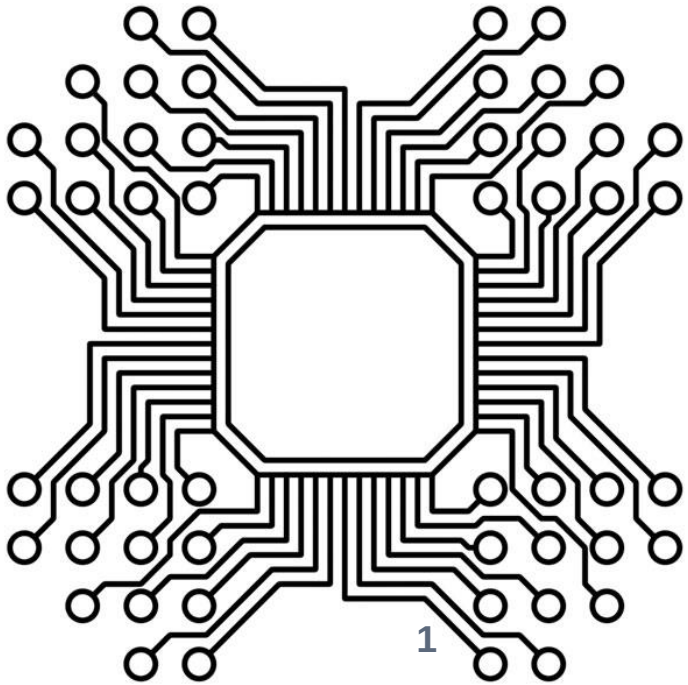
WIRELESS COMMUNICATION

BLUETOOTH LOW ENERGY

Lecturer: Dr. Bui Ha Duc

Dept. of Mechatronics

Email: ducbh@hcmute.edu.vn

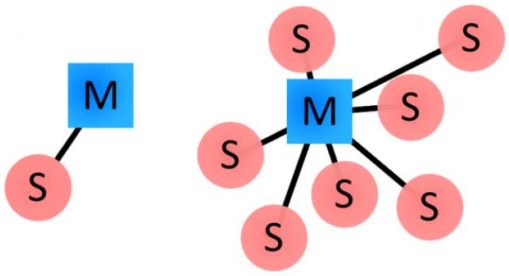


Bluetooth

Feature	Bluetooth Classic	Bluetooth 4.x	Bluetooth 5
Radio Frequency (MHz)	2400 to 2483.5	2400 to 2483.5	2400 to 2483.5
Distance/Range (m)	Up to 100	Up to 100	Up to 200
Medium Access Technique	Frequency Hopping	Frequency Hopping	Frequency Hopping
Nominal Data Rate (Mbps)	1–3	1	2
Latency (ms)	<100	<6	<3
Network Topology	Piconet, Scatternet	Star-bus	Star-bus, Mesh before the end of the 2017
Multi-hop Solution	Scatternet	Not currently supported	Not currently supported
Profile Concept	Yes	Yes	Yes
Nodes/Active Slaves	7	Unlimited	Unlimited
Message Size (bytes)	Up to 358	31	255
Government Regulation	Worldwide	Worldwide	Worldwide
Certification Body	Bluetooth SIG	Bluetooth SIG	Bluetooth SIG

Bluetooth Classic

- Bluetooth classic is like a RF version of serial communication
- Bluetooth focus on enhancing data rate
  - V1.2: 1 Mbps
  - V2.1: 3 Mbps in theory – around 2.1 Mbps in practice
  - V3.0: up to 24Mbps
    - Bluetooth is use to establish and manage a connection
    - Data is transmitted over Wifi connection



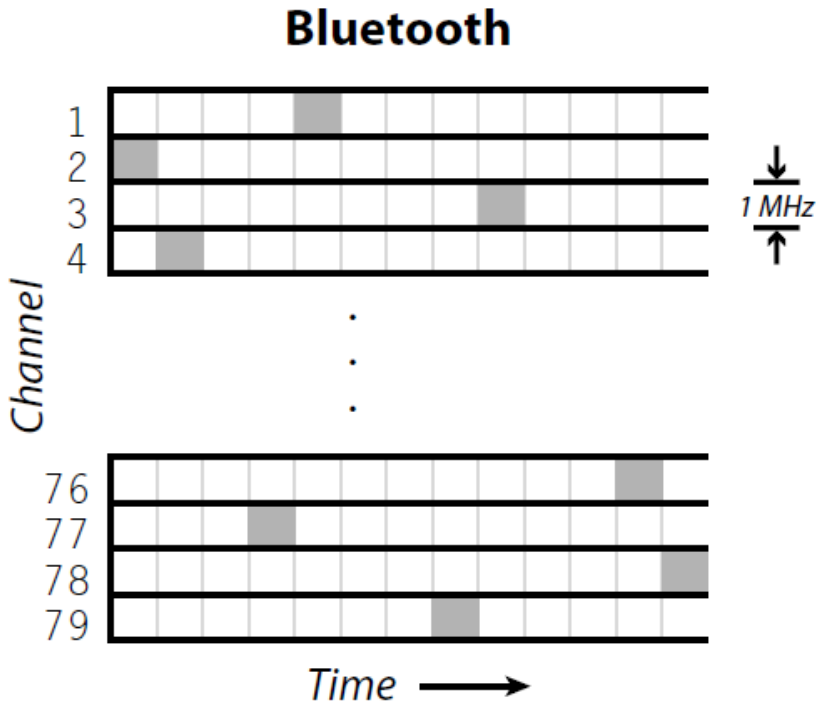
Connection Procedure

1. **Device Inquiry**: search for nearby devices
2. **Paging :** Choose device to connect, need device name/MAC
3. **Form Connection**:
   - Choose transport protocol
   - Choose port number and matching record
   - Mode of operation

Bluetooth Channel Hopping

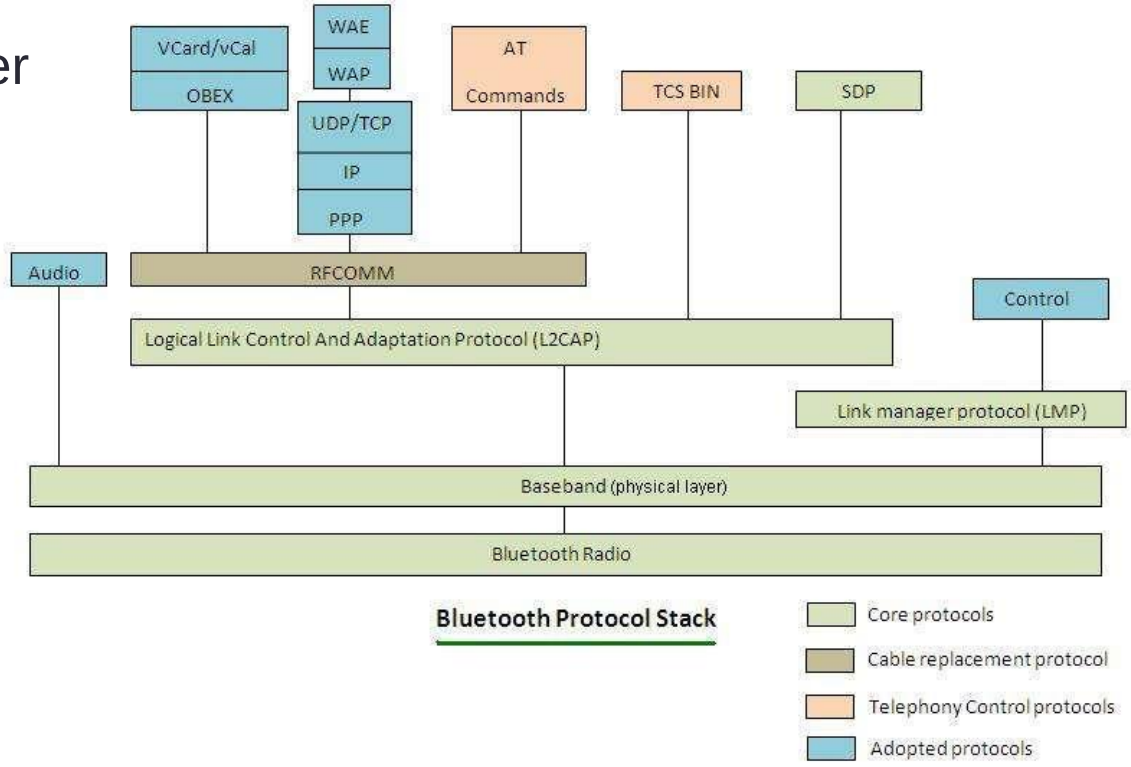
- Bluetooth has 79 channels, each channel is 1MHz wide, from 2.402 GHz to 2.480 GHz.
- Bluetooth devices never stay on the same channel
- Change channel 1600 times per second

-> unaffected by other electronic radiowaves



Bluetooth protocols

- **Bluetooth protocol layers:**
  - Core layer
  - Cable replacement layer
  - Telephony control layer
  - Adopted protocol



https://infotainmenttechnology.wordpress.com/2017/01/27/bluetooth-protocol-stacklayers/

Bluetooth protocols

- **Core layer:**
  - **Radio**: air interface, frequency bands, frequency hopping specifications, modulation technique
  - **Baseband**: addressing scheme, packet frame format , timing and power control algorithms
  - **Link Manager**: establish and maintain link between bluetooth devices
  - **Logical link control and adaptation protocol (L2CAP)**
  - **Service discovery protocol**: Service related queries including device information

Bluetooth protocols

- **Cable replacement protocol**
  - **RFCOMM**: virtual serial port, transport of binary digital data bits
- **Telephony Control Protocols (TCP)**: set up and control speech and data calls between Bluetooth devices
- **Adopted protocols:**
  - protocols are already defined by other standard bodies
  - incorporate without any change in the bluetooth protocol

Bluetooth protocols

Bluetooth transport protocols

- **RFCOMM protocol**
  - point-to-point connection
  - reliably exchange streams of data
  - RFCOMM allows 30 ports

- **L2CAP**
  - connection-oriented protocol
  - can be configured for varying levels of reliability

https://people.csail.mit.edu/albert/bluez-intro/

Bluetooth programming in C

- https://people.csail.mit.edu/albert/bluez-intro/c404.html
- www.drdobbs.com/mobile/using-bluetooth/232500828?pgno=1

Bluetooth Classic Modules

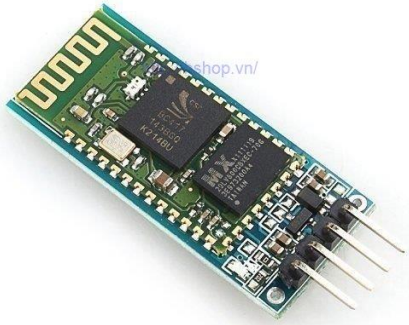
Main Features:

- Bluetooth V2.0
- 3Mbps Modulation
- Adaptive Frequency Hopping
- UART interface
  - Data bits:8, Stop bit:1,Parity:No parity
  - HC-05: default baudrate 38400, pin: 1234
  - HC-06: default baudrate 9600, pass: 1234

- Communicate with **AT command**

Master
HC-05

Slave
HC-06



Connect to MCU

Bluetooth module pinout

- Vcc 3.3V – 5V
- GND
- RXD -> TX
- TXD -> RX

3.3V

- Key -> GPIO
- State -> GPIO

AT command

- AT – Attention
- AT commands are **instructions** used to control a device
- Every command line starts with "AT" or "at“

- **<CR>** Carriage return character (in C: “**/r**”)
- **<LF>** Linefeed character (in C: “**/n**”)

AT command

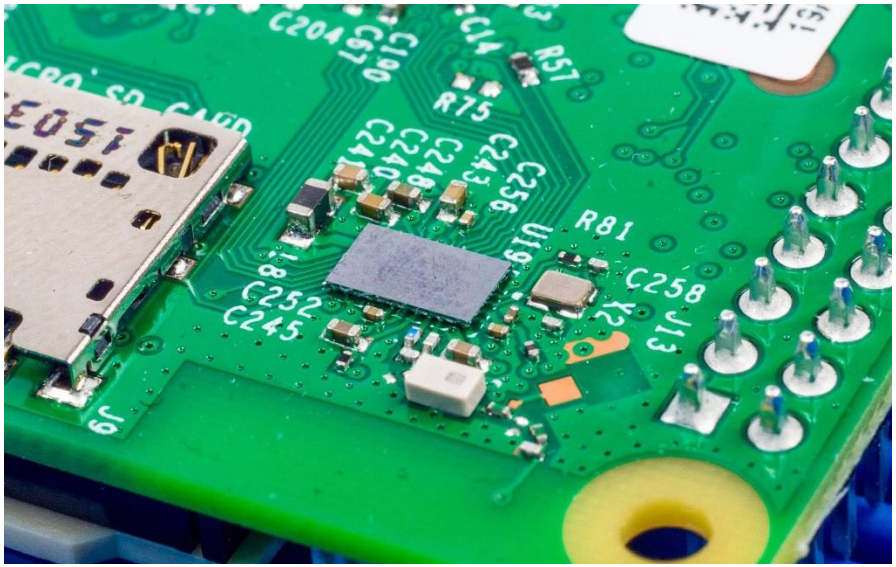
http://www.martyncurrey.com/hc-05-fc-114-and-hc-06-fc-114-part-2-basic-at-commands/

