uses the even-numbered slots
    - All secondary units listen on even-numbered slots, **but only one secondary sends** in any odd-numbered slot

In slot 0, primary sends a frame to secondary 1
In slot 1, only secondary 1 sends a frame to primary
because previous frame was addressed to secondary 1
In slot 2, primary sends a frame to secondary 2
In slot 3, only secondary 2 sends a frame to primary
If secondary has no frame to send, channel is silent.



Bluetooth Technology

- The Bluetooth specification defines two types of links between BT devices
  - Synchronous connection-oriented (SCO)
    - A synchronous connection-oriented link is used when avoiding latency is more important
    - A physical link is created between the primary and a secondary by reserving specific slots at regular intervals
    - No retransmission if packet is damaged
  - Asynchronous connectionless link (ACL) is used when error-free delivery is more important than avoiding latency
    - Retransmission if packet is damaged

Bluetooth Technology

- The Logical Link Control and Adaptation Protocol (L2CAP)
  - It is used for data exchange on an ACL link
  - SCO channels do not use L2CAP
  - Multiplexing
  - Segmentation and reassembly
  - Quality of service
  - Group management

Bluetooth Technology

- Multiplexing
  - At the sender site, it accepts data from one of the upper-layer protocols, frames them, and delivers them to the baseband layer
  - At the receiver site, it accepts a frame from the baseband layer, extracts the data, and delivers them to the appropriate protocol layer
- Segmentation and Reassembly
  - The maximum size of payload field in baseband layer is 2774 bits or 343 bytes
  - Application layers sometimes need to send a data packet that can be up to 65,535 bytes
  - The L2CAP divides these large packets into segments and adds extra information to define the location of the segments in the original packet
  - The L2CAP segments the packet at the source and reassembles them at the destination

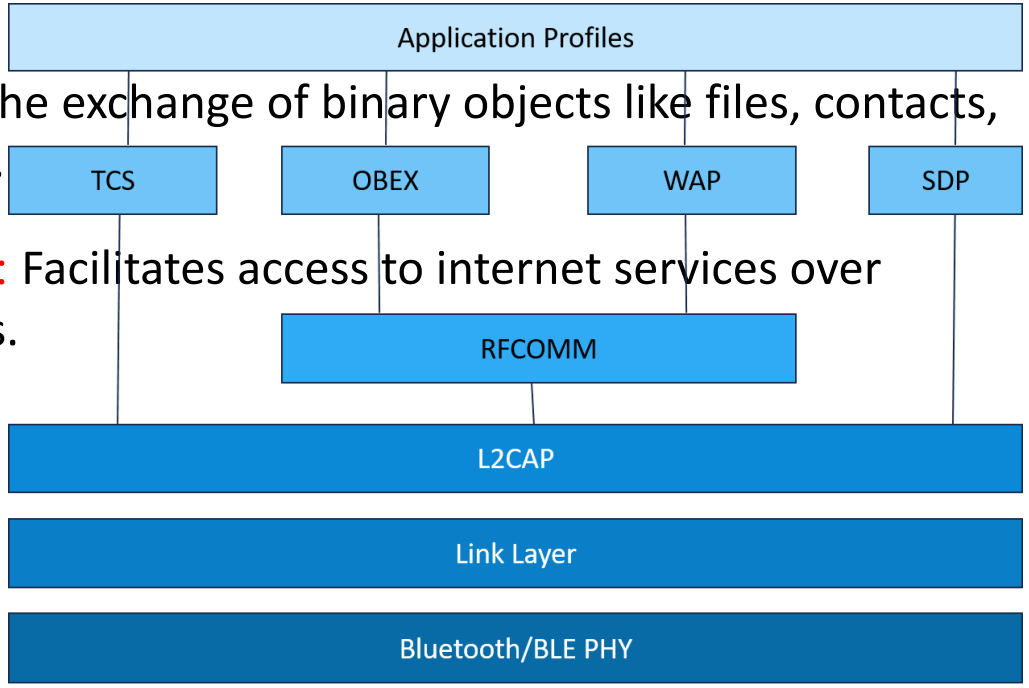
Bluetooth Technology

- Quality of service
  - Bluetooth allows the stations to define a quality-of-service level
  - If no quality-of-service level is defined, Bluetooth defaults to what is called best-effort service
- Group Management
  - Another functionality of L2CAP is to allow devices to create a type of logical addressing between themselves
  - For example, two or three secondary devices can be part of a multicast group to receive data from the primary

Bluetooth Technology

- Bluetooth defines several **protocols** for the upper layers that use the services of L2CAP

  - Service discovery protocol (SDP): Allows Bluetooth devices to discover available services and their characteristics on other devices.
  - Radio frequency communication (RFCOMM): Emulates serial port communication over Bluetooth for simple data transfer.
  - Telephony control protocol (TCS): Supports call control signaling for voice and telephony services over Bluetooth.
  - (Object Exchange) OBEX: Enables the exchange of binary objects like files, contacts, and calendar entries over Bluetooth.
  - (Wireless Application Protocol) WAP: Facilitates access to internet services over Bluetooth, mainly for mobile devices.



Bluetooth Connection

- A connection between two devices occur in the following fashion
  - Nothing is known about a remote device
    - The inquiry and page procedure
  - Some details are known about a remote device
    - The paging procedure
- Inquiry procedure enables a device to discover which devices are in range, and determine the addresses and clocks for the devices.
  - A device (inquirer) sends out inquiry messages and receive inquiry reply
  - Device sends inquiry packets on 16 different frequencies

    (16 channel train)
- Inquiry Scan A device periodically listens for inquiry packets at a single frequency chosen out of 16 frequencies.
  - It will re-enter inquiry scan state even after responding to an inquire

Bluetooth Connection

- Inquiry Response
  - When a device receives inquire, it will wait between 0 and 0.32 seconds before sending an FHS packet as a response
    - This is done to avoid collision with another device that also wants to send an FHS packet
  - FHS Packet contains
    - Device ID
    - Clock
  - After inquiring procedure, inquiring device knows all discoverable devices within range

