Bluetooth Low Energy

Bluetooth Low Energy (BLE)

- Bluetooth Low Energy (BLE), aka "Bluetooth Smart", is a lightweight subset of the Bluetooth 4.0 core specification
- Originally designed by Nokia as Wibree then adopted by the Bluetooth Special Interest Group (2010)
- Goal: design a radio standard with the **lowest possible power consumption**, specifically optimized for low cost, low bandwidth
  - A coin cell battery may last for years
  - $2 for a system-on-chip
  - Typical throughput is in the order of 10-100 kbps
- Early adopted by Apple and now one of the most widely used wireless communication technologies
- Smartphones and tablets greatly contributed to massive adoption
  - Rich UI, low barrier to adoption of devices that talk to smartphones
  - Users familiar with using their smartphones, effort about learning a new UI is reduced

Basic concepts

Everything has a **state**

- devices expose their state (information)
- these devices are servers

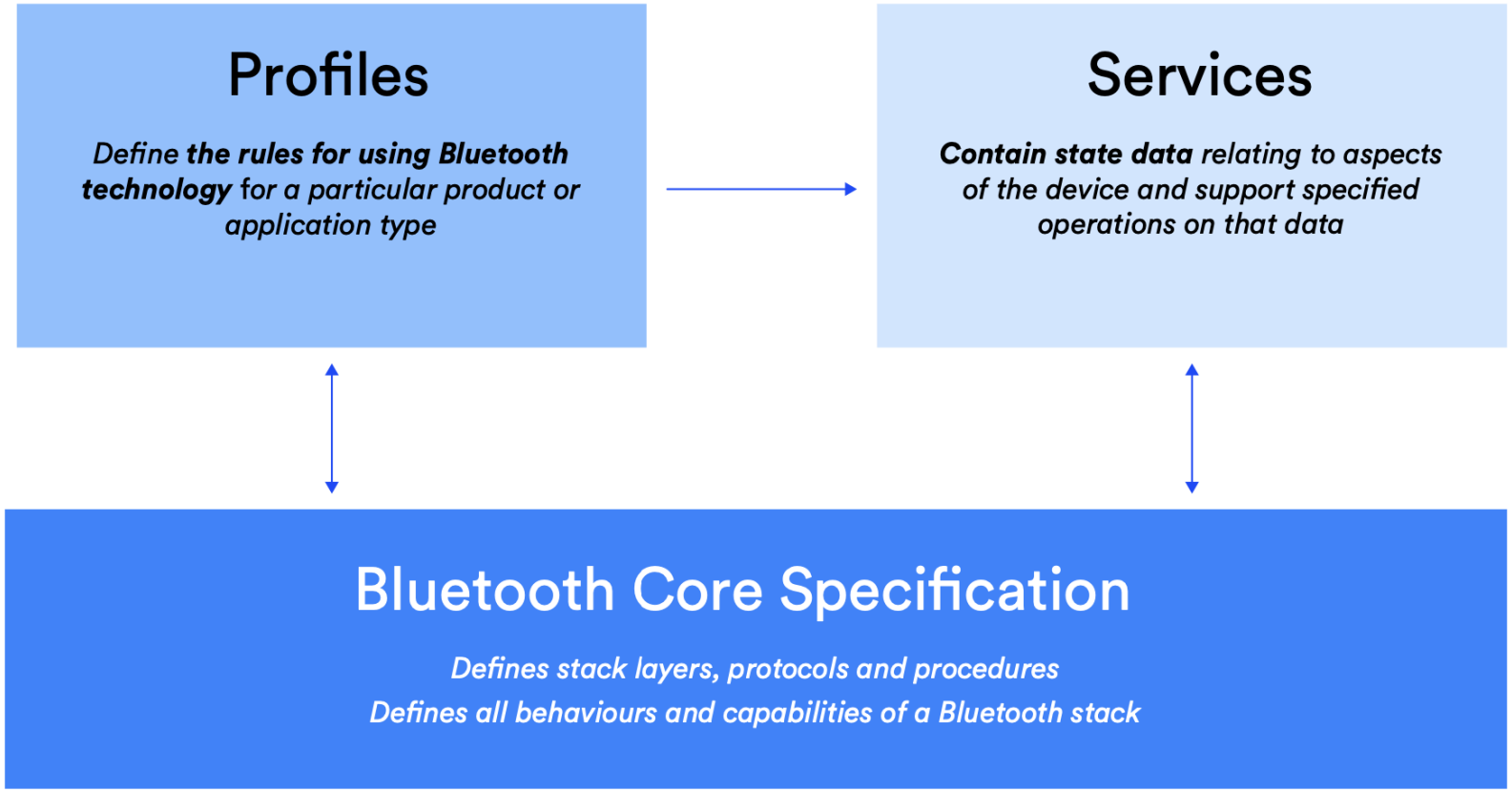
Clients can use the state exposed on servers