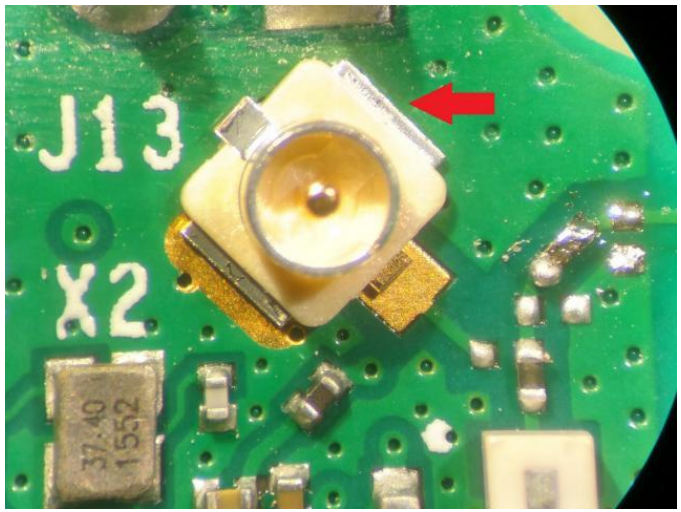
Basic AT commands

Command	Return	Parameter	Description
AT	OK	None	Test
AT+VERSION?	+VERSION:<Param> OK	Param: Version number	Get the soft version
AT+ORGL	OK	None	Restore default status
AT+ADDR?	+ADDR: <Param> OK	Param: Bluetooth address	Get module Bluetooth address
AT+NAME=<Param>	OK	Param: Bluetooth device name	Set device's name
AT+NAME?	+NAME:<Param> OK	Param: Bluetooth device name	Inquire device's name
AT+ROLE=<Param>	OK	Param:0=Slave role; 1=Master role; 2=Slave-Loop role	Set module role
AT+ ROLE?	+ ROLE:<Param>	Param:0=Slave role; 1=Master role; 2=Slave-Loop role	Inquire module role
AT+UART=<Param>,<Param2>,<Param3>	OK	Param1: baud rate (bits/s); Param2: stop bit; Param3: parity bit	Set serial parameter
AT+ UART?	+UART=<Param>,<Param2>,<Param3> OK	Param1: baud rate (bits/s); Param2: stop bit; Param3: parity bit	Inquire serial parameter

Raspberry built-in wireless modules

- Built-in Wifi 802.11n
- Bluetooth Low Energy 4.1
- IoT ready
- Home cloud storage





Built-in Bluetooth

- BLE 4.1
- By default, it can’t be used for audio
- Support Bluetooth GATT (generic attribute profile) and Mesh
  - GATT

  https://www.bluetooth.com/specifications/gatt/generic-attributes-overview
  - Mesh

  https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-faq

Transfer file/message via bluetooth

- **Bluetooth librarie**
  - **Bluez** – official Linux Bluetooth protocol stack
    - Open source library which provides the Bluetooth protocol stack and the bluetoothctl utility.
    - Allows a Raspberry Pi to communicate with Bluetooth classic and Bluetooth low energy (LE) devices
    - Pre-installed in Raspberry OS
    - Website: www.bluez.org/
  - **Bluetooth** – This package provides all the plugins supported by Bluez Bluetooth stack.

Checking Bluetooth status

- Type **sudo systemctl status bluetooth** to check

```
pi@raspberrypi:~ $ sudo systemctl status bluetooth
● bluetooth.service - Bluetooth service
   Loaded: loaded (/lib/systemd/system/bluetooth.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2022-03-17 13:56:48 +07; 7h ago
     Docs: man:bluetoothd(8)
 Main PID: 426 (bluetoothd)
   Status: "Running"
    Tasks: 1 (limit: 729)
   CGroup: /system.slice/bluetooth.service
           └─426 /usr/lib/bluetooth/bluetoothd
```

- If the Bluetooth service status is not active, start it with:

  sudo systemctl enable bluetooth then

  sudo systemctl start bluetooth

- Stop the Bluetooth service with:

  sudo systemctl stop bluetooth

Setting up Bluetooth

Using Bluetooth tool in Terminal

- In terminal, type **sudo bluetoothctl** to open Bluetooth tool

```
pi@raspberrypi:~ $ sudo bluetoothctl
Agent registered
[bluetooth]#
```

- Type **power on** to make sure Bluetooth is on
- Type **agent on** to make sure bluetooth is running
- Type **scan on** to search for nearby Bluetooth devices

```
[bluetooth]# power on
Changing power on succeeded
[bluetooth]# agent on
Agent registered
[bluetooth]# scan on
Discovery started
```

Setting up Bluetooth

```
[CHG] Controller B8:27:EB:BC:98:29 Discovering: yes
[CHG] Device 58:51:00:00:1F:8F RSSI: -59
[CHG] Device F4:B7:E2:E7:7A:4F RSSI: -60
[NEW] Device 88:1F:A1:20:0A:33 OSTML0204141
[CHG] Device 50:56:A8:00:0E:EB RSSI: -61
[CHG] Device 58:51:00:00:1F:8F RSSI: -69
[CHG] Device 88:1F:A1:20:0A:33 RSSI: -85
[CHG] Device 50:56:A8:00:0E:EB RSSI: -90
[DEL] Device C0:EE:FB:26:95:C5 OnePlus One-spaceteam
[DEL] Device 50:56:A8:00:0E:EB Jon's Jolla
[DEL] Device F4:B7:E2:E7:7A:4F
[DEL] Device 58:51:00:00:1F:8F H163
[DEL] Device 88:1F:A1:20:0A:33 OSTML0204141
[NEW] Device F4:B7:E2:E7:7A:4F
[NEW] Device 58:51:00:00:1F:8F H163
[NEW] Device 50:56:A8:00:0E:EB Jon's Jolla
[CHG] Device 58:51:00:00:1F:8F RSSI: -71
[CHG] Device 58:51:00:00:1F:8F RSSI: -60
```

- Type **pair <MAC address>** to pair a device
  - `pair 01:02:03:04:05:06`
- Type **connect <MAC address>** to connect.
  - `connect 50:54:B4:45:00:EB`

Setting up Bluetooth

Bluetoothctl commands:

```
[bluetooth]# help
Menu main:
Available commands:
-------------------
advertise                                Advertise Options Submenu
scan                                     Scan Options Submenu
gatt                                     Generic Attribute Submenu
list                                     List available controllers
show [ctrl]                              Controller information
select <ctrl>                            Select default controller
devices                                  List available devices
paired-devices                           List paired devices
system-alias <name>                      Set controller alias
reset-alias                              Reset controller alias
power <on/off>                           Set controller power
pairable <on/off>                        Set controller pairable mode
discoverable <on/off>                    Set controller discoverable mode
agent <on/off/capability>                Enable/disable agent with given capability
default-agent                            Set agent as the default one
advertise <on/off/type>                  Enable/disable advertising with given type
set-alias <alias>                        Set device alias
scan <on/off>                            Scan for devices
info [dev]                               Device information
pair [dev]                               Pair with device
trust [dev]                              Trust device
untrust [dev]                            Untrust device
block [dev]                              Block device
unblock [dev]                            Unblock device
remove <dev>                             Remove device
connect <dev>                            Connect device
disconnect [dev]                         Disconnect device
menu <name>                              Select submenu
version                                  Display version
quit                                     Quit program
exit                                     Quit program
help                                     Display help about this program
export                                   Print evironment variables
```

Transfer message via bluetooth

- Establish connection between Raspberry Pi and devices via Bluetooth with

```
sudo rfcomm bind 0 <MAC address>
```

  it will create a device in /dev/rfcomm0

- To see received character/messages, enter following command

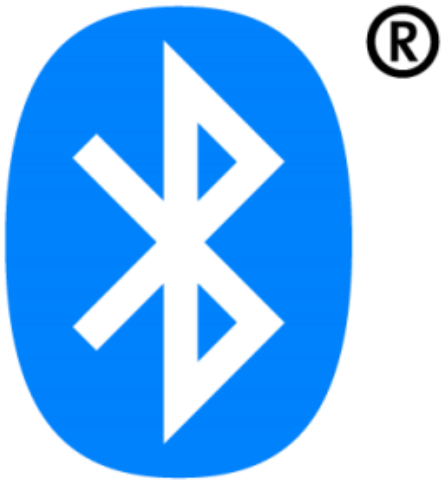
```
cat /dev/rfcomm0
```

- to send messages over Bluetooth, use following command

```
echo "Your massage" >/dev/rfcomm0
```

BLE Evolution

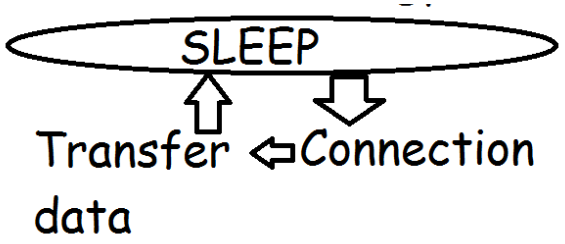
- 2010 - Bluetooth 4.0
- 2013 - Bluetooth 4.1
  - Concurrent Peripheral/Central
- 2014 - Bluetooth 4.2
  - LE Secure Connections
  - Data Length Extension
- 2016 - Bluetooth 5
  - 2 Mbps
  - Long Range
  - Advertising Extensions
  - 10 -> 20 dBm max TX power

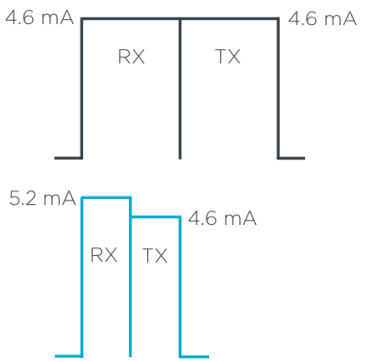


- 2017 - Bluetooth mesh Profile
- 2019 - Bluetooth 5.1
  - Direction Finding
- 2020 - Bluetooth 5.2
  - Isochronous channels
  - LE Power Control
  - Enhanced Attribute Protocol
- Soon – LE Audio

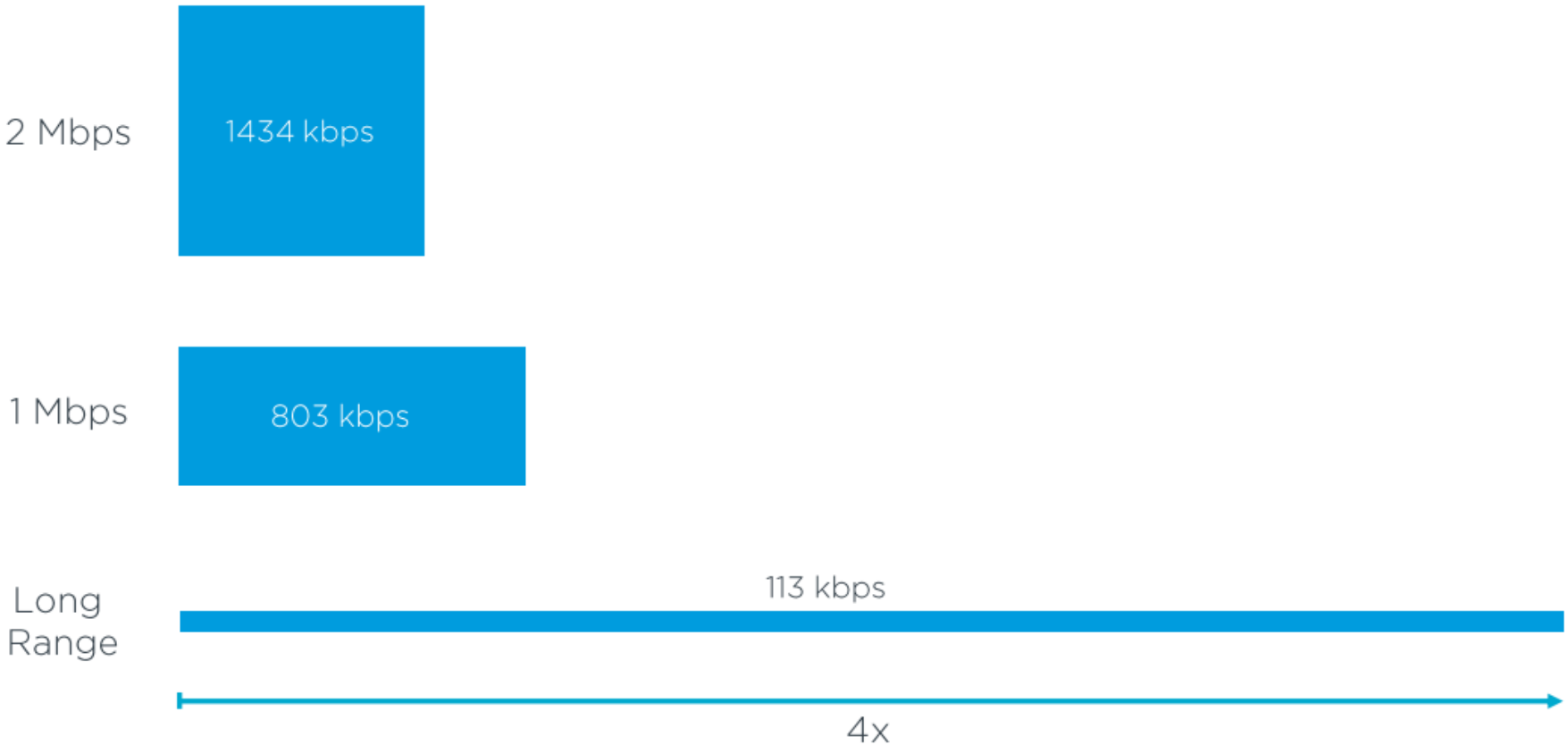
Bluetooth Low Energy

- BLE aims to operate at very low power
- BLE compromises in data rate
  - 1 Mbps in theory – 0.27 Mbps in practice
- BLE devices switch between sleep state and working state continuously to save energy
- BLE requires rapid connection, small package