Bluetooth Connection

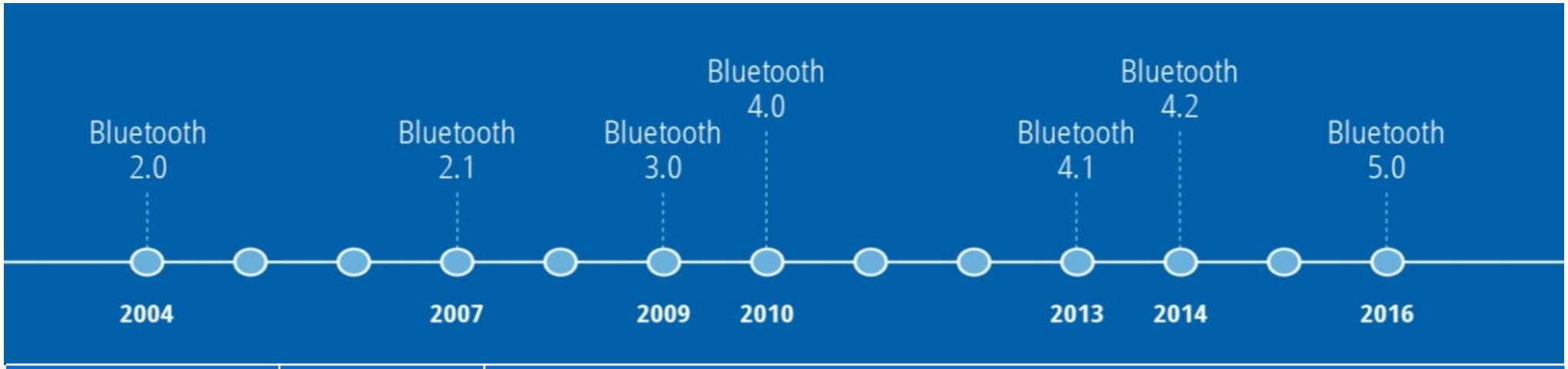
- Paging procedure
  - A unit that establishes a connection will carry out a page procedure and will automatically be the master of the connection.
  - Connection process involves a 6 steps of communication between the master and the slave
  - Slave ID from Master ➔ Slave: page message
  - Response from Slave ➔ Master
  - Master sends FHS packet ➔ Slave

    (**address**, **clock info**, and **hop sequence**.)
  - Slave acknowledges FHS ➔ Master
  - First Master Packet (Connection Request)
  - First Slave Packet (Response)

Step	Message	Direction
1	Slave ID	Master to Slave
2	Slave ID	Slave to Master
3	FHS	Master to Slave
4	Slave ID	Slave to Master
5	1st Master Packet	Master to Slave
6	1st Slave Packet	Slave to Master

Bluetooth Technology

- A device in connection state can be in following modes
  - Active mode is a regular connected mode, where device is actively transmitting or receiving data
  - Sniff mode is a power-saving mode, where device is less active. It sleeps and only listen for transmissions at a set interval
  - Hold mode is a temporary, power-saving mode where a device sleeps for a defined period and then returns back to active mode when that interval has passed. The master can command a slave device to hold.
  - Park mode is a deepest of sleep modes. A master can command a slave to *park*, and that slave will become inactive until master tells it to wake back up



Version	Year	Major Improvements
1.2	2003	Faster connection and discovery, adaptive frequency hopping, introduced flow control and retransmission modes
2.0	2004	2.1 Mbps peak data rates
2.1	2007	3.0 Mbps peak data rates
3.0	2009	24 Mbps peak data rates using Wi-Fi PHY + Bluetooth PHY for lower rates
4.0	2010	Lower energy consumption, broadcasting, lower connection latency
4.2	2014	Improved security, low energy packet length extension, link layer privacy
5.0	2016	48 Mbps peak data rates, energy efficiency, higher broadcasting message capacity, larger range and strong point-to-point connection and reliability
5.2	2020	LE Audio (better sound quality) Enhanced ATT(faster data tranfer) LE power
5.4	2023	Periodic Advertising with Responses (PAwR) 2Mbps

Bluetooth Technology

- Configure Bluetooth Environment
  - Install Bluetooth Terminal Program on Android Device

[Figure 4-6] Installing Bluetooth Terminal

Bluetooth terminal
wingoodhar

Bluetooth Technology

- Configure Bluetooth Environment for RaspberryPi

  Check BT module

```
pi@raspberrypi:~ $ hciconfig
hci0:   Type: BR/EDR  Bus: UART
        BD Address: B8:27:EB:DE:37:3B  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING PSCAN
        RX bytes:724 acl:0 sco:0 events:43 errors:0
        TX bytes:1537 acl:0 sco:0 commands:43 errors:0
```

[Figure 4-7] The Result of *hciconfig*

Bluetooth Technology

Activate BT module

HCI = **Host Controller Interface**
It is the **interface** between the **Bluetooth software (host)** and the **Bluetooth hardware (controller)**.

```
pi@raspberrypi:~ $ sudo hciconfig hci0 up
```

[Figure 4-8] Activating Bluetooth

```
pi@raspberrypi:~ $ sudo hciconfig hci0 name BluePi
```

