# Introduction

* Bluetooth Low Energy is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range.
* In contrast to [Classic Bluetooth](https://developer.android.com/guide/topics/connectivity/bluetooth), Bluetooth Low Energy (BLE) is designed to provide significantly lower power consumption.
* This technology is used in making alternative and innovative devices for healthcare, fitness, beacons, security and home entertainment industries.

# Terms and Concepts

* **Generic Attribute Profile (GATT)**—The GATT profile is a general specification for sending and receiving short pieces of data known as "attributes" over a BLE link. All current Low Energy application profiles are based on GATT.
* **Attribute Protocol (ATT)**—GATT is built on top of the Attribute Protocol (ATT). ATT is optimized to run on BLE devices. Each attribute is uniquely identified by a Universally Unique Identifier (UUID), which is a standardized 128-bit format for a string ID used to uniquely identify information.
* **Attributes –** they are the data elements that consists of two fields i.e., a tag and a value. Value can be Boolean, string, integer or other attributes.
* **Service**—A service is a collection of characteristics.
* **Characteristic**—A characteristic contains a single value and 0-n descriptors that describe the characteristic's value. A characteristic can be thought of as a type, analogous to a class.
* **Descriptor**—Descriptors are defined attributes that describe a characteristic value.

Diagram, table

Description automatically generated

Fig 2.1 GATT profile for a peripheral

# 3. Different Functions of a BLE

## 3.1 Broadcaster

* To transmit advertising packets
* Consists of only transmitter, may or may not have a Receiver

## 3.2 Observer

* Must have the receiver, but may not have a transmitter
* To receive the advertising packets

## 3.3 Central

* Listens to the advertising packet and interacts to make a connection
* To negotiate making a connection over the L2CAP channel
* It must have both the receiver and the transmitter

## 3.4 Peripheral

* It must have both the receiver and the transmitter
* It can advertise as connectable, not connectable, scannable or non-scannable.

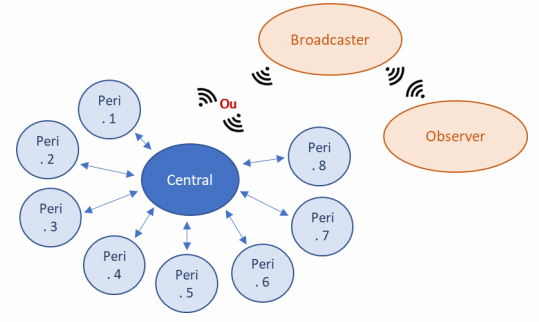


Fig 3.1 Different functions of a BLE

# Layered Architecture of BLE

* **PHY layer**(Physical layer). The PHY layer is used to specify the radio frequency band used by BLE, the modulation and demodulation methods and methods. [6]
* **LL layer**(Link Layer link layer). The LL layer is the core of the entire BLE protocol stack and is the difficulty and focus of the BLE protocol stack. There are many things to do in the LL layer, such as
  + Which RF channel to choose for communication,
  + How to identify the air packet,
  + At which point in time to send the packet,
  + How to ensure the integrity of the data,
  + How to receive the ACK,
  + How to retransmit and how to manage and control the link and so on.
* **HCI** is mainly used in the case where two chips implement the BLE protocol stack and is used to standardize communication protocols and communication commands between the two.
* **GAP layer**(Generic access profile).GAP simply makes some specifications and definitions for LL payload, so the functions that GAP can implement are extremely limited. GAP is currently used primarily for broadcasting, scanning and initiating connections.
* **L2CAP layer**(Logic link control and adaptation protocol). L2CAP needs to distinguish between encrypted channel and normal channel and manages the connection interval.
* **SMP**(Secure manager protocol). SMP is used to manage the encryption and security of BLE connections.
* **ATT**(Attribute protocol). In simple terms, the ATT layer is used to define data for user commands and command operations, such as reading a certain data or writing a certain data.
* **GATT**(Generic attribute profile). GATT is used to standardize the data content in the attribute and uses the concept of group to classify and manage the attribute.

Graphical user interface, application

Description automatically generated

Fig 4.1 Different protocol layers of BLE

# Link States of BLE

* **Advertisers**: Devices that transmit the advertising packets on the advertising channels are referred to as advertisers. [7]
* **Scanners**: Devices that receive advertising on the advertising channels without intention to connect to that are referred to as scanners.
* **Initiators**: Devices that need to form a connection to another device listens for connectable advertising packets.
* **Central/Peripheral**: Once a connection is established, the initiator become the Central device, the advertising device becomes the Peripheral device.