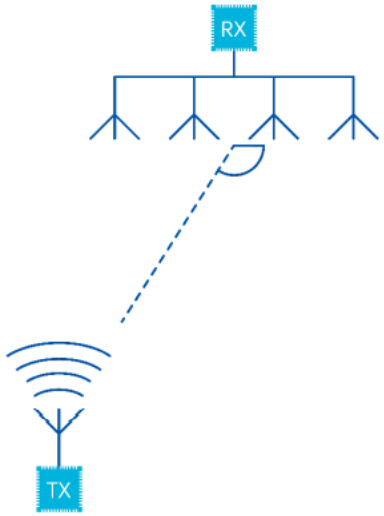


BLE Modes

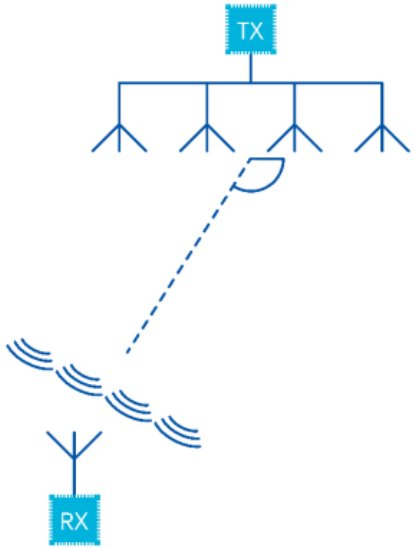


BLE Direction Finding

- Direction finding is feature of BLE 5.1
- Enables positioning solution
  - Old BLE rely on receive signal strength indicator (RSSI)
  - New BLE knows the actual direction of signal

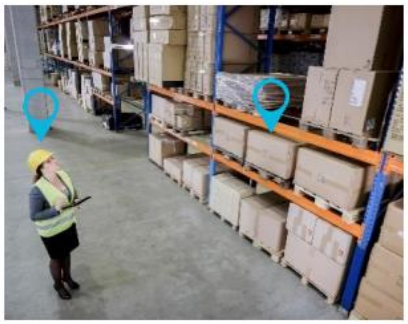
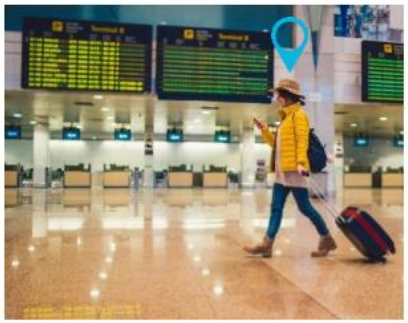


Angle of Arrival (AoA)



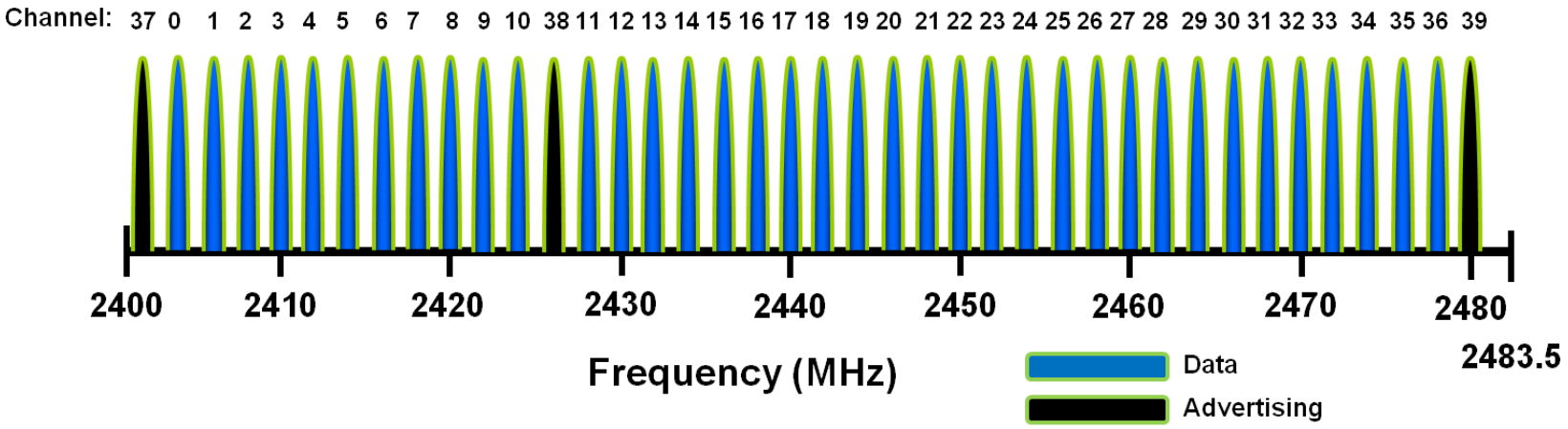
Angle of Departure (AoD)

BLE Direction Finding

Asset tracking	Wayfinding	Point of interest	Item finding
			
AoA	AoD	AoD	AoD
Multiple receivers at fixed locations	Multiple transmitters at fixed locations	Only relative direction needed	Only relative direction needed
Transmitter can be beacon or smart phone	Receiver typically a smart phone	Receiver typically a smart phone	Receiver typically a smart phone

How BLE works

- **Physical layer**
  - BLE can communicate over 40 channels from 2.4000 GHz to 2.4835 GHz.
  - 37 of these channels are used for connection data
  - the last three channels (37, 38, and 39) are used as advertising channels

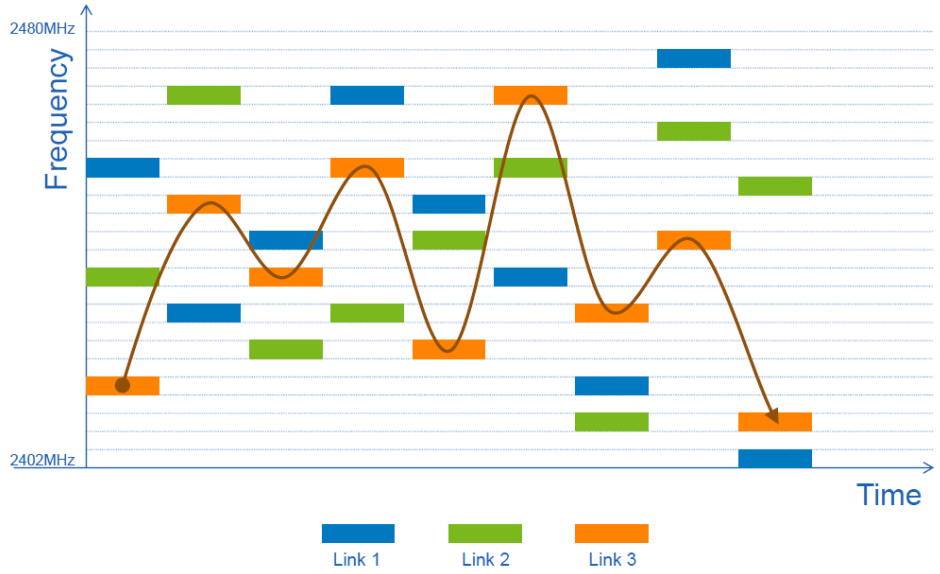


How BLE works

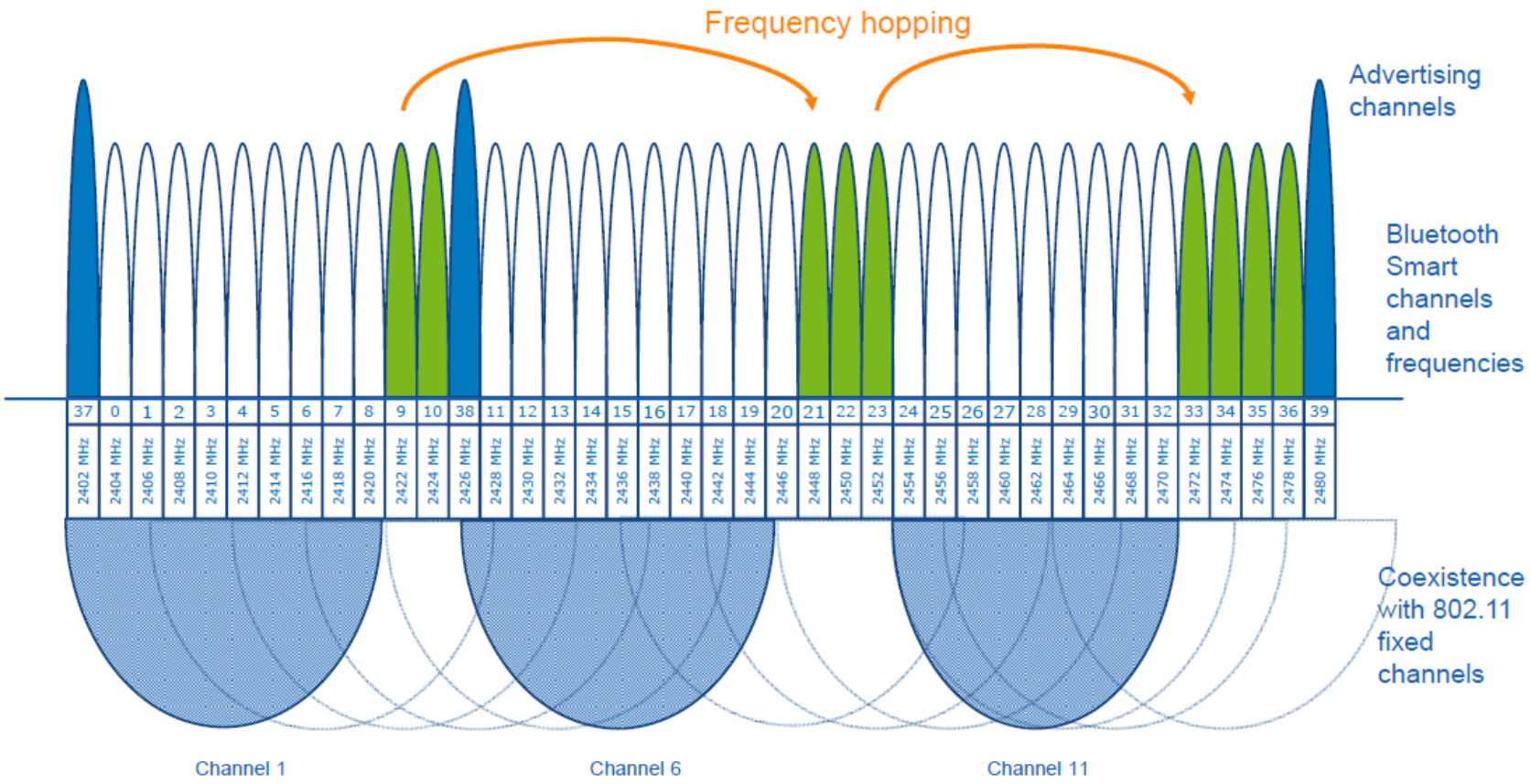
- BLE uses **frequency hopping spread spectrum** technique to send and receive data
- Frequency (channel) to use on the next connection event:

$$f_{n+1} = (f_n + hop) mod\, 37$$

  - *hop* is a value that can range from 5-16



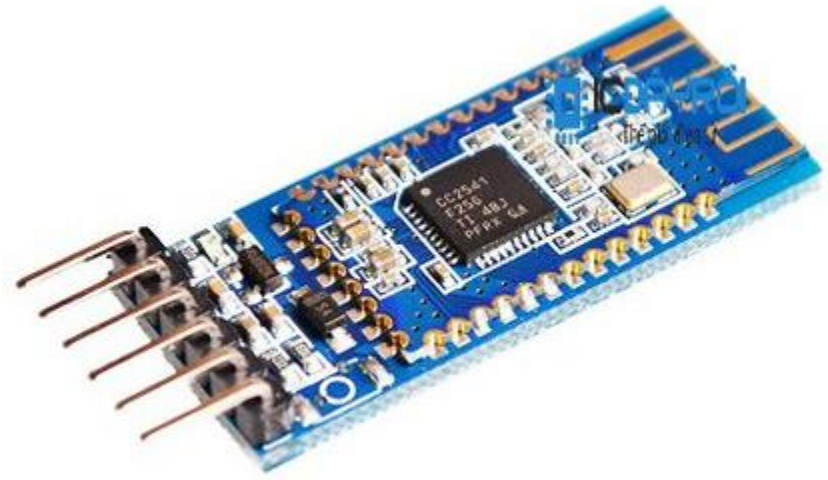
How BLE works



BLE MCU

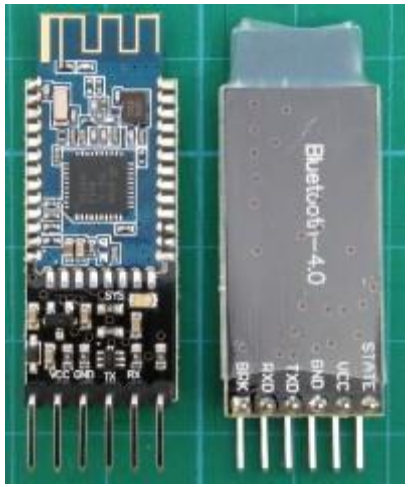
- Nordic Semiconductor
  - nRF series: nRF51822, nRF52832,…
  - ARM MCU

- Texas Instruments
  - CC25 series: CC2540, CC2541
  - CC2630/40/50
  - 8051 MCU



CC2451 Modules

Pin	Description
STATE	Connection status LOW when not connected. HIGH when connected
VCC	Power in. 3.6v to 6v
GND	Common ground
TXD	Serial UART transmit
RXD	Serial UART receive
BRK	Break pin. When there is an active connection, bringing the BRK pin LOW breaks the connection