- read state, e.g. get current temperature
- write state, e.g. increase the temperature of a room

Servers can tell clients when state changes

- notify change, e.g. temperature reached the desired value

BLE Specifications

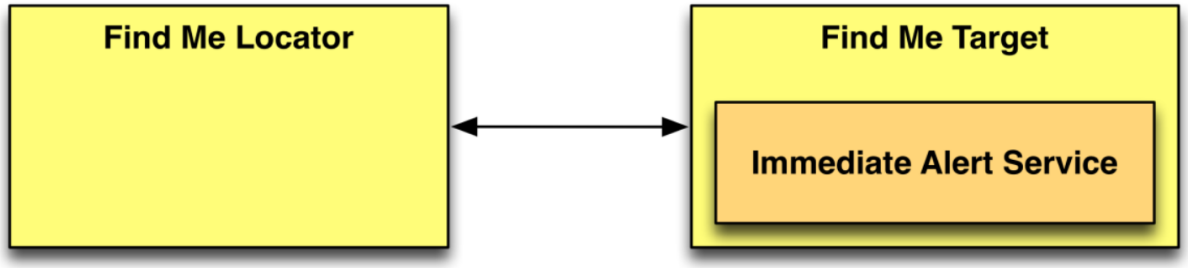


BLE Specifications

- The **Bluetooth Core Specification** is the primary specification for both Bluetooth LE and Bluetooth Classic. It defines the architecture of the technology and its layers.
- **Profile specifications**: When two Bluetooth LE devices are communicating over a connection, it is usually the case that a client / server relationship has been formed.
  - Servers contain state data and clients use data
  - Example:
    - key fob helps you find your lost keys
    - a smartwatch might act as the client device and the Bluetooth key fob would act as the server
    - pressing a button on the smartwatch's display could cause the key fob's state to be changed and it to make a loud noise
  - Profile specifications define the roles that related devices assume (*Find Me* Profile specification)
- **Service specifications**: data on servers resides in organized in characteristics and descriptors.
  - Characteristics and descriptors are grouped inside constructs known as services.
  - Services provide a context within which to assign meaning and behaviours to the characteristics and descriptors that they contain.
  - In the smartwatch and key fob example, the key fob, is acting as a server and implements the Immediate Alert Service.

Profile specification: Find Me

- *Find Me* profile defines the behavior when a button is pressed on a device to cause an immediate alert on a peer device. This can be used to allow users to find devices that have been misplaced.



The Find Me Target has an instance of the Immediate Alert service.

5.1.2 Connection Procedure for Unbonded Devices

This procedure is used for device discovery and connection establishment when the Peripheral is unbonded and the user initiates a connection.

It is recommended that the Peripheral advertises using the parameters in Table 5.1. The interval values in the first row are designed to attempt fast connection during the first 30 seconds; however, if a connection is not established within that time, the interval values in the second row are designed to reduce power consumption for devices that continue to advertise.

Advertising Duration	Parameter	Value
First 30 seconds (fast connection)	Advertising Interval	20 ms to 30 ms
After 30 seconds (reduced power)	Advertising Interval	1 s to 2.5 s

Table 5.1: Recommended advertising interval values

Service specification: Immediate Alert

- The UUID value assigned to «Immediate Alert» is ...
- There shall only be one instance of the Immediate Alert service on a device.

Characteristic	Ref.	Mandatory / Optional
Alert Level	3.1	M

Table 3.1: Service characteristics

3.1 Alert Level

The Alert Level characteristic is a control point that allows a peer to command this device to alert to a given level.

3.1.1 Characteristic Behavior

The Alert Level characteristic can be written using the GATT Write Without Response sub-procedure with an alert level of either “No Alert," “Mild Alert," “High Alert," to set the written alert level.

4.1 Writing Alert Level Behavior

When the Alert Level characteristic is written the device shall start alerting to the written alert level.