[Figure 4-9] Specifying the Name Bluetooth

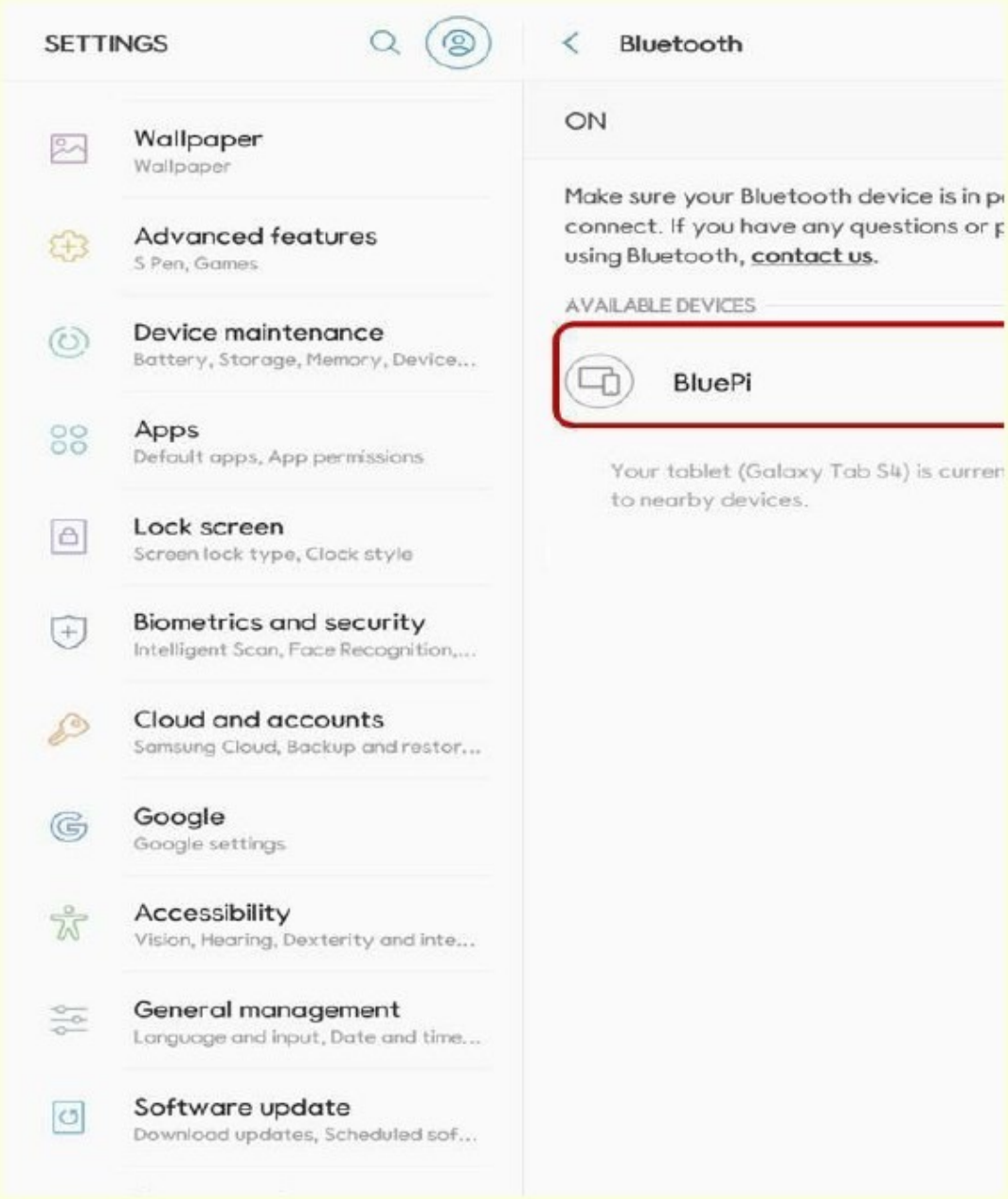
```
pi@raspberrypi:~ $ sudo hciconfig hci0 sspmode 1
```

[Figure 4-10] Switching Simple Pairing Mode of Bluetooth

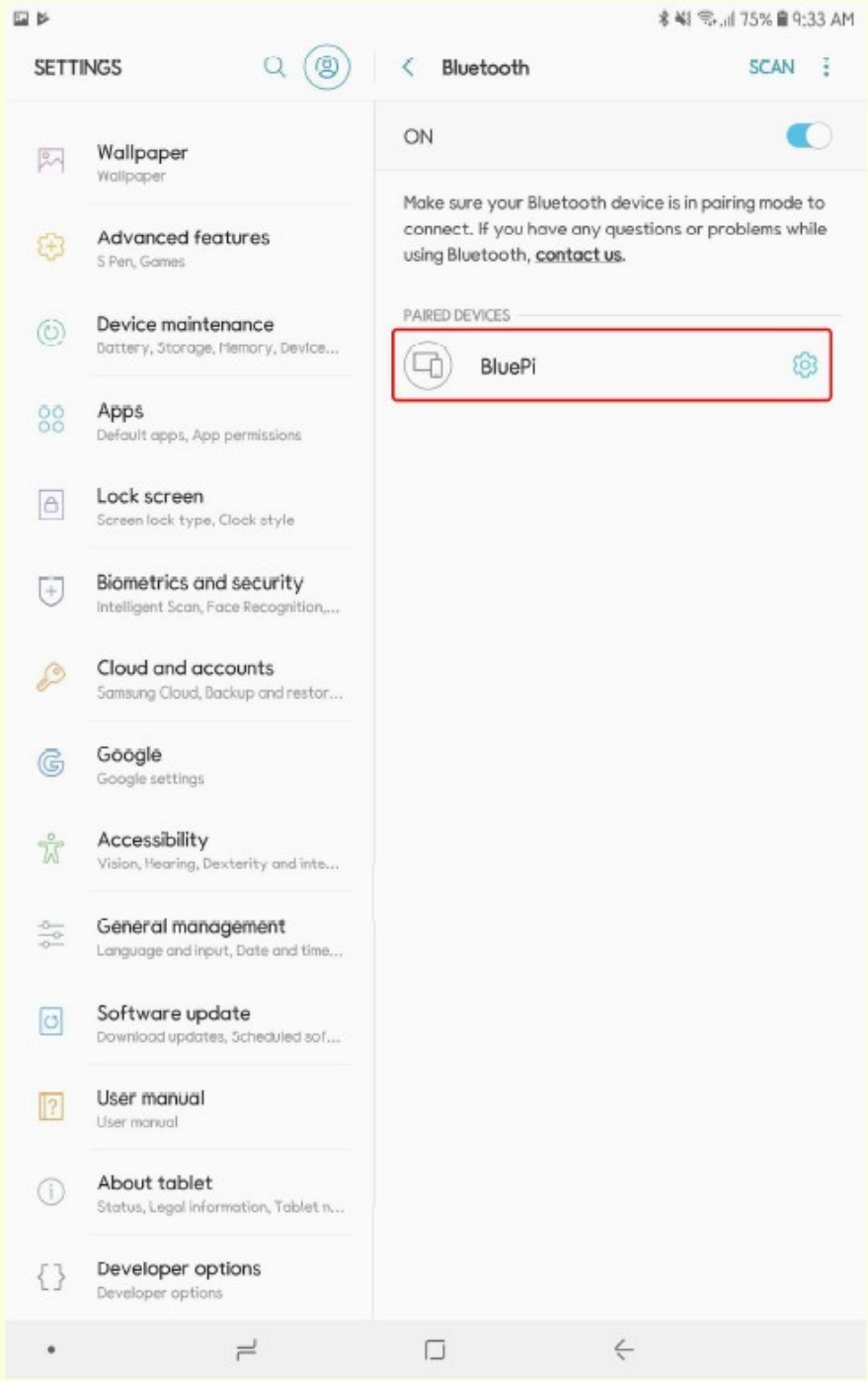
```
pi@raspberrypi:~ $ sudo hciconfig hci0 piscan
```