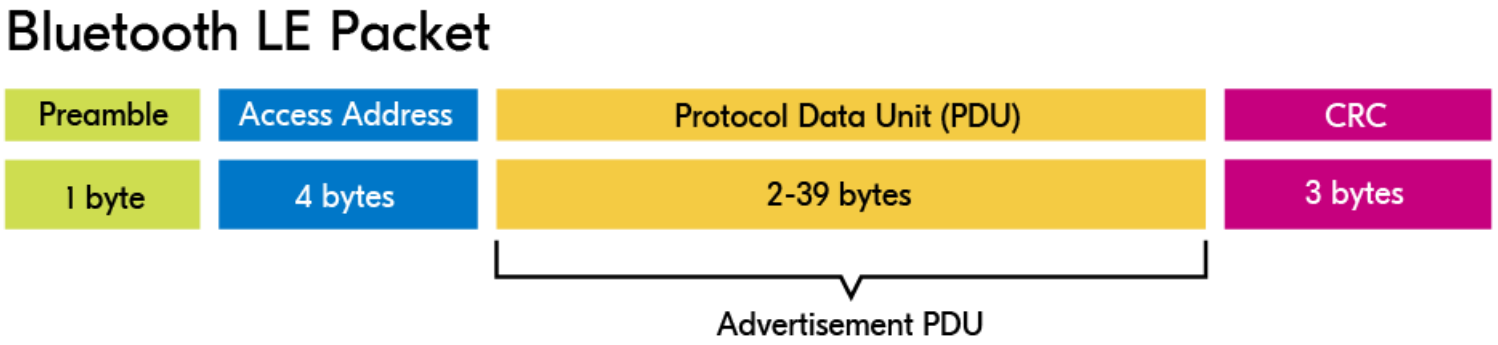
http://www.martyncurrey.com/hm-10-bluetooth-4ble-modules/

AT Commands

Command	Description
AT	Check if the command terminal work normally
AT+DEFAULT	Restore factory default
AT+BAUD	Get/Set baud rate
AT+RESET	Software reboot
AT+ROLE	Get/Set current role.
AT+DISC	Disconnect connection
AT+ADVEN	Broadcast switch
AT+ADVI	Broadcast interval
AT+NINTERVAL	Connection interval
AT+POWE	Get/Set RF transmit power
AT+NAME	Get/Set local device name
AT+LADDR	Get local bluetooth address
AT+VERSION	Get firmware, bluetooth, HCI and LMP version
AT+TYPE	Binding and pairing settings
AT+PIN	Get/Set pin code for pairing
AT+UUID	Get/Set system SERVER_UUID .
AT+CHAR	Get/Set system CHAR_UUID .
AT+INQ	Search from device
AT+RSLV	Read the scan list MAC address
AT+CONN	Connected scan list device
AT+CONA	Connection specified MAC
AT+BAND	Binding from device
AT+CLRBAND	Cancel binding
AT+GETDCN	Number of scanned list devices
AT+SLEEP	Sleep mode
AT+HELP	List all the commands

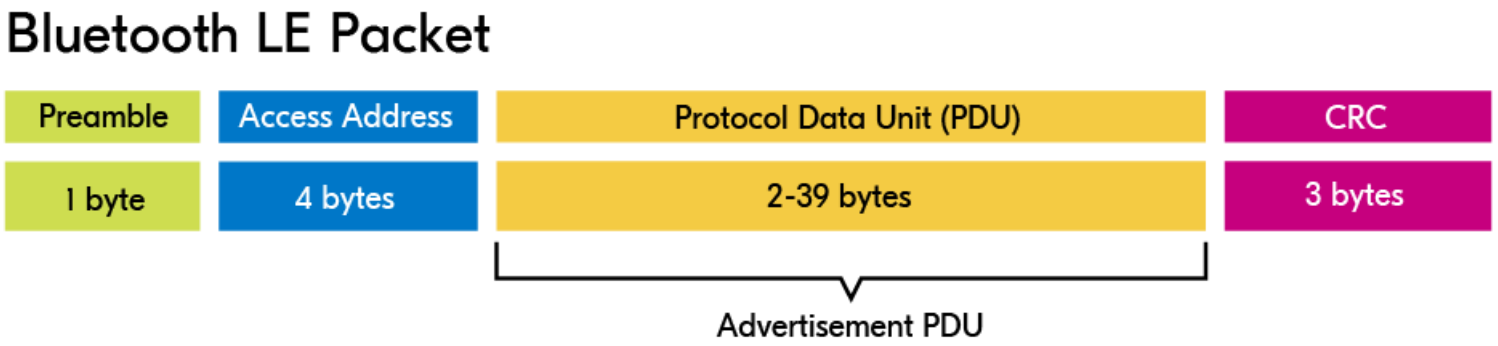
https://makersportal.com/blog/2019/10/14/bluetooth-module-with-arduino

BLE Packet Structure

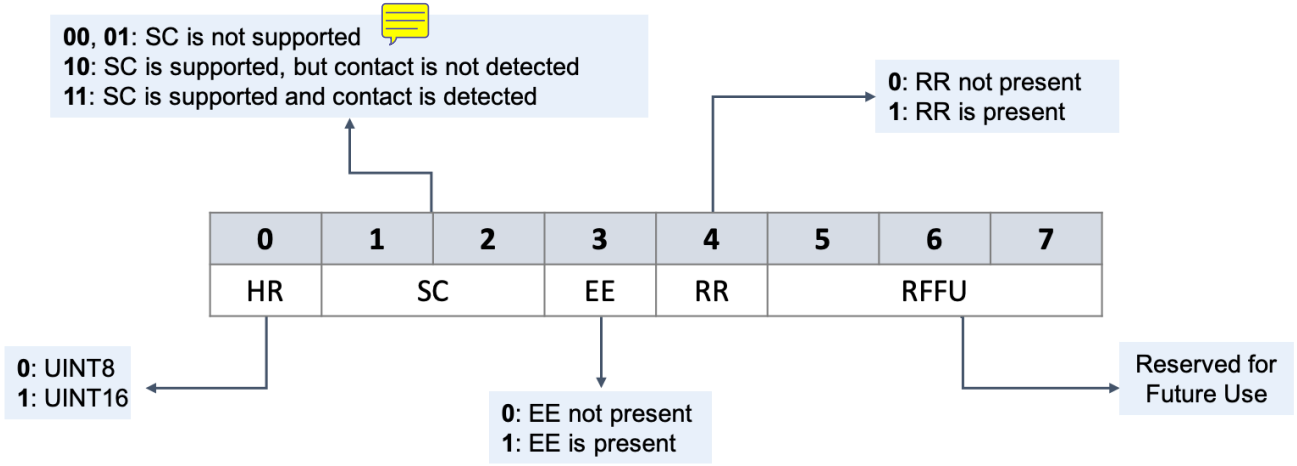


Preamble (1 byte):
This is a fixed 1-byte field used for synchronization between the receiver and transmitter. It helps the receiver recognize the start of the packet.

BLE Packet Structure

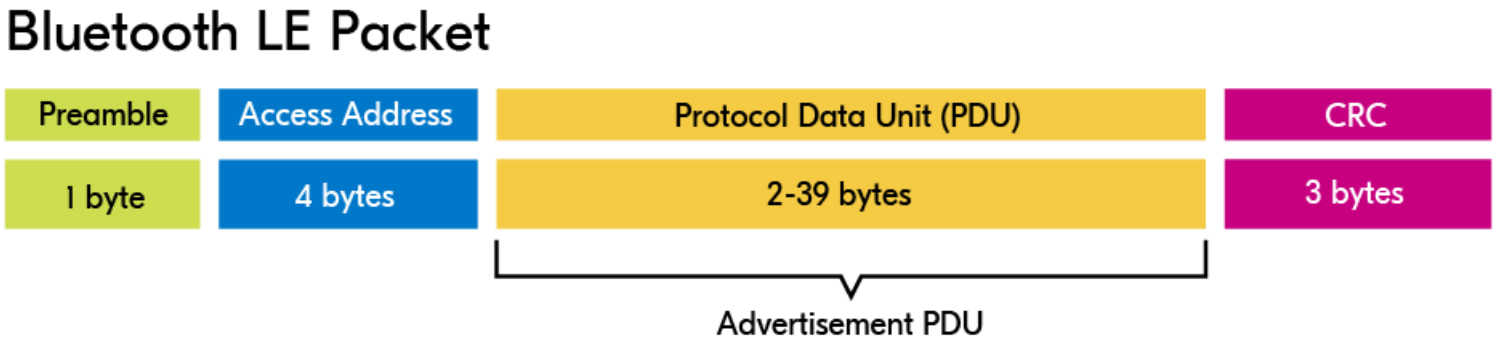


Preamble (1 byte):
This is a fixed 1-byte field used for synchronization between the receiver and transmitter. It helps the receiver recognize the start of the packet.



Heart Rate Service

BLE Packet Structure



Access Address (4 bytes):

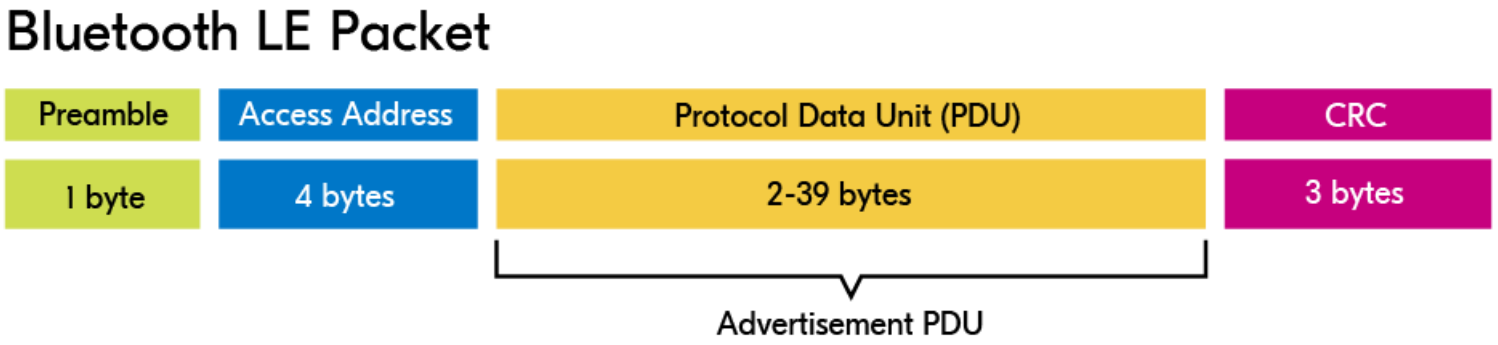
A unique 4-byte address used to identify the communication channel. It differentiates between different connections and is used in advertising and data channels.

Packet Header (2 bytes):

Advertisement Packets: Includes fields like type (advertising, scan response, etc.) and length of the payload.

Data Packets: Contains flags such as LLID (Link Layer Identifier), NESN (Next Expected Sequence Number), and SN (Sequence Number) for managing flow control and acknowledgment.

BLE Packet Structure

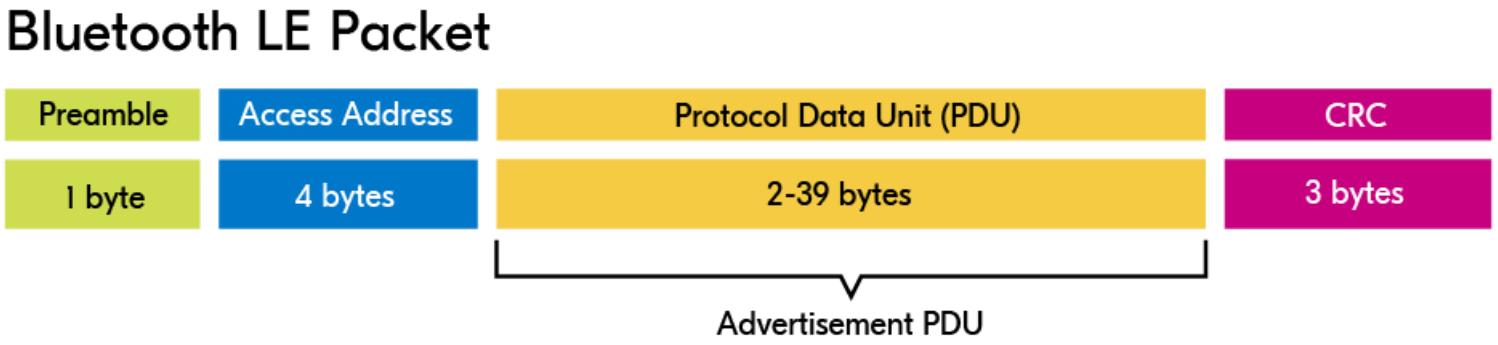


Payload (up to 37 bytes):

Advertisement Packets: Contains device-specific information like the device name, connection requests, and service data. It could include information such as the device's MAC address or sensor data.

Data Packets: Carries actual data being transmitted between the devices during a connection (after advertising). It can include application-level data, like sensor readings or control signals.

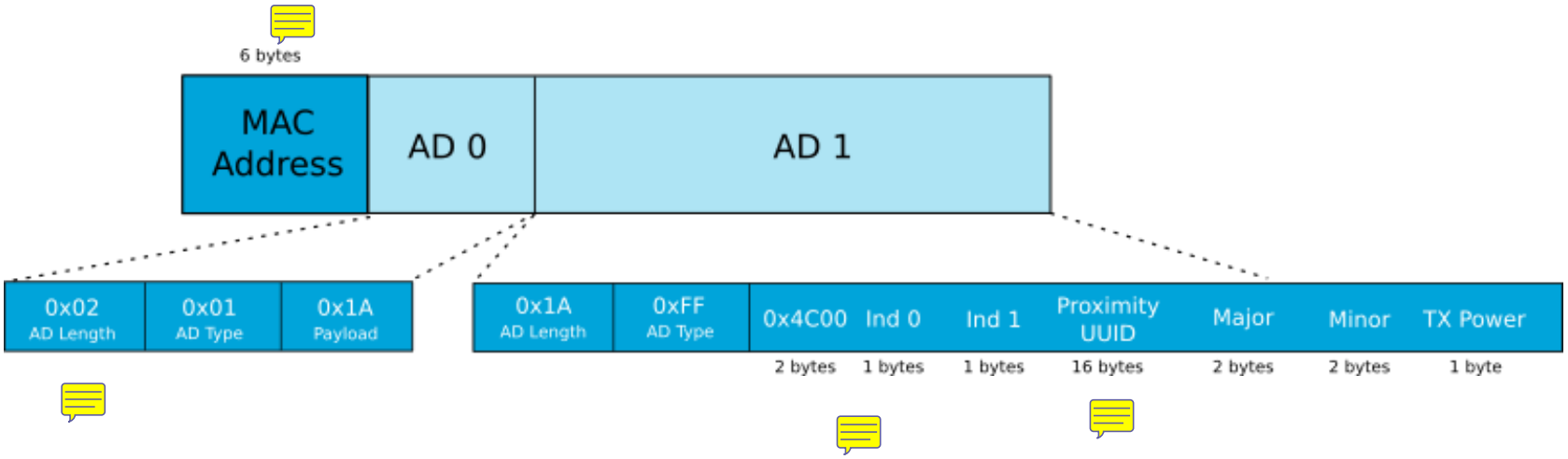
BLE Packet Structure



CRC (3 bytes):
The Cyclic Redundancy Check (CRC) is used for error detection, ensuring the integrity of the data being transmitted. If the CRC check fails, the packet is discarded.