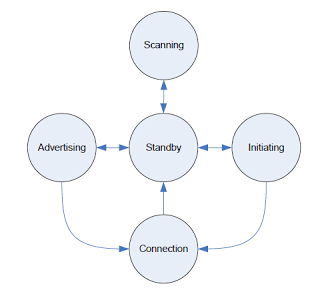


Fig 5.1 Different link states

**Advertisement Interval:**

* When a BLE peripheral device is in advertising mode, advertising packets are sent periodically on each advertising channel. The time interval between packet set has both a fixed interval (advert delay) and a random delay. The interval is specified between the set of 3 packets.
* The fixed interval can be set from 20ms to 10.24 seconds, in steps of 0.625ms.
* The random delay is a pseudo-random value from 0ms to 10ms that is automatically added. This randomness helps reduce the possibility of collisions between advertisements of different devices.

**Scan Parameters**

* **Scan Interval** between the start of two consecutive **scan** windows.
* The duration in which the Link Layer **scans** on one channel is the **Scan Window** (is 10ms to 10.24s).
* **Scan** **Duration** is the duration in which the device stays in the **scanning** state.

# Connections

* The Peripheral needs to start Advertising and send out Connectable Advertisement packets.
* The Central device needs to be Scanning for Advertisements while the Peripheral is Advertising.
* If the Central happens to be listening on an Advertising Channel that the Peripheral is Advertising on.
* The Central then sends a Connection Request packet.
* The peripheral always listens for a short interval on the same Advertising Channel after it sends out the Advertising packet.

**Connection Event [8]**

* During a **Connection Event**, the Master and Slave alternate sending data packets to each other until neither side has data to send.
* A Connection Event contains at least one packet sent by the Master.
* The Slave always sends a packet back if it received a packet from the Master.
* If the Master does not receive a packet back from the Slave, the Master will close the Connection Event — it resumes sending packets at the next Connection Event.
* The Connection Event can be closed by either side.
* The starting points of consecutive Connection Events are spaced by a period called the Connection Interval.

**Connection Parameters**

* **Connection Interval**: the interval at which two connected BLE devices wake up the radio and exchange data (at each Connection Event).
* **Slave Latency**: this value allows the Peripheral to skip a few consecutive Connection Events and not listen to the Central at these Connection Events without compromising the Connection. For example, a Slave Latency of **3** allows a Slave to skip 3 Connection Events and a value of **0** means that the Slave must send data to the Master at every Connection Event.
* **Supervision Timeout**: the maximum time between two received data packets before the Connection is considered lost.

# D-bus

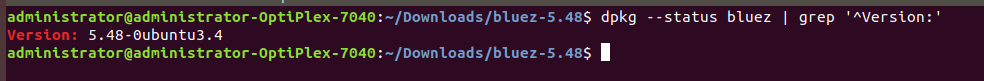
* Mechanism for IPC. Is a layered architecture. At the lowest level it has D-bus wire protocol and at the highest level it is D-bus daemon. [9]
* Messages is the unit of data transfer and are discrete in nature. There are four different types of messages,
  + **SIGNAL –** broadcast message by a process and is received by interested processes.
  + **METHOD\_CALL –** is a message requested by sender for a particular operation on an object of the receiver.
  + **METHOD\_RETURN –** is a response back to the sender by the receiver after the completion of the operation by the receiver.
  + **ERROR –** the receiver can respond with an error message if an error occurs.
* A **message bus** is a daemon process which routes messages between other processes.
* The D-Bus daemon configuration files are located in the /usr/share/dbus-1 directory.
* When an application connects with the D-Bus daemon, it is assigned a unique connection name. A unique connection name starts with the colon character ":
* A **service** is a daemon process that provides some utility in the system
* An **object** is an entity in a process, which does some work.

# BlueZ

* The BlueZ package contains the Bluetooth protocol stack for Linux.[10]
* bccmd - is used to issue BlueCore commands to Cambridge Silicon Radio devices.
* **Bluemoon -** is a Bluemoon configuration utility.
* **Bluetoothctl -** is the interactive Bluetooth control program
* **Bluetoothd -** is the Bluetooth daemon.
* **l2ping -** is used to send a L2CAP echo request to the Bluetooth MAC address given in dotted hex notation.