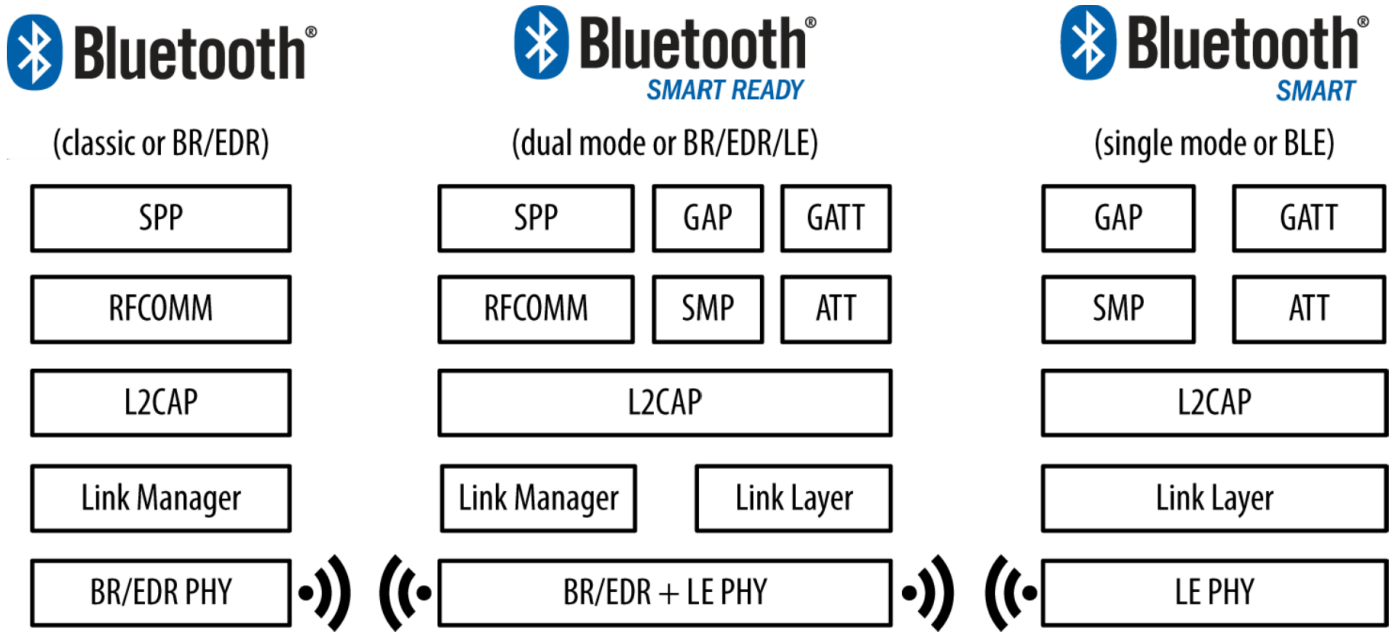
If the written alert level is “No Alert," no alerting shall be done on this device.

If the written alert level is “Mild Alert," the device shall alert.

If the written alert level is “High Alert," the device shall alert in the strongest possible way.

Classic Bluetooth and BLE

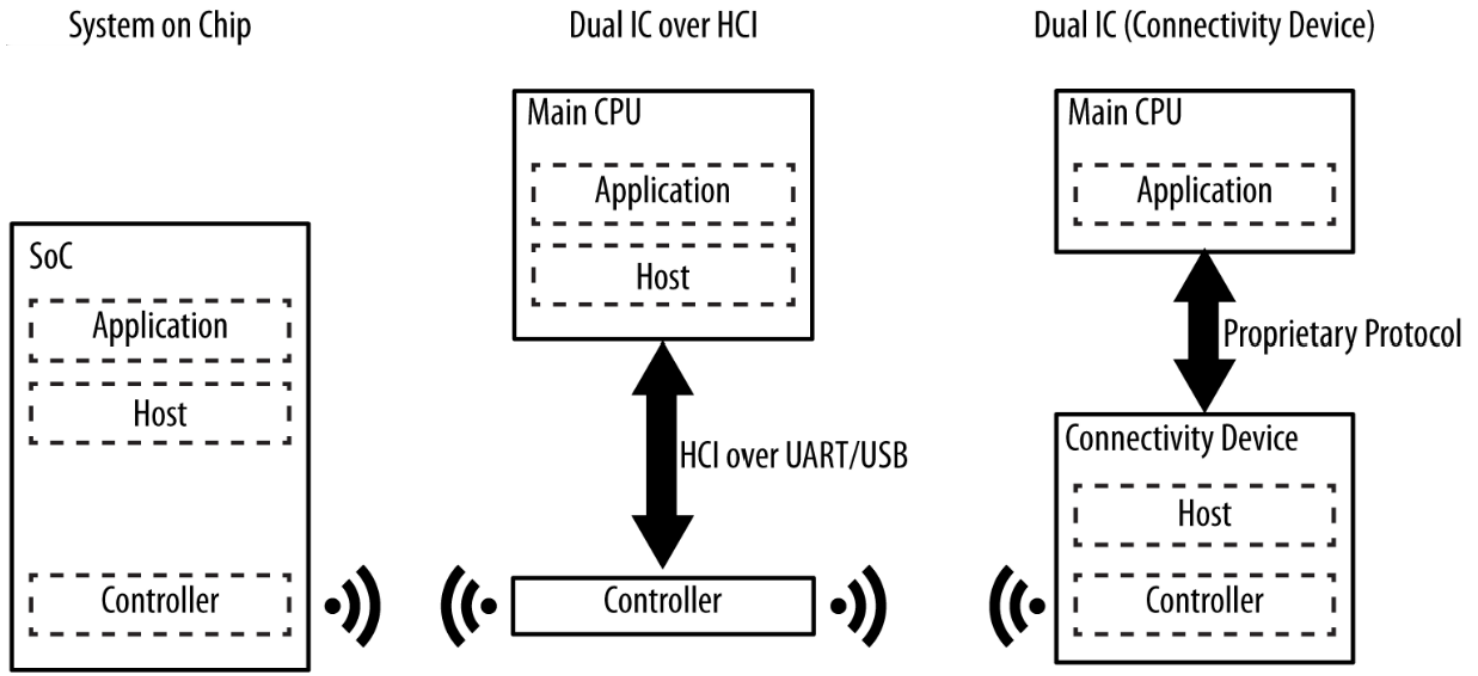
- Not compatible with classic bluetooth
- Need double stack
- Simple devices only BLE