Example

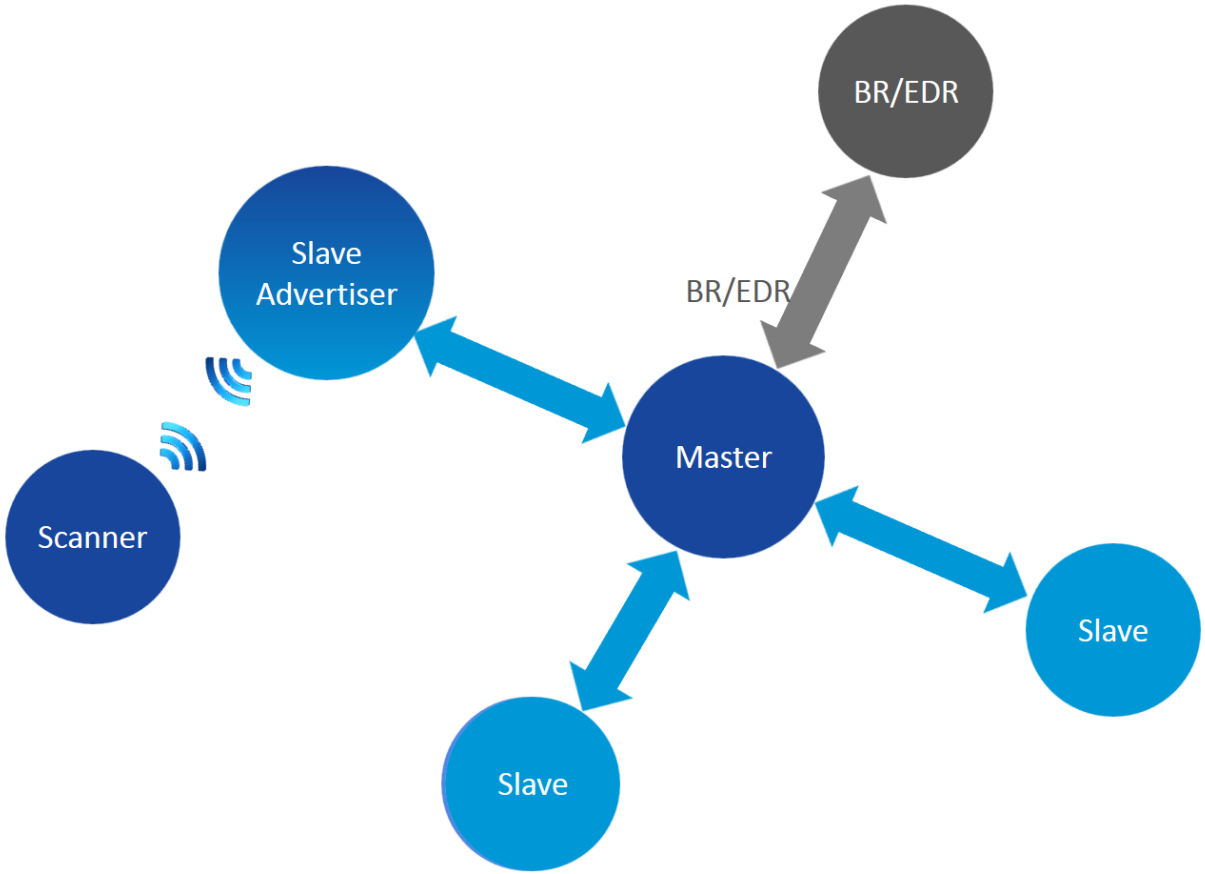
- Apple BLE beacons



Generic information

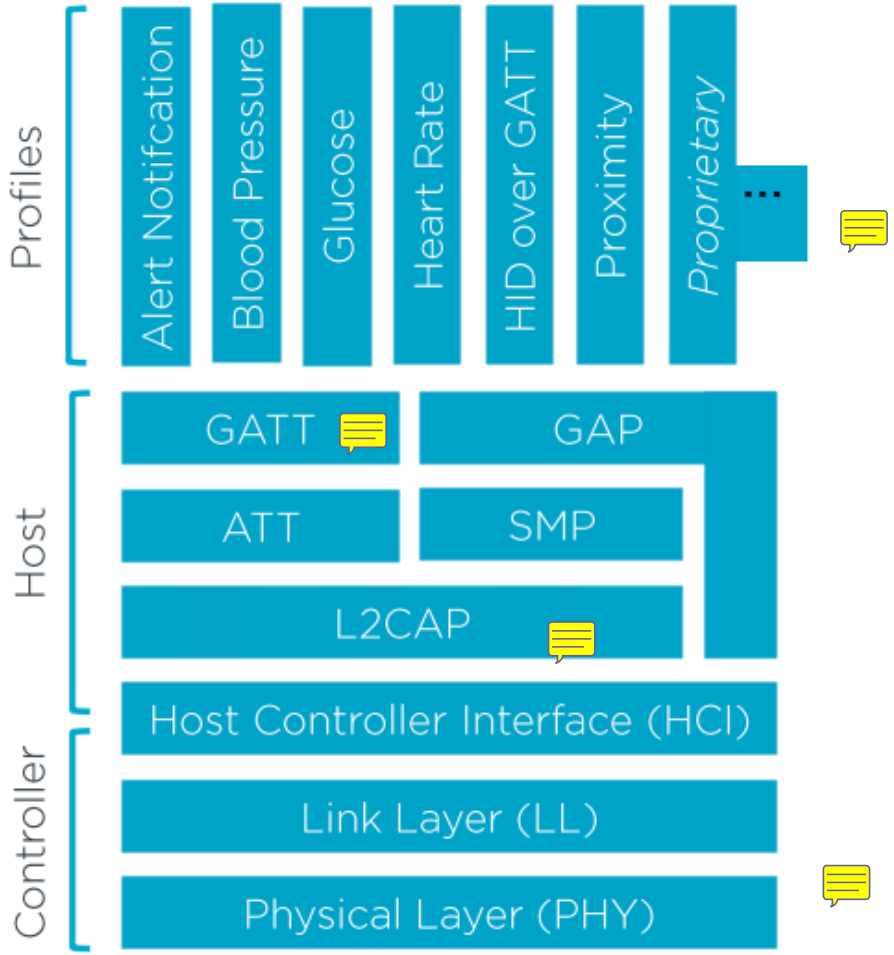
iBeacon information

How BLE works



BLUETOOTH TOPOLOGY

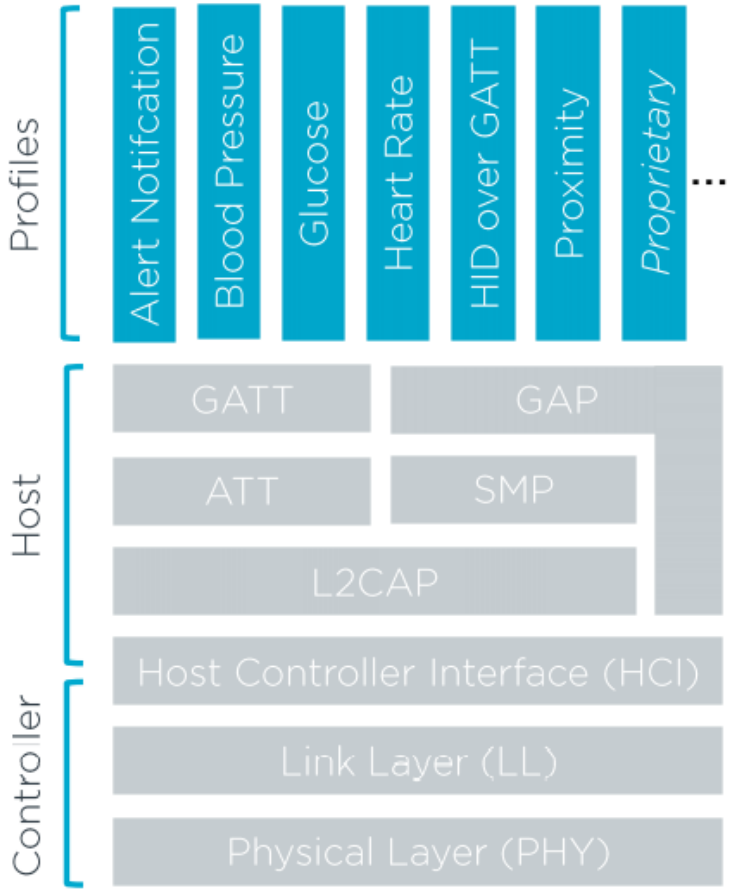
BLE ACHITECTURE



Nordic: Introductory to BLE

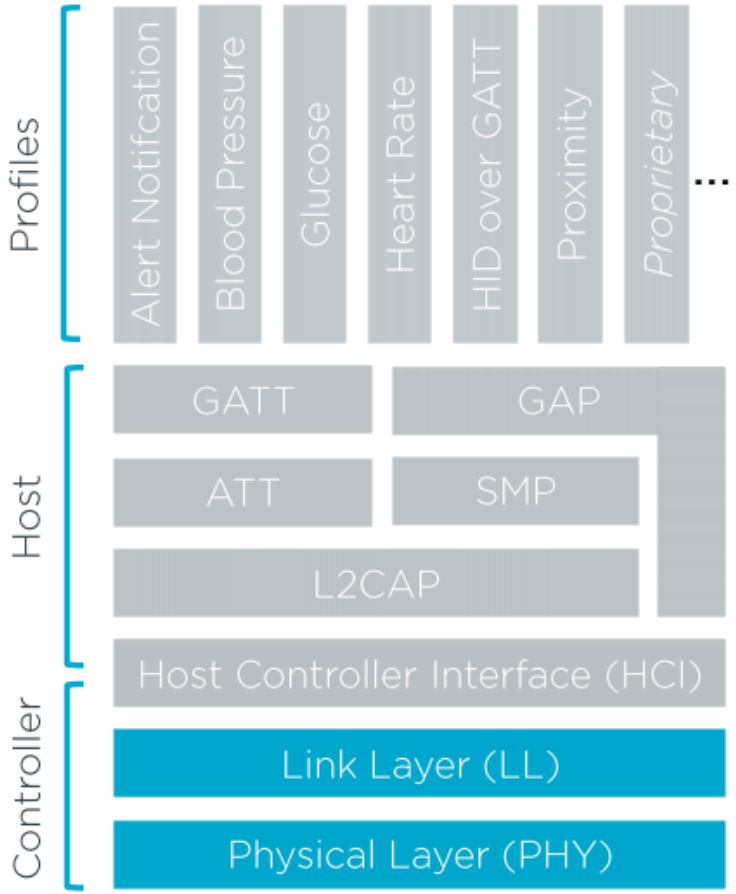
Profiles

- Describe how devices can discover and communicate with each other
- Each profile has its own specification

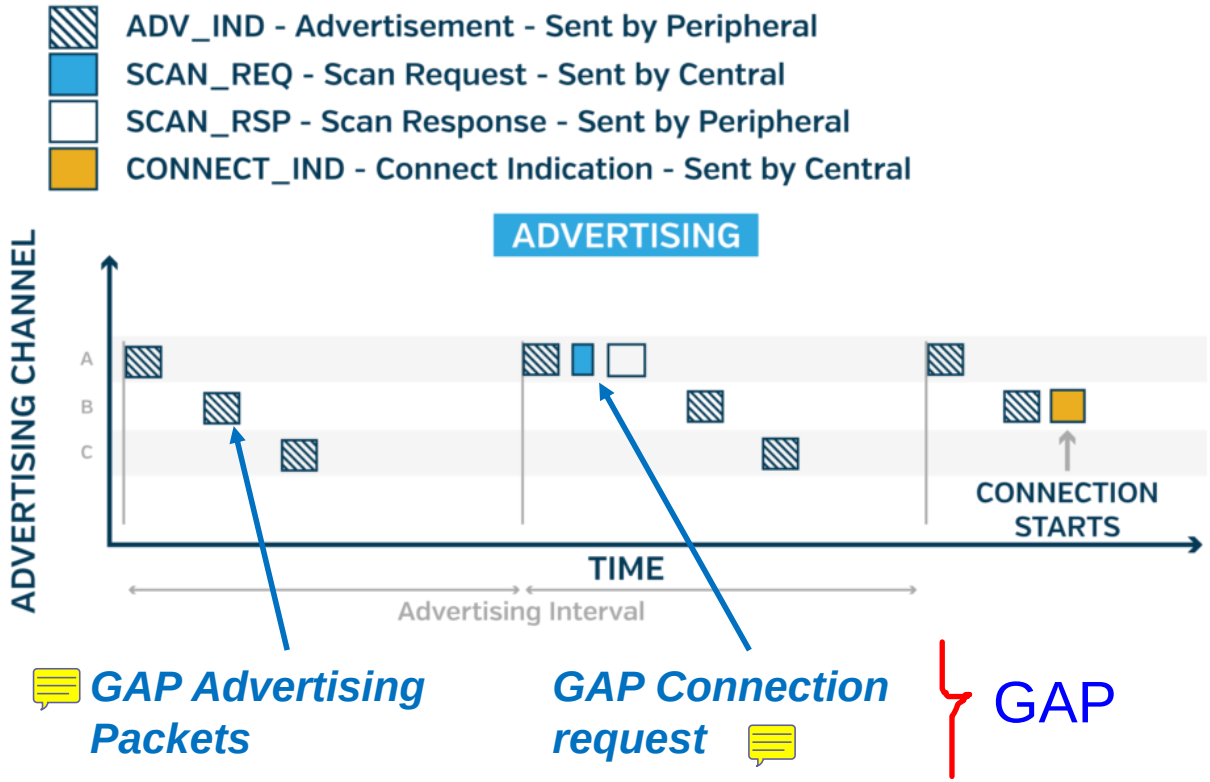


Controller

- Physical layer
  - Define how radios can send signal
- Link layer
  - Define device address
  - Define Link states
  - Packet format