# Bluez Compilation for native ubuntu system

* To check the version of ubuntu kernel use the command: uname -v [11]
* To see the version of bluez already installed by default: dpkg --status bluez | grep '^Version:'



* To uninstall the current version of the bluez: sudo apt-get - -purge remove bluez
* Download or git clone the same version of bluez from bluez.org
* Run the configure command and generate the make file

./configure --prefix=/usr \

--sysconfdir=/etc \

--localstatedir=/var \

--enable-library \

--disable-systemd

* To check the versions of previously installed packages use the commands,

ldd - -version (Glib version)

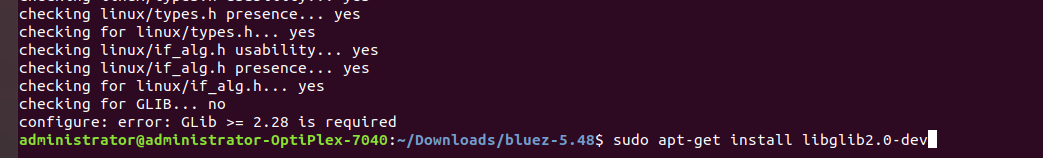
dbus-deamon - - version (dbus version)

* If there is a error while using the configure command, install the required package
  + To search for the presence of the library

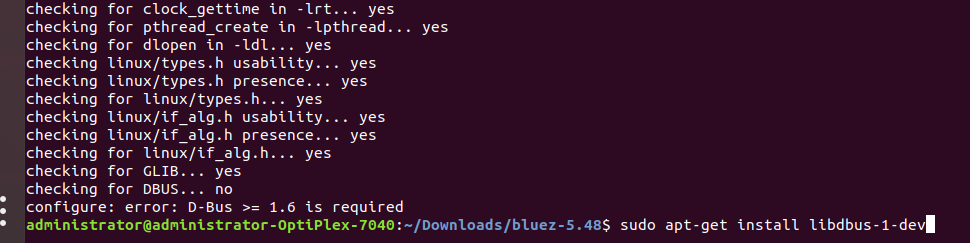
apt-cache search libglib (from the list install -dev)