[Figure 4-11] Allowing Bluetooth Scanning

Search for BT on android device



[Figure 4-12] Searching Bluetooth Devices



[Figure 4-13] Pairing Bluetooth Module

Bluetooth Technology

- Modify BT library file to facilitate Bluetooth programming
  - Open following file in editor

```
pi@raspberrypi:~ $ sudo nano /etc/systemd/system/dbus-org.bluez.service
```

[Figure 4-14] Opening */etc/systemd/system/dbus-org.bluez.service*

Add '--compat' after 'ExecStart = / usr / lib / bluetooth / bluetoothd' and save it

```
[Unit]
Description=Bluetooth service
Documentation=man:bluetoothd(8)

[Service]
Type=dbus
BusName=org.bluez
ExecStart=/usr/lib/bluetooth/bluetoothd --compat
NotifyAccess=main
#WatchdogSec=10
#Restart=on-failure
CapabilityBoundingSet=CAP_NET_ADMIN CAP_NET_BIND_SERVICE
LimitNPROC=1

[Install]
WantedBy=bluetooth.target
Alias=dbus-org.bluez.service
```

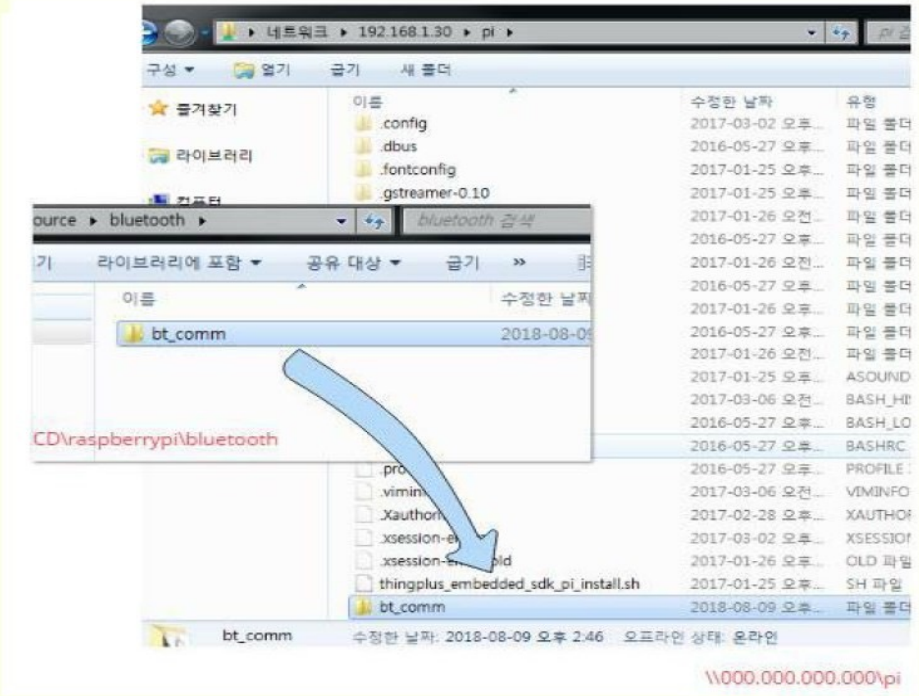
[Figure 4-15] File Contents of */etc/systemd/system/dbus-org.bluez.service*

--compat is to enable backward compatibility

```
pi@raspberrypi:~ $ sudo reboot
```

[Figure 4-16] Rebooting Command

Copy the *bt_comm* folder on the *CD ₩ Raspberry Pi ₩ source ₩ bluetooth* to the *pi* using Samba server.



[Figure 4-17] Copy *bt_comm* Folder

```
pi@raspberrypi:~ $ cd bt_comm/
pi@raspberrypi:~/bt_comm $
```

[Figure 4-18] *bt_comm* Folder

```
pi@raspberrypi:~/bt_comm $ ls -l
total 12
-rwxr--r-- 1 pi pi 7476 Jul 25 05:59 bt_master.h
-rwxr--r-- 1 pi pi  322 Aug  9 07:30 bt_server.c
```

[Figure 4-19] Folder Contents of *bt_comm*

Bluetooth Technology

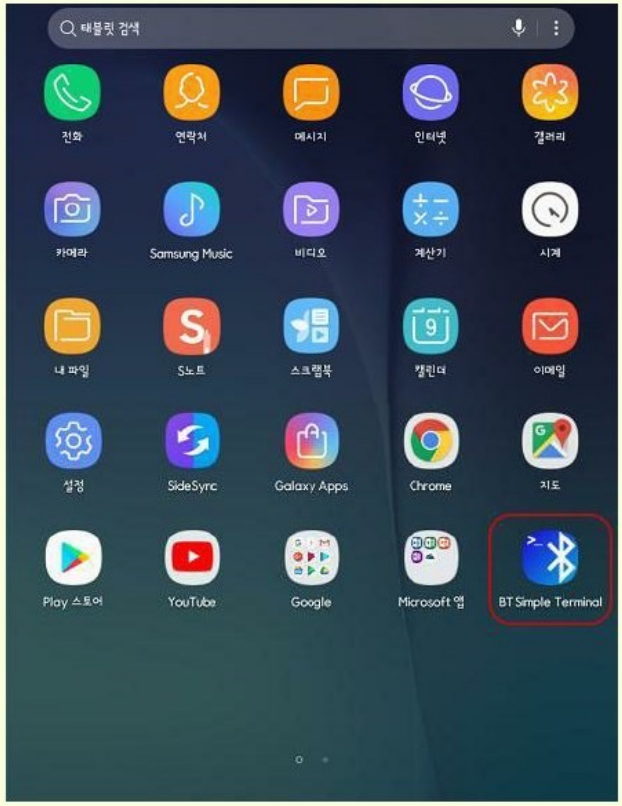
- Compile and execute bt_server program

```
pi@raspberrypi:~/bt_comm $ gcc -o bt_server bt_server.c -lbluetooth
pi@raspberrypi:~/bt_comm $ sudo ./bt_server
Registering UUID 00001101-0000-1000-8000-00805f9b34fb
socket() returned 4
bind() on channel 3 returned 0
listen() returned 0
calling accept()
```