GAP and GAT



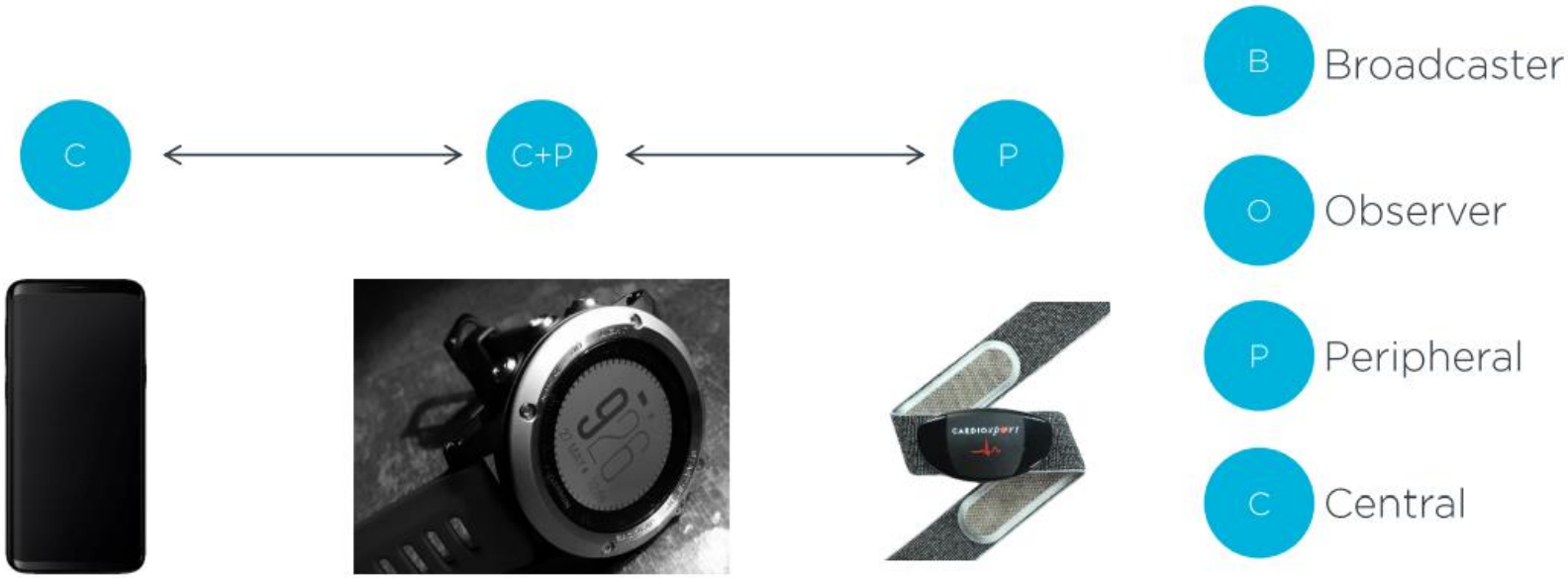
GAP is responsible for handling device discovery and connection establishment.

- Advertising packets
- Connection request packets

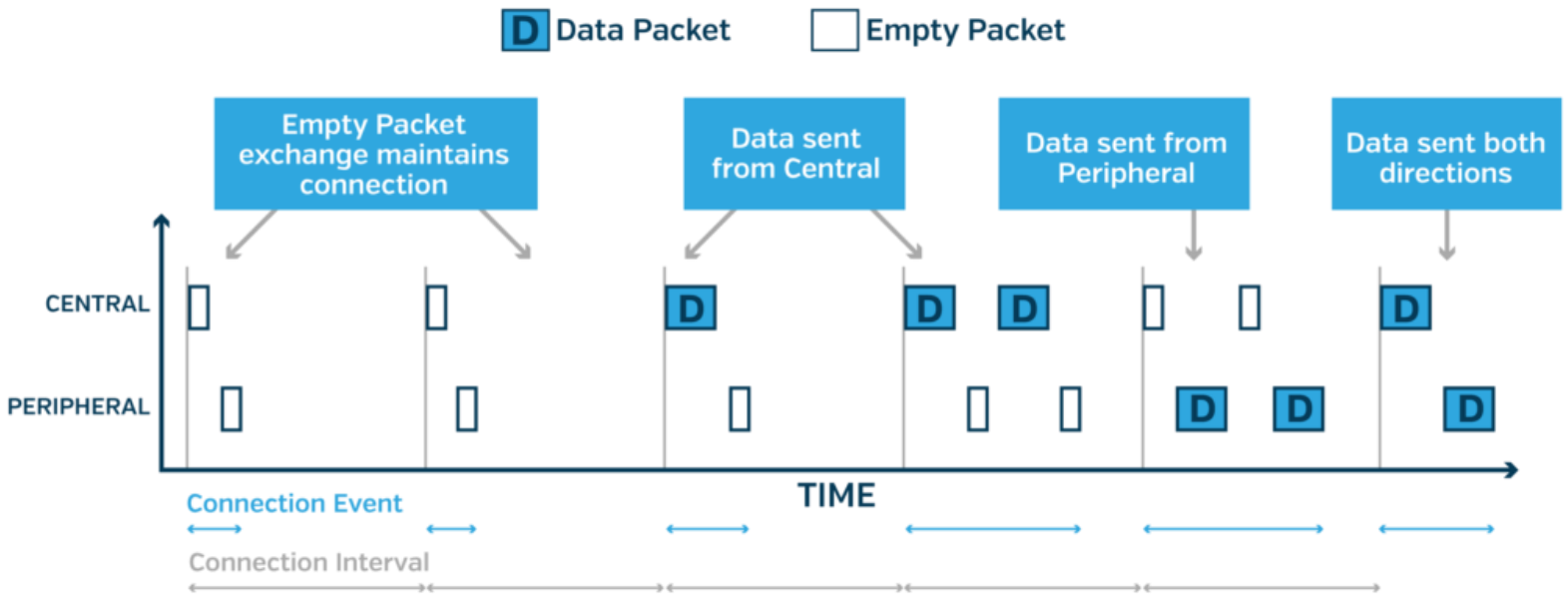
Generic Access Profile (GAP)

- GAP dictates how devices interact with each other at a low level
- GAP is the BLE topmost control layer
- It's main focus are
  - Roles and interaction between them *(broadcaster, observer, central or peripheral)*
  - Operational modes and transitions across those *(advertising and connection modes)*
  - Security

Generic Access Profile (GAP)



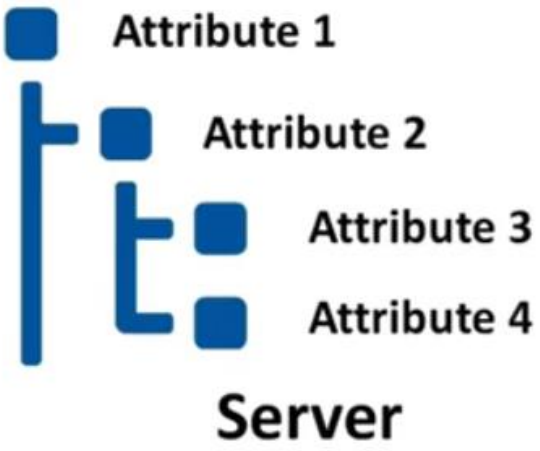
GAP and GAT

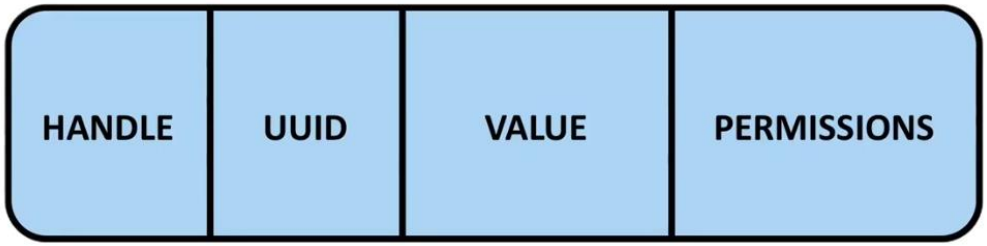


GATT facilitates the data exchange between the devices
- defines how the payload part of the packet is structured for reading, writing, and notifying data

ATTRIBUTE PROTOCOL (ATT)

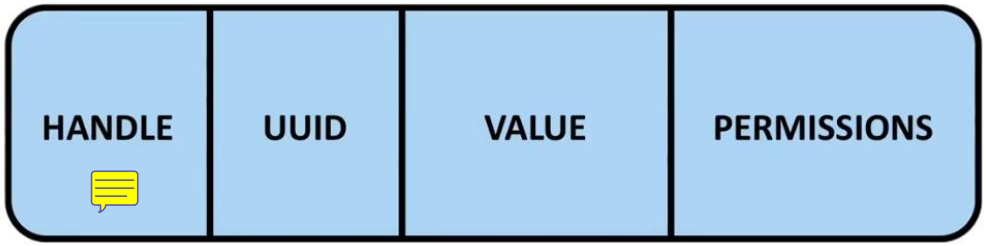
- Attribute Protocol (ATT) is a simple client/server protocol
  - client requests data from a server
  - server sends data to it's clients
- Server contains data organized in the form of attributes





Attribute Structure

ATTRIBUTE PROTOCOL (ATT)



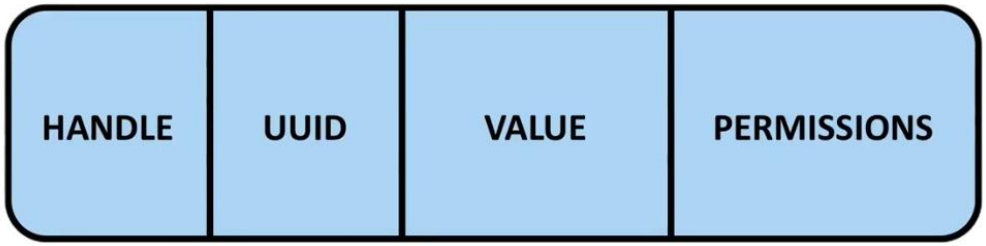
Attribute Structure

- **Attribute Handle**
  - 16-bit value that the server assigns → Address
  - Range 0x0001-0xFFFF
  - Used by the client to reference a specific attribute
- **UUID - Universally Unique Identifier**
  - 128-bit number
  - Specify the type and nature of the data
  - Example:

    SIG - UUID : 00000000-0000-1000-8000-00805F9B34FB

    Custom : F5A1287E-227D-4C9E-AD2C-11D0FD6ED640

ATTRIBUTE PROTOCOL (ATT)

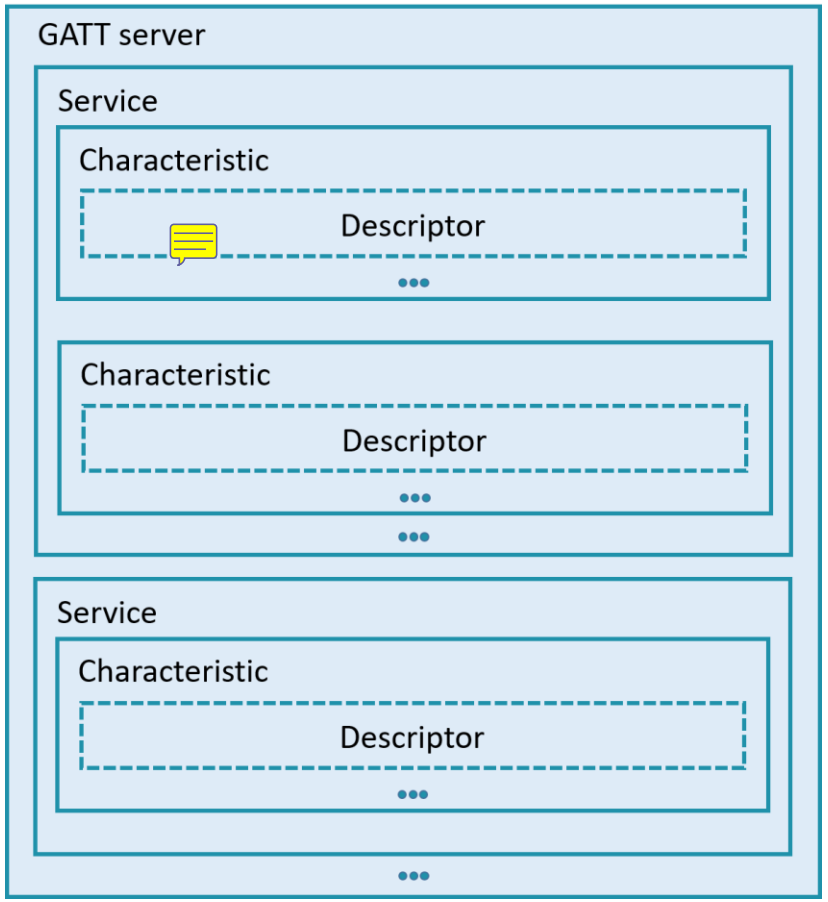


Attribute Structure

- **Value**
  - Hold data
  - Has variable length
- **Permission**
  - determine whether an attribute can be read or written to
  - whether it can be notified or indicated
  - Security levels