* + To install use the following commands,

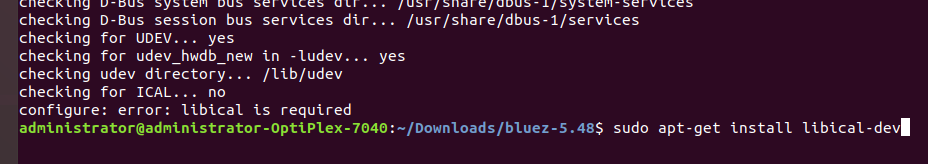
sudo apt-get install libglib2.0-dev



sudo apt-get install libdbus-1-dev

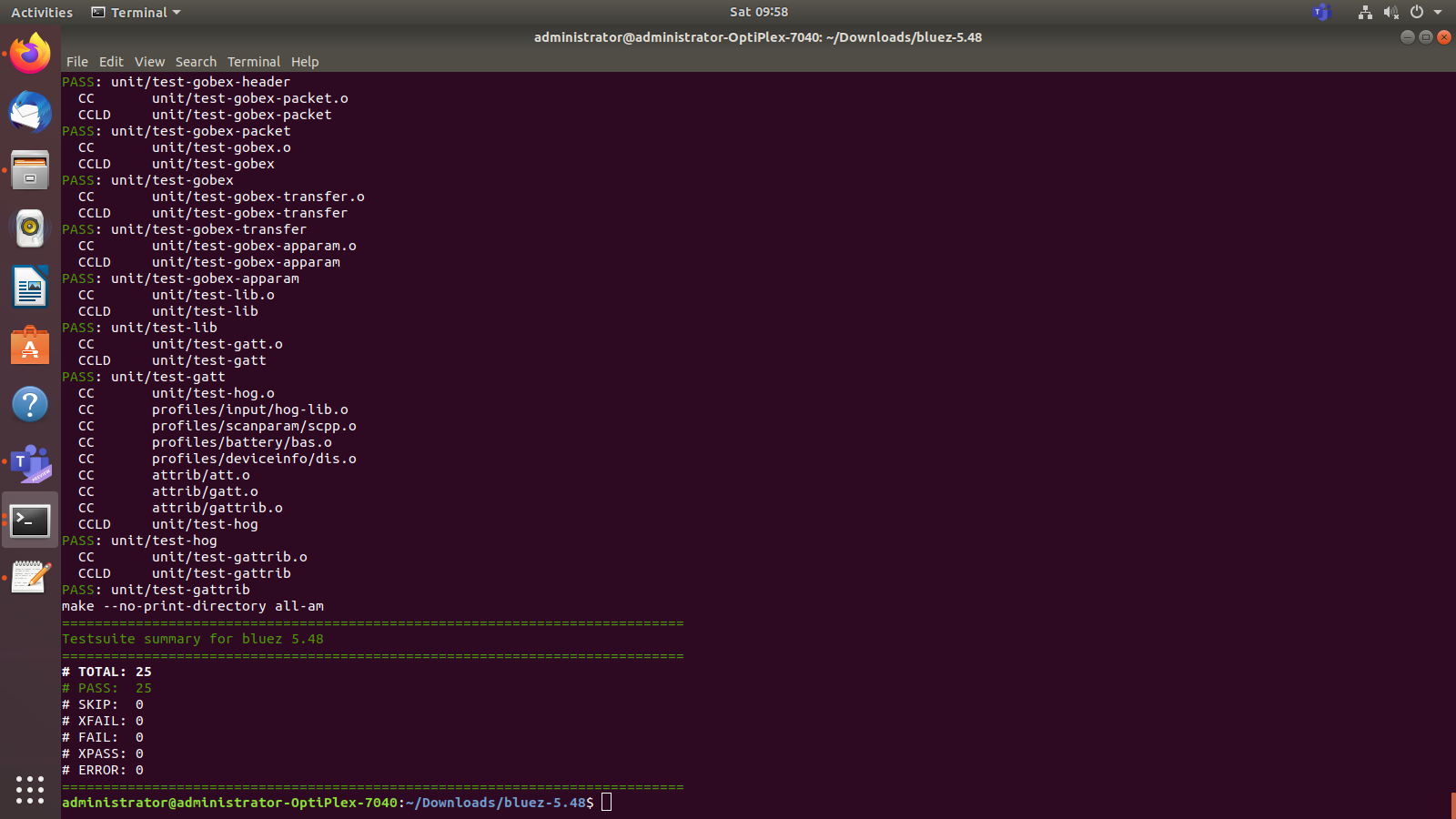


sudo apt-get install libical-dev



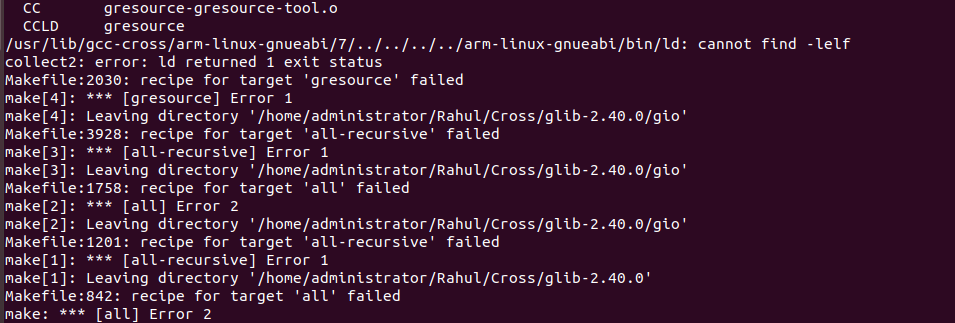
sudo apt-get install libreadline-dev

* Now use the command make and make install to install the bluez. Use make check to see if the package is installed correctly.