[Figure 4-20] *bt_server* Program

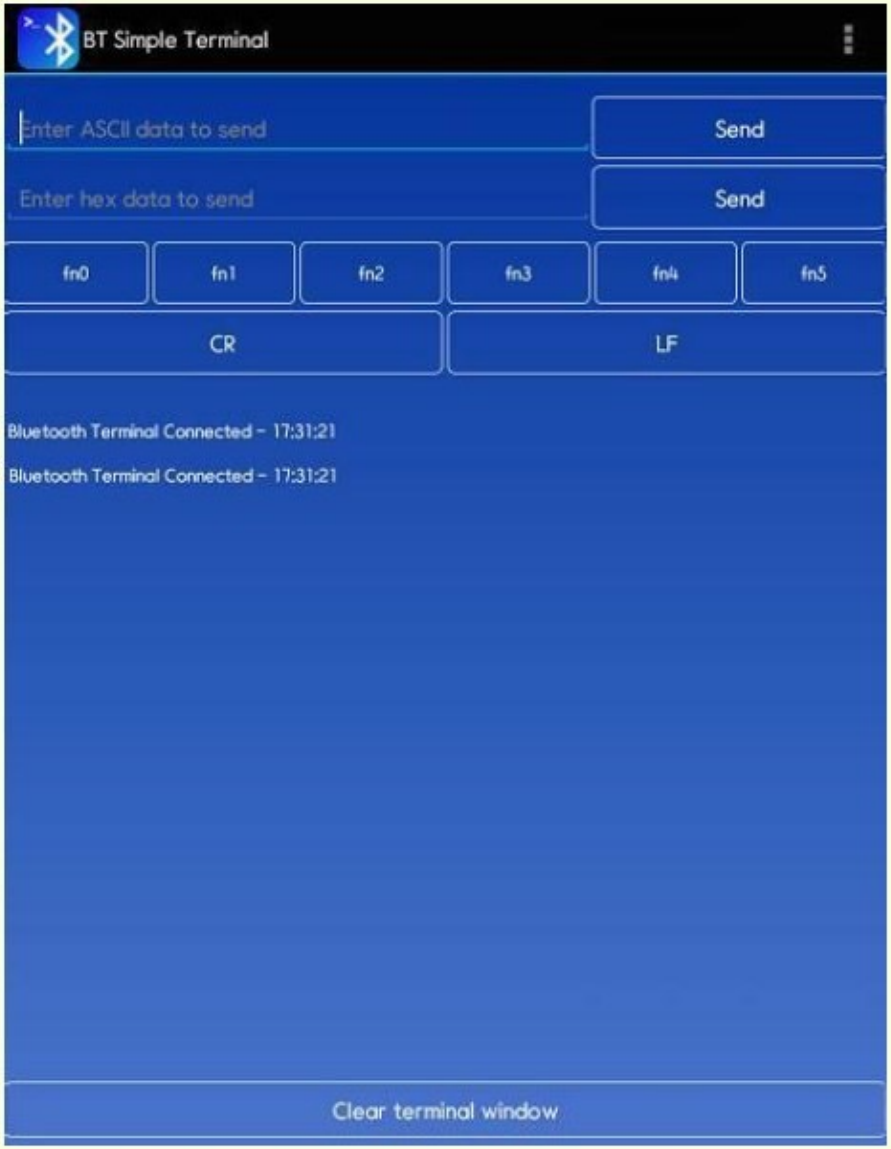
Bluetooth Technology

- Run BT Simple Terminal on android device to connect to echo server program

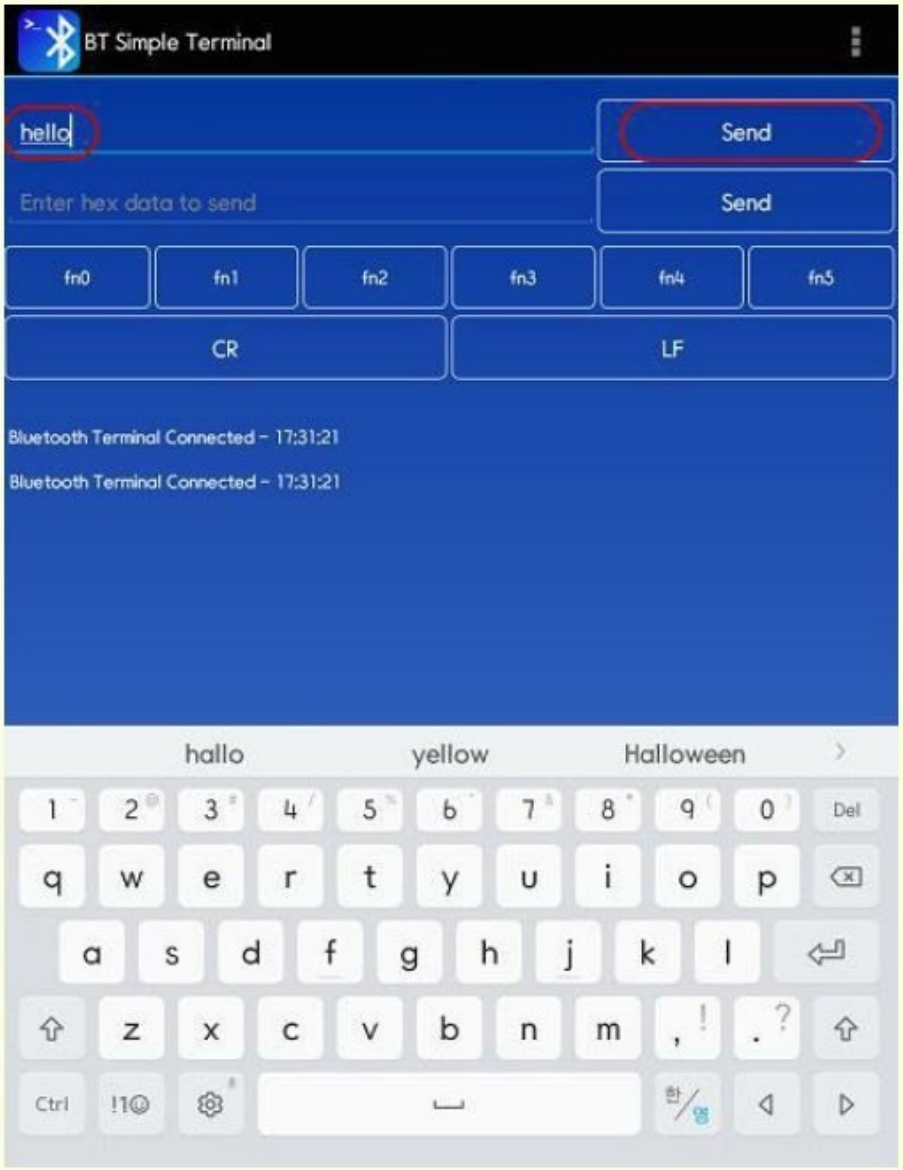


[Figure 4-21] Running the Terminal

[Figure 4-22] Pairing List



[Figure 4-23] Connected Terminal



[Figure 4-24] Data Transmission

```
pi@raspberrypi:~/bt_comm $ sudo ./bt_server
Registering UUID 00001101-0000-1000-8000-00805f9b34fb
socket() returned 4
bind() on channel 3 returned 0
listen() returned 0
calling accept()


accept() returned 5
accepted connection from F4:42:8F:38:BD:5C


Bluetooth Terminal Connected - 15:00:58


Bluetooth Terminal Connected - 15:00:58


Bluetooth Terminal Connected - 15:00:58


hello
hello
hello
```

[Figure 4-25] Data Receive

Any Questions!

bt_server program

```c
#include "bt_master.h"
#include <unistd.h>

int main()
{
        int client = init_server();

        char *recv_message;

        while(1){

        recv_message = read_server(client);

        if ( recv_message == NULL ){
            printf("client disconnected\n");
            break;
        }

        write_server(client, recv_message);
    }
}
```

Remote control program for LED

```c
#include "bt_master.h"
#include <unistd.h>
#include <wiringPi.h>

#define PIN 7

int main()
{
    int client = init_server();

    char *recv_message;
    char *send_message;

    if(wiringPiSetup() == -1) return 1;

    pinMode(PIN,OUTPUT);

    while(1) {
        recv_message = read_server(client);
        if ( recv_message == NULL ){
            printf("client disconnected\n");
            break;
        }
    if(strcmp(recv_message,"LEDON") == 0){
        digitalWrite(PIN,HIGH);
        strcpy(recv_message, "LED ON!\n");