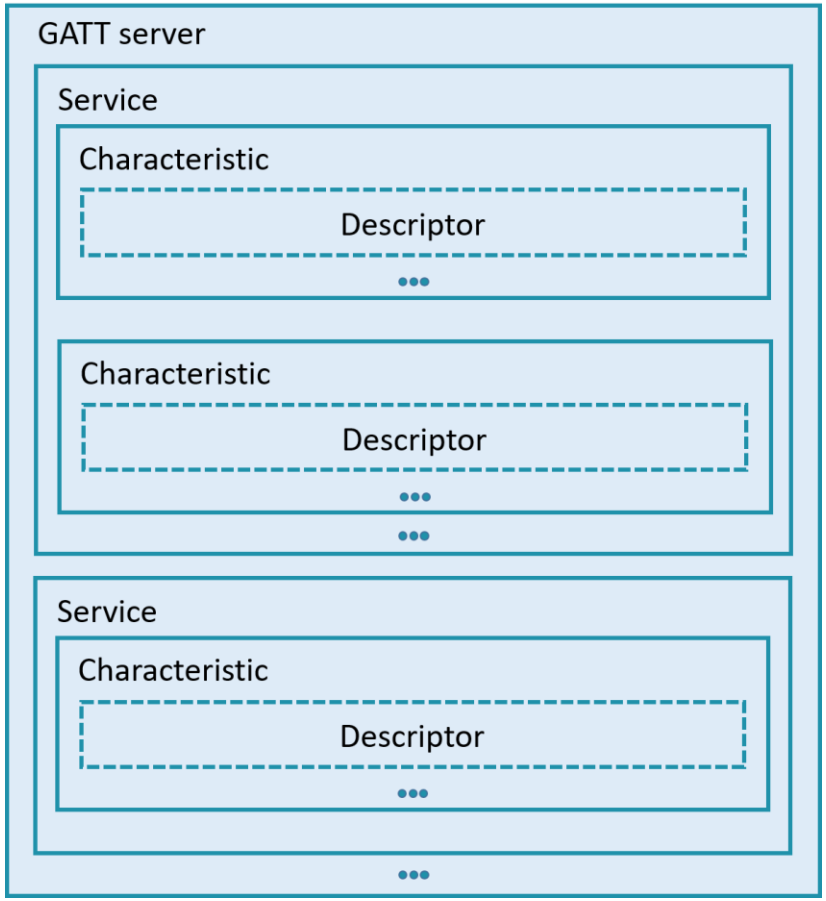
GENERIC ATTRIBUTE PROFILE (GATT)

- GATT dictates how ATT is employed in BLE communication
- GATT deals only with actual data transfer procedures and formats
- Data is organized hierarchically in sections called **services**



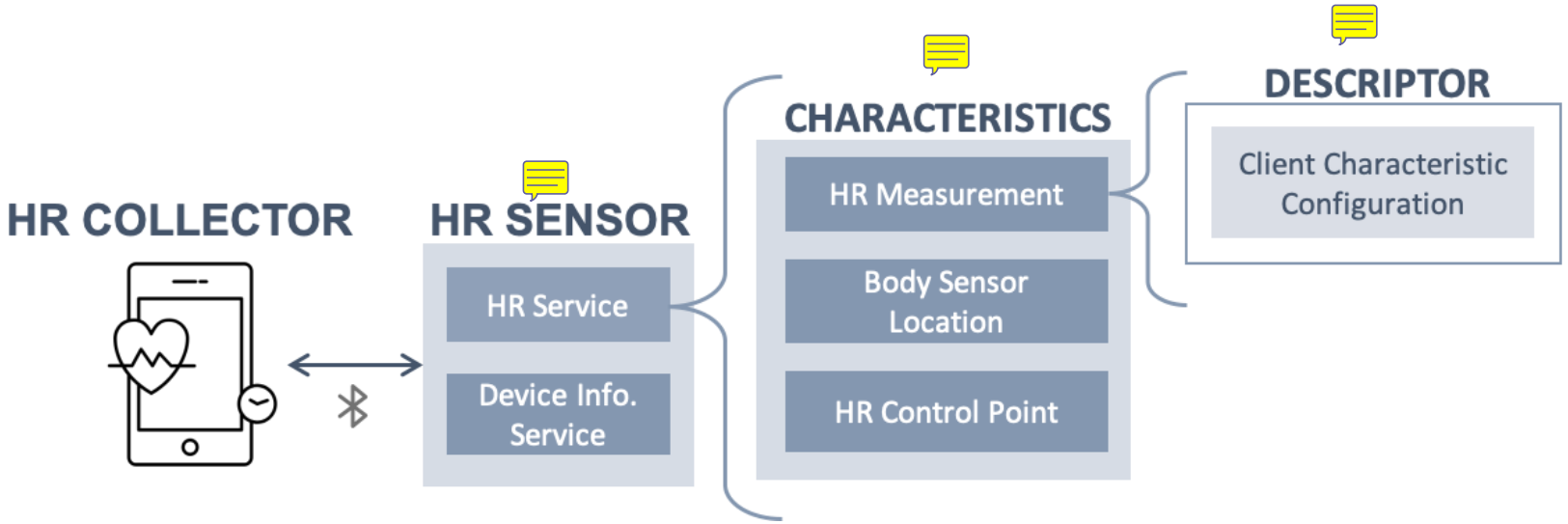
GENERIC ATTRIBUTE PROFILE (GATT)

- Service
  - group of attributes
- Characteristic
  - End-point data
  - Always contain descriptor and value



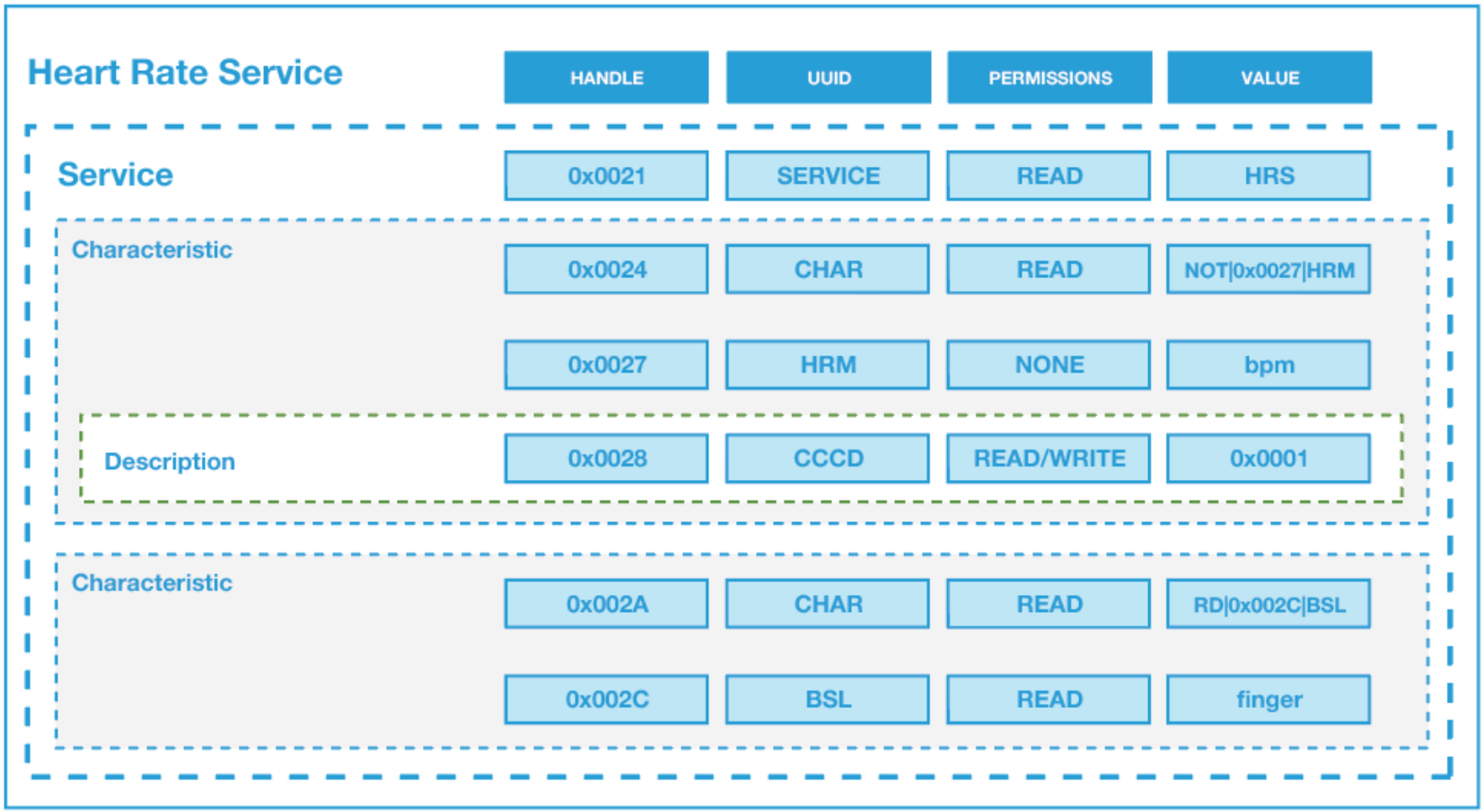
GENERIC ATTRIBUTE PROFILE (GATT)

- Example



GENERIC ATTRIBUTE PROFILE (GATT)

- Example



GENERIC ATTRIBUTE PROFILE (GATT)

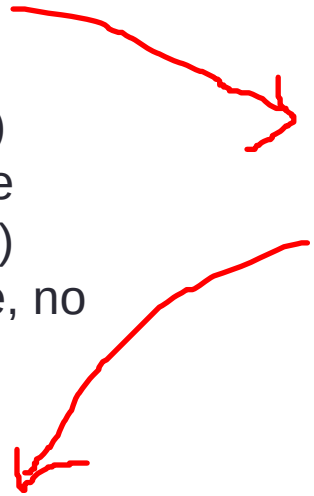
- Example

[0x1B, 0x25, 0x00, 0x48]

HR sensor
Opcode: 0x1B (Notification)
Handle: 0x0025 (Heart Rate Measurement characteristic)
Flags: 0x00 (8-bit heart rate, no additional data)
Heart rate: 72 bpm (0x48)

[0x0A, 0x25, 0x00]

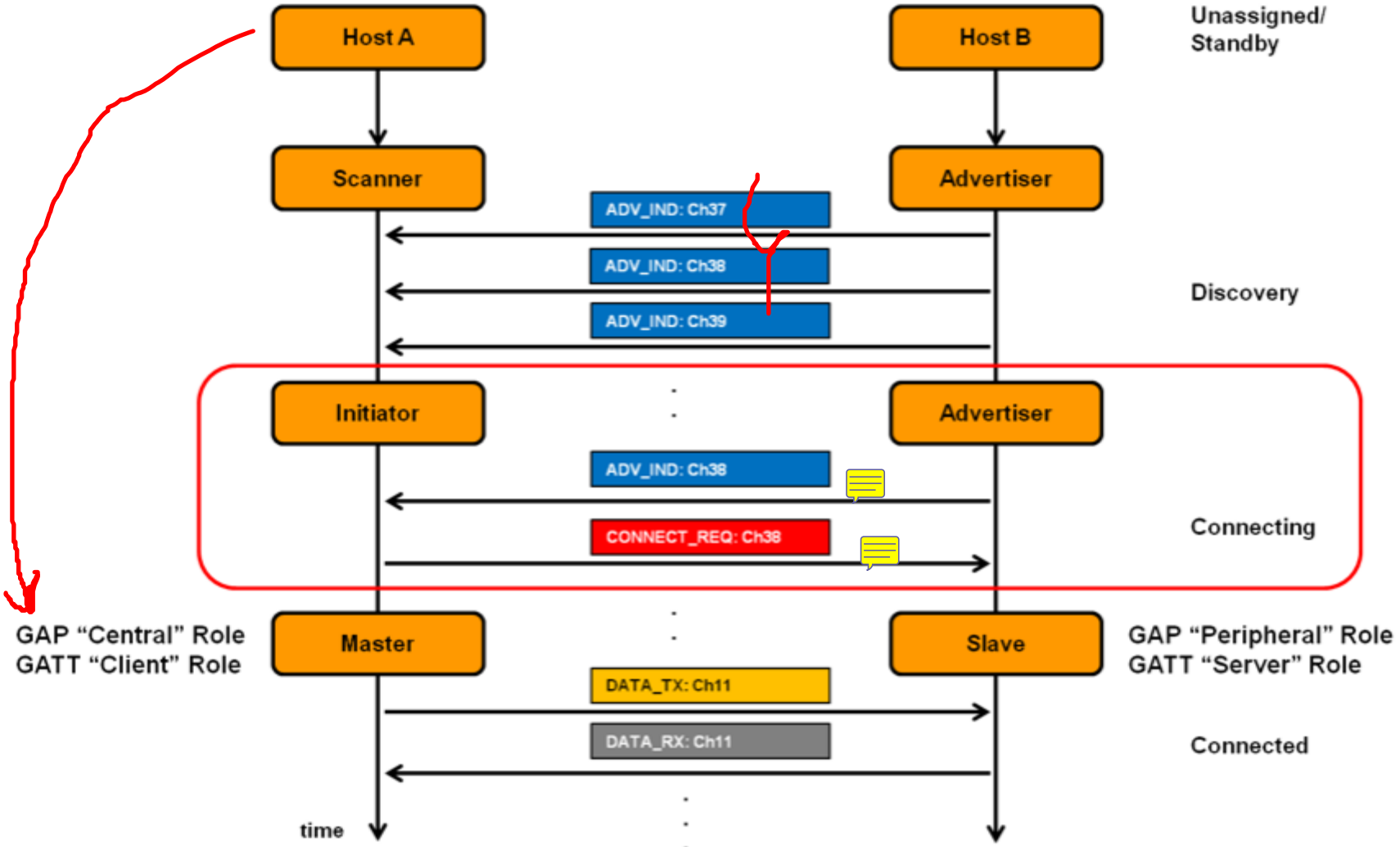
HR collector
Opcode: 0x0A (Read request)
Handle: 0x0025 (Heart Rate Measurement characteristic)