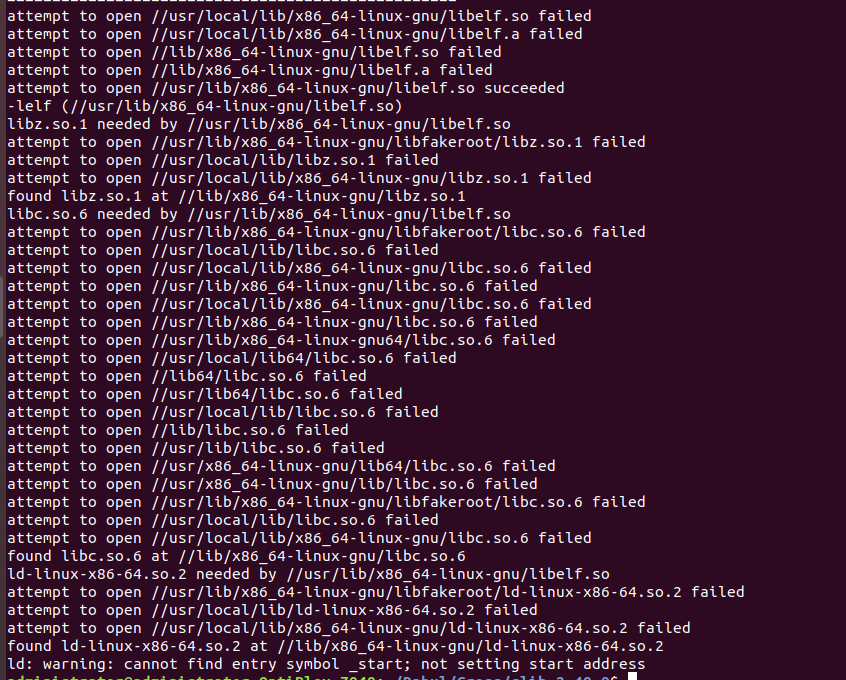


# Cross Compilation of Bluez for arm64

* 1. Zlib
* To download, cross-compile and install, use the following commands, [12]
  + wget http://zlib.net/zlib-1.2.8.tar.gz
  + tar -xzf zlib-1.2.8.tar.gz
  + cd zlib-1.2.8/
  + ./configure --prefix=/usr/arm-linux-gnueabi
* Then in the Makefile make the following changes
  + CC=arm-linux-gnueabi-gcc
  + LDSHARED=arm-linux-gnueabi-gcc -shared -Wl,-soname,libz.so.1,--version-script,zlib.map
  + CPP=arm-linux-gnueabi-gcc -E
  + AR=arm-linux-gnueabi-ar
  + RANLIB=arm-linux-gnueabi-ranlib
* Then use the command make and make install
  1. Libffi
* To download and install, use the following commands,
* wget ftp://sourceware.org/pub/libffi/libffi-3.0.13.tar.gz
* tar -xzf libffi-3.0.13.tar.gz
* cd libffi-3.0.13/
* ./configure --host=arm-linux-gnueabi --prefix=/usr/arm-linux-gnueabi
* make
* make install
  1. Glib
* To download and install, use the following commands,
  + wget http://ftp.gnome.org/pub/gnome/sources/glib/2.40/glib-2.40.0.tar.xz
  + tar -xJf glib-2.40.0.tar.xz
  + cd glib-2.40.0
  + ./configure --host=arm-linux-gnueabi --prefix=/usr/arm-linux-gnueabi PKG\_CONFIG\_PATH=/usr/arm-linux-gnueabi/lib/pkgconfig glib\_cv\_stack\_grows=no glib\_cv\_uscore=yes ac\_cv\_func\_posix\_getpwuid\_r=yes ac\_cv\_func\_posix\_getgrgid\_r=yes
  + make
  + make install
  1. Expat XML
* To download and install, use the following commands,
  + wget http://sourceforge.net/projects/expat/files/expat/2.1.0/expat-2.1.0.tar.gz
  + tar -xzf expat-2.1.0.tar.gz
  + cd expat-2.1.0/
  + ./configure --host=arm-linux-gnueabi --prefix=/usr/arm-linux-gnueabi
  + make
  + make install
  1. D-Bus
* To download and install, use the following commands,
  + wget http://dbus.freedesktop.org/releases/dbus/dbus-1.8.0.tar.gz
  + tar -xzf dbus-1.8.0.tar.gz
  + cd dbus-1.8.0/
  + ./configure --host=arm-linux-gnueabi --prefix=/usr/arm-linux-gnueabi
  + Make
* This was incomplete as the following logs were present,



This was the error that we got when we run the make command. We used the verbose to check what the possible error was but it gave an output as follows, and we couldn’t complete the cross compilation. (ld -lelf - -verbose).



# References:

1. <https://elainnovation.com/what-is-ble.html#:~:text=growing%20industry%20nowadays.-,Features,half%20as%20much%20as%20Bluetooth>.
2. <https://blog.beaconstac.com/2018/08/ble-made-simple-a-complete-guide-to-ble-bluetooth-beacons/>
3. <https://developer.android.com/guide/topics/connectivity/bluetooth-le#setup>
4. <https://www.tutorialspoint.com/the-bluetooth-protocol-architecture#:~:text=Bluetooth%20network%20technology%20connects%20mobile,model%20or%20TCP%2FIP%20model>
5. <https://www.ques10.com/p/2704/explain-park-hold-and-sniff-mode-of-operations-in-/#:~:text=PARKED%20devices%20have%20given%20up,mode%20with%20lowest%20duty%20cycle>.
6. <https://www.programmersought.com/article/3353104625/>
7. <https://medium.com/@zpcat/how-bluetooth-le-works-link-layer-b18475250259#:~:text=The%20Link%20Layer%20of%20BLE,Connection(Master%2C%20Slave)>.
8. <https://www.novelbits.io/basics-bluetooth-low-energy/>
9. <https://www.softprayog.in/programming/d-bus-tutorial>
10. <http://www.linuxfromscratch.org/blfs/view/svn/general/bluez.html>
11. <http://www.linuxfromscratch.org/blfs/view/8.2/general/bluez.html>
12. <https://wiki.beyondlogic.org/index.php?title=Cross_Compiling_BlueZ_Bluetooth_tools_for_ARM>