    }else if(strcmp(recv_message, "LEDOFF") == 0){
        digitalWrite(PIN,LOW);
        strcpy(recv_message, "LED OFF!\n");
    }

        write_server(client, recv_message);
    }
}
```

```
pi@raspberrypi:~/bt_comm $ gcc -o BT_LED BT_LED.c -lwiringPi -lbluetooth
```

[Figure 4-29] Compiling BT_LED.c

```
pi@raspberrypi:~/bt_comm $ gcc -o BT_LED BT_LED.c -lwiringPi -lbluetooth
pi@raspberrypi:~/bt_comm $ sudo ./BT_LED
Registering UUID 00001101-0000-1000-8000-00805f9b34fb
socket() returned 4
bind() on channel 3 returned 0
listen() returned 0
calling accept()
accept() returned 5
accepted connection from F4:42:8F:38:BD:5C









Bluetooth Terminal Connected - 15:11:23

Bluetooth Terminal Connected - 15:11:23

Bluetooth Terminal Connected - 15:11:23


LEDON
LEDON
LED ON!


LEDOFF
LEDOFF
LED OFF!
```

[Figure 4-30] The Screen of Raspberry Pi

[Figure 4-31] The Screen of Android

Human Detection Sensor

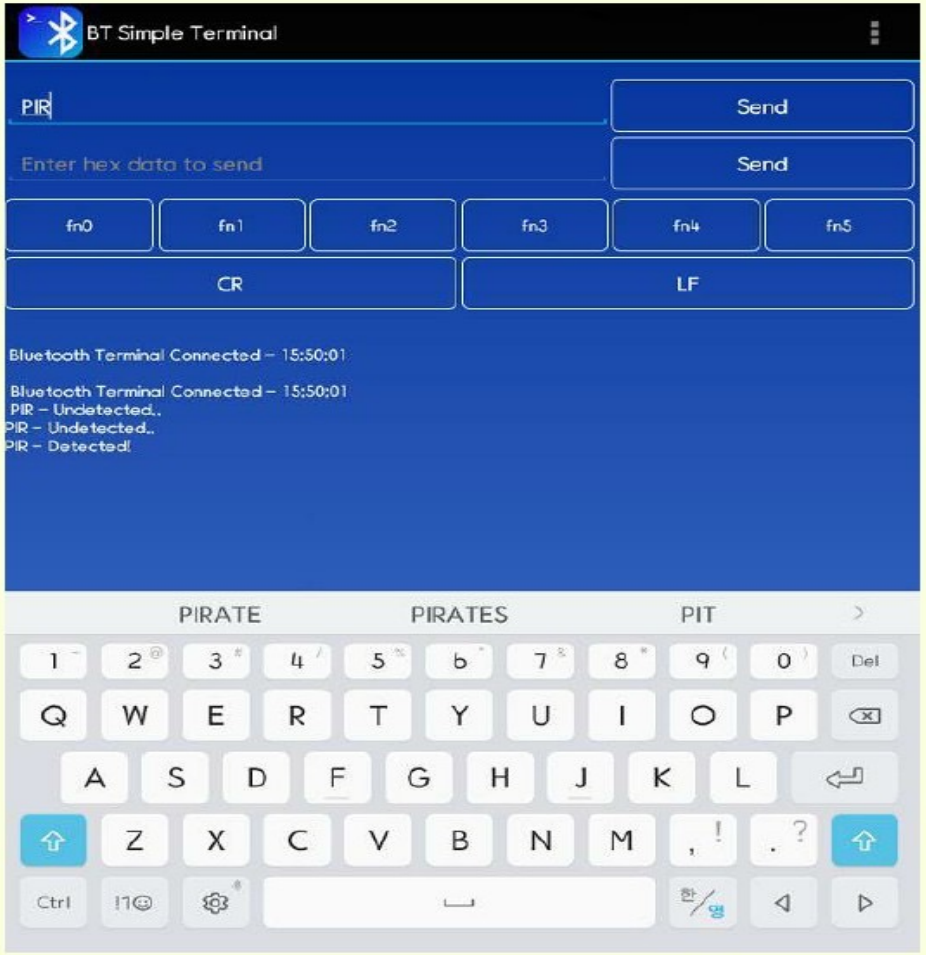
```c
#include "bt_master.h"
#include <unistd.h>
#include <wiringPi.h>

#define PIN 2

int main()
{
        int client = init_server();
        int pir;
        char *recv_message;

        if(wiringPiSetup() == -1) return 1;

        pinMode(PIN,INPUT);

        while(1){

        recv_message = read_server(client);
        if ( recv_message == NULL ){
            printf("client disconnected\n");
            break;
        }
        if(strcmp(recv_message,"PIR") == 0){
            pir = digitalRead(PIN);

        if(pir == HIGH){
            strcpy(recv_message, "PIR - Detected!\n");
        }else if(pir == LOW){
            strcpy(recv_message, "PIR - Undetected..\n");
        }
        }
        write_server(client, recv_message);
    }
}
```

```
pi@raspberrypi:~/bt_comm $ gcc -o BT_PIR BT_PIR.c -lwiringPi -lbluetooth
```