[0x0B, 0x25, 0x00, 0x49]

Opcode: 0x0B (Read Response)
Handle: 0x0025 (Heart Rate Measurement characteristic)
Flags: 0x00 (8-bit heart rate, no additional data)
Heart rate: 73 bpm (0x49)

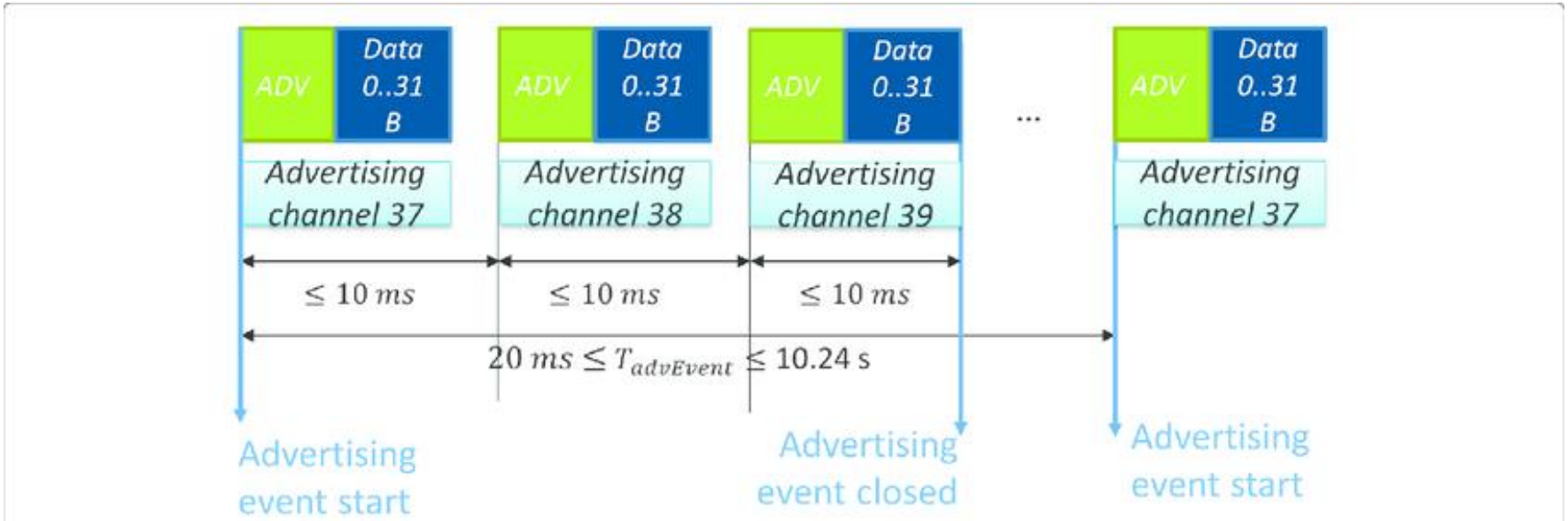
BLE Communication



Advertising

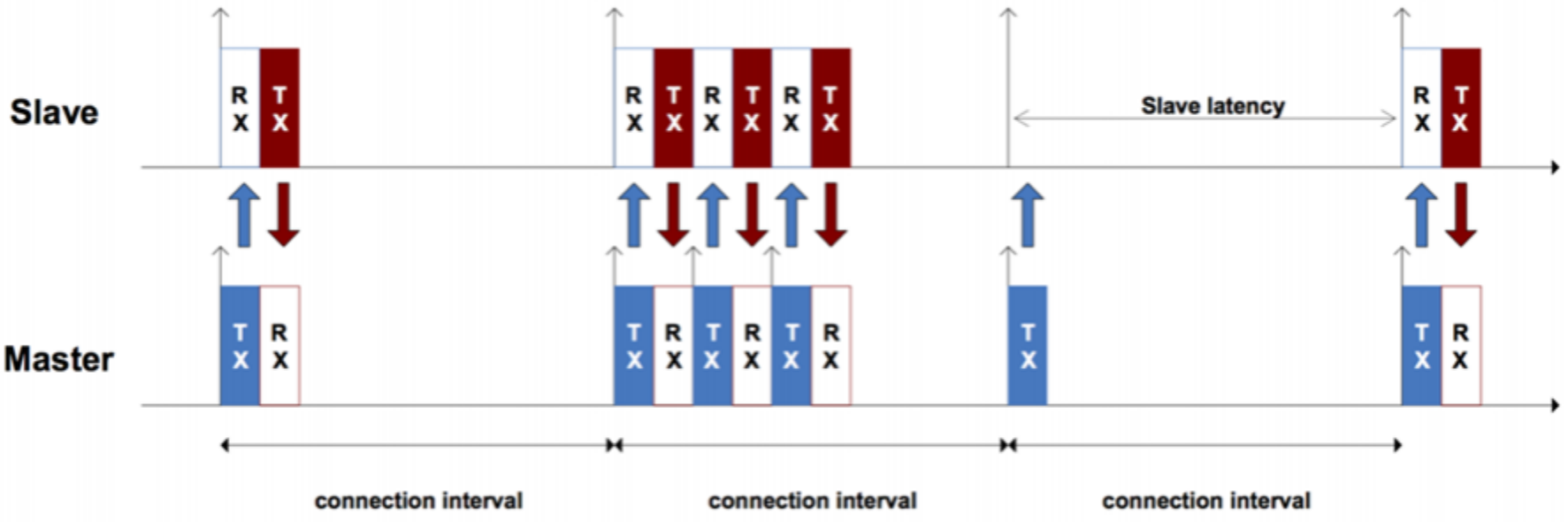
- An Advertisement packet contain:
  - 6 bytes for address (MAC), device name
  - 0 – 31 bytes for advertisement data

Advertising



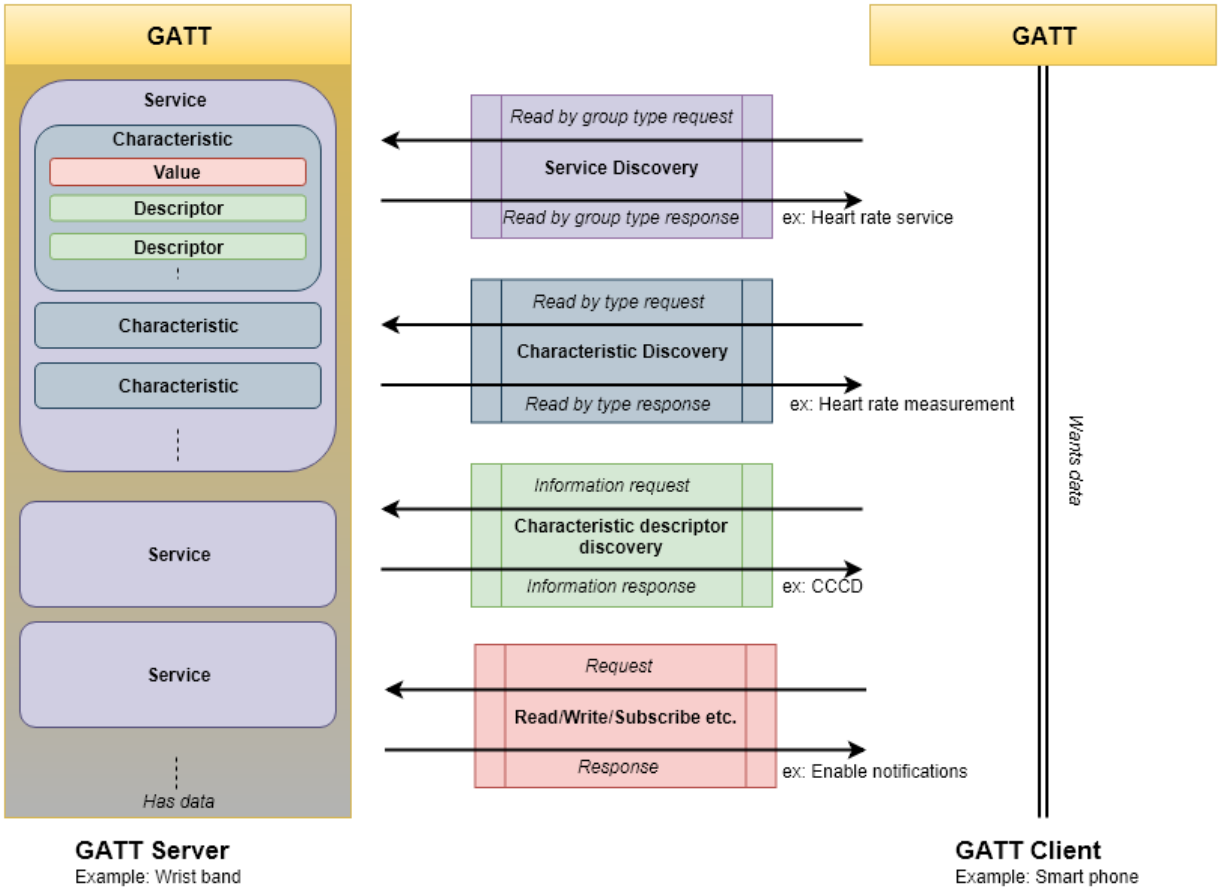
Advertising Timing

Connection



- **Connection events**
  - The **connection interval** is between **7.5 ms** to **4 s** (step size: 1.25 ms)
  - 0-byte data packets are exchanged if there is no other data to exchange

GENERIC ATTRIBUTE PROFILE (GATT)

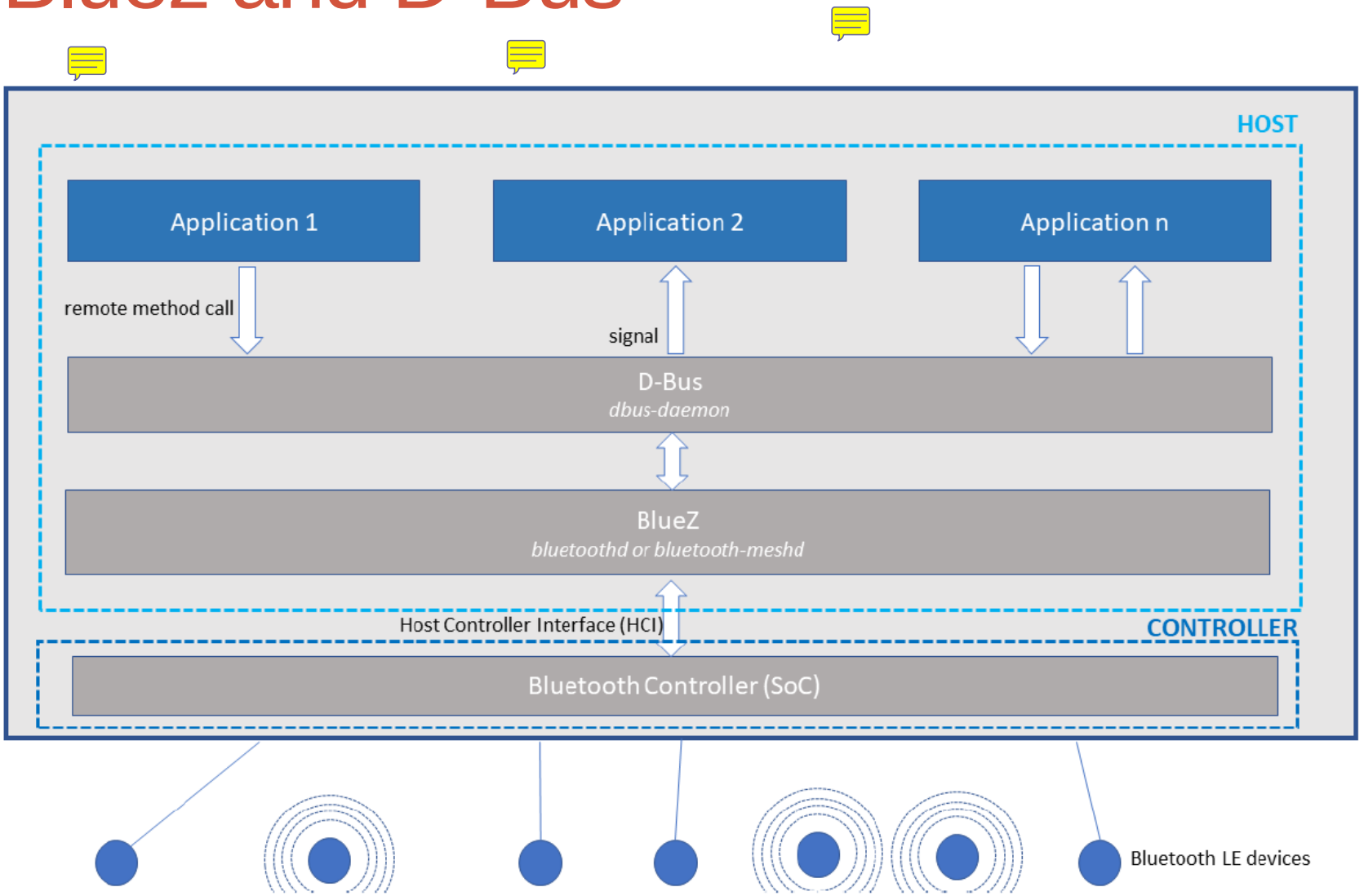


BLE GATT Client-Server model

Examples

- https://punchthrough.com/creating-a-ble-peripheral-with-bluez/

Bluez and D-Bus



D-Bus system bus