[Figure 4-33] Compiling BT_PIR.c

```
pi@raspberrypi:~/bt_comm $ gcc -o BT_PIR BT_PIR.c -lwiringPi -lbluetooth
pi@raspberrypi:~/bt_comm $ sudo ./BT_PIR
Registering UUID 00001101-0000-1000-8000-00805f9b34fb
socket() returned 4
bind() on channel 3 returned 0
listen() returned 0
calling accept()
accept() returned 5
accepted connection from F4:42:8F:38:BD:5C

Bluetooth Terminal Connected - 15:50:01

Bluetooth Terminal Connected - 15:50:01

Bluetooth Terminal Connected - 15:50:01

PIR
PIR
PIR - Undetected..

PIR
PIR
PIR - Undetected..

PIR
PIR
PIR - Detected!
```

[Figure 4-34] The Screen of Raspberry Pi



[Figure 4-35] The Screen of Android

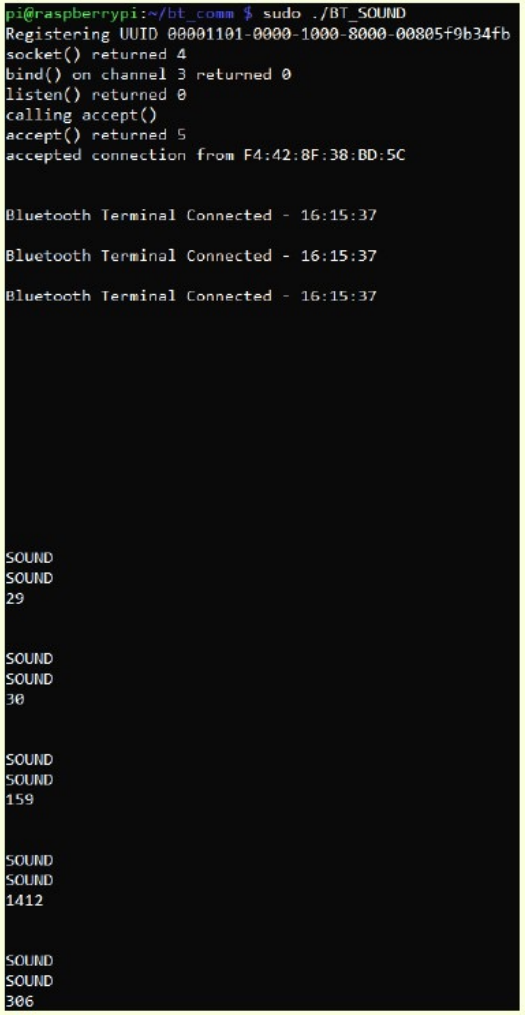
Sound Sensor

```c
#include "bt_master.h"
#include <unistd.h>
#include <wiringPi.h>

#define SPI_CH 0
#define ADC_CH 2
#define ADC_CS 29
#define SPI_SPEED 500000

int main()
{
        int client = init_server();
        int adcValue = 0;
        char *recv_message;
        unsigned char buf[3];

        if(wiringPiSetup() == -1) return 1;
        if(wiringPiSPISetup() == -1) return -1;

        pinMode(ADC_CS,OUTPUT);

```

```
22          while(1) {
23
24          recv_message = read_server(client);
25
26          if ( recv_message == NULL ){
27              printf("client disconnected\n");
28              break;
29          }
30          if(strcmp(recv_message,"SOUND") == 0){
31              buf[0] = 0x06 | ((ADC_CH & 0x04)>>2);
32              buf[1] = ((ADC_CH & 0x03)<<6);
33              buf[2] = 0x00;
34              digitalWrite(ADC_CS,0);
35              wiringPiSPIDataRW(SPI_CH,buf,3);
36              buf[1] = 0x0F & buf[1];
37              adcValue = (buf[1] << 8) | buf[2];
38              digitalWrite(ADC_CS,1);
39              sprintf(recv_message, "%d\n", adcValue);
40          }
41          write_server(client, recv_message);
42      }
43  }
```

```
pi@raspberrypi:~/bt_comm $ gcc -o BT_SOUND BT_SOUND.c -lwiringPi -lbluetooth
```

[Figure 4-37] Compiling BT_SOUND.c



[Figure 4-38] The Screen of Raspberry Pi

[Figure 4-39] The Screen of Android

DC Motor

```c
#include "bt_master.h"
#include <unistd.h>
#include <wiringPi.h>

#define PIN 26

int main()
{
        int client = init_server();

        char *recv_message;

        if(wiringPiSetup() == -1) return 1;

        pinMode(PIN,OUTPUT);

        while(1){

        recv_message = read_server(client);
        if ( recv_message == NULL ){
            printf("client disconnected\n");
            break;
        }
        if(strcmp(recv_message,"DCMON") == 0){
           digitalWrite(PIN,HIGH);
           strcpy(recv_message, "DC Motor ON!\n");
        }else if(strcmp(recv_message, "DCMOFF") == 0){
           digitalWrite(PIN,LOW);
           strcpy(recv_message, "DC Motor OFF!\n");
        }
        write_server(client, recv_message);
    }
}
```

```
pi@raspberrypi:~/bt_comm $ gcc -o BT_DCM BT_DCM.c -lwiringPi -lbluetooth
```

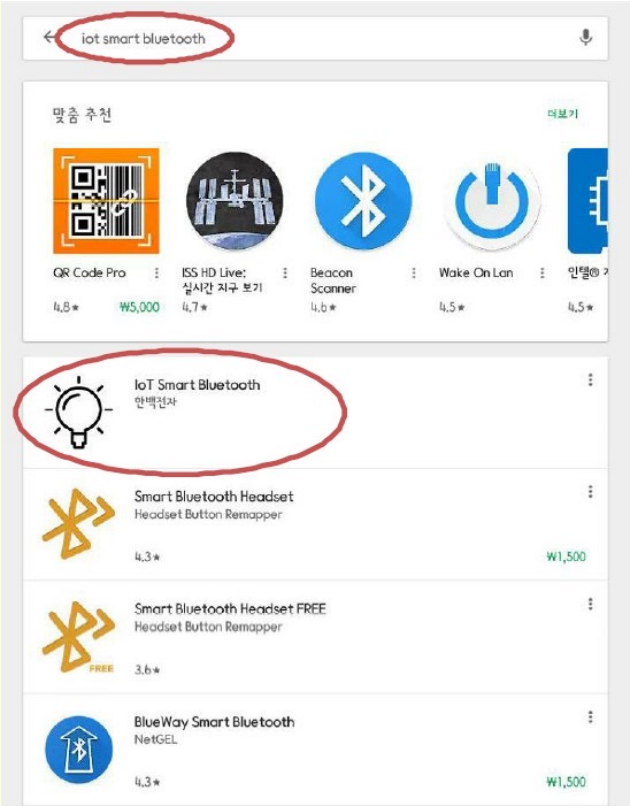
[Figure 4-45] Compiling BT_DCM.c

```
pi@raspberrypi:~/bt_comm $ sudo ./BT_DCM
Registering UUID 00001101-0000-1000-8000-00805f9b34fb
socket() returned 4
bind() on channel 3 returned 0
listen() returned 0
calling accept()
accept() returned 5
accepted connection from F4:42:8F:38:BD:5C


Bluetooth Terminal Connected - 16:54:45


Bluetooth Terminal Connected - 16:54:45


Bluetooth Terminal Connected - 16:54:45


DCMON
DCMON
DC Motor ON!


DCMOFF
DCMOFF
DC Motor OFF!
```

[Figure 4-46] The Screen of Raspberry Pi

[Figure 4-47] The Screen of Android

Bluetooth IoT Syste

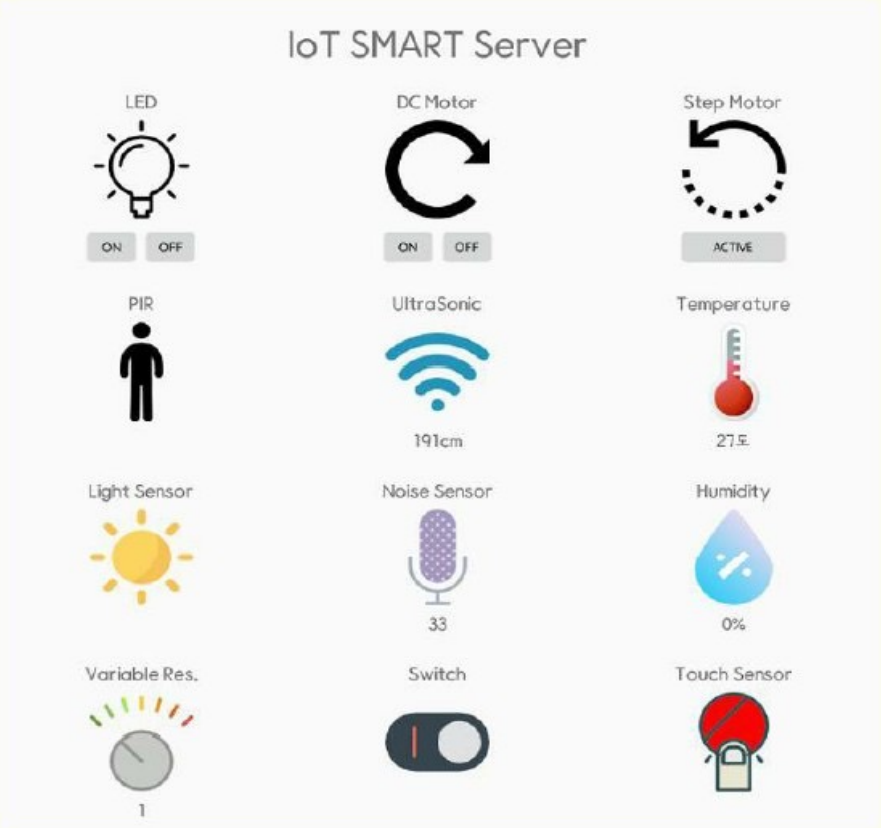


```
pi@raspberrypi:~/bt_comm $ gcc -o BT_IoT BT_IoT.c -lwiringPi -lbluetooth -lm
```

[Figure 4-152] Compiling BT_IoT.c

```
TEMP
TEMP
27

HUMI
HUMI
-1

SOUND
SOUND
30

VR
VR
1

MERC
MERC
Y

TILT
TILT
Y

FLAME
FLAME
N

REED
REED
N
```

[Figure 4-153] The Screen of Raspberry Pi



[Figure 4-154] The Screen of Android

Any Questions!