Source: Getting started with Bluetooth Low Energy

BLE

- HW configurations
- Simple devices generally based on SoC
- Smartphones rely more on SW



Throughput

- Because of BLE radio modulation -> 1Mbps theoretical upper limit
- Generally much lower for power efficiency
- Let’s suppose a master device is connected with a slave device
- Connection interval is the interval between two consecutive connection events
- Connection events: data is exchanged then devices go back to idle state to save energy
- Connection interval can be set to a value between 7.5 ms and 4 s
- The nRF51822 can transmit up to six data packets per connection interval
- Each data packet can contain up to 20 bytes of user data (default)
- 133.3 connection events/s * 6 * 20 B = 16000 B/s = 128 kbps

Throughput

- ~15 KB/s might seem slow these days
- Design goal of BLE: low energy
- 15 KB/s quickly depletes a coin cell battery, throughput is generally much smaller
- Turn the radio off whenever possible, as long as possible
- Data transmitted in short bursts during connection events
- Connection events separated as much as possible to save energy
- 7.5 ms–4 s connection interval
  - Small interval: responsiveness, higher throughput
  - Large interval: save energy

Connection events

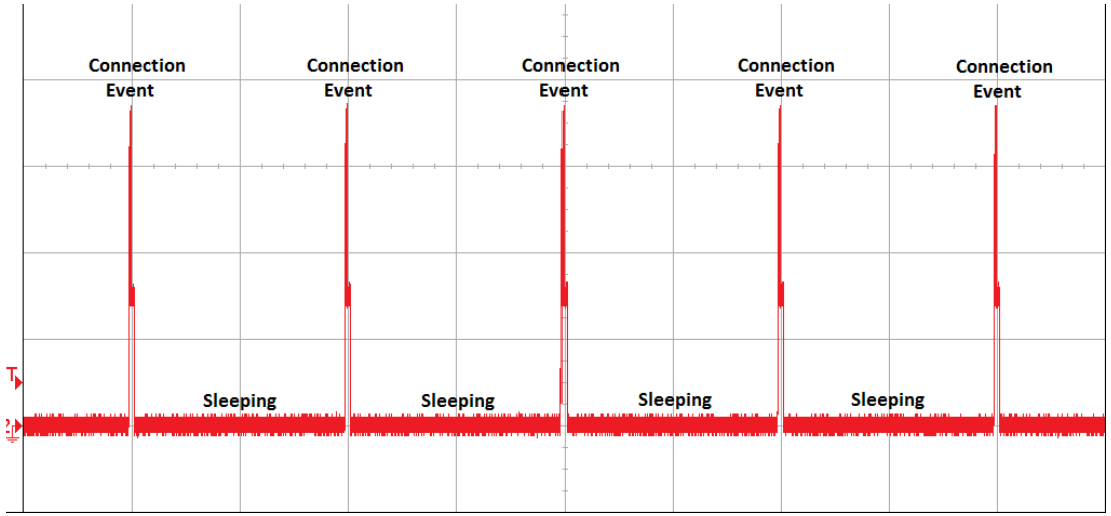


Figure 1- Current Consumption versus Time during a BLE Connection

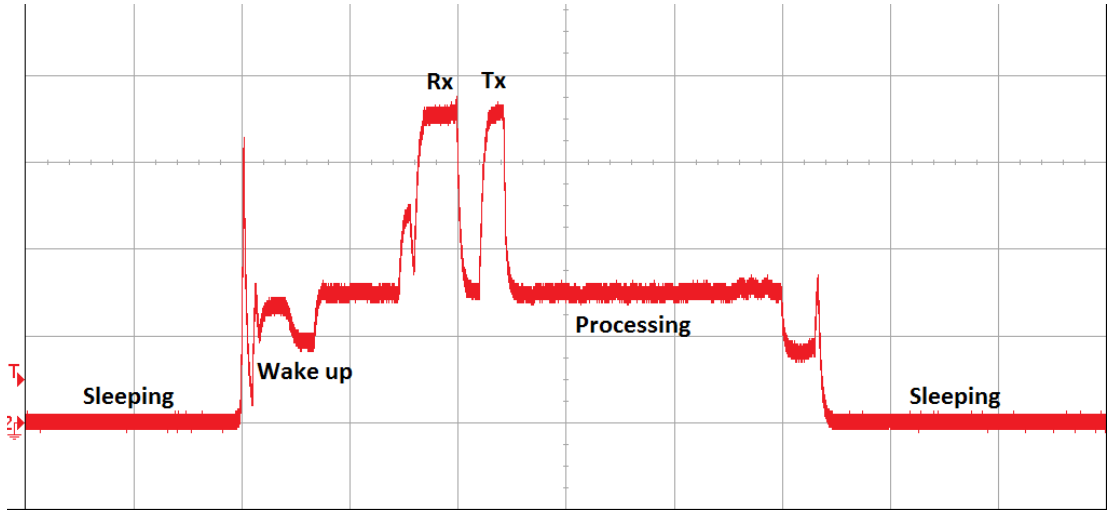


Figure 2- Current Consumption versus Time during a single Connection Event

Source: Texas Instruments app. notes

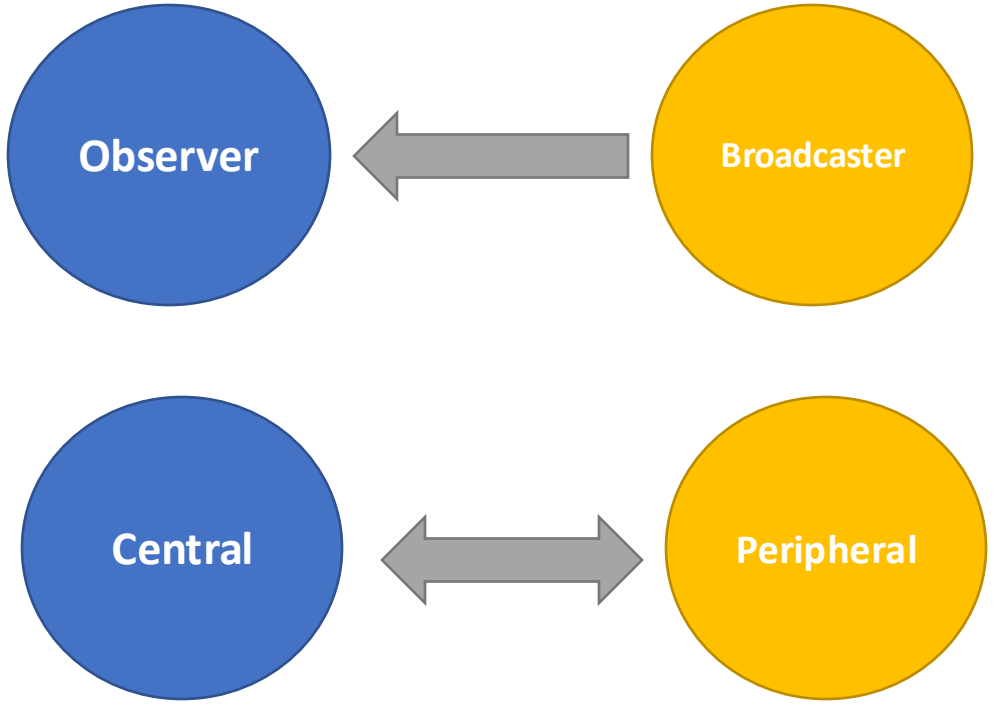
Transmission range

- Tx power can be configured (expressed in dBm)
- Maximum range: 30 m
- Typical range: 2-5 m to save energy
- Recent versions of BLE (5.2) make possible to dynamically adjust the Tx power
  - A Receiver may ask to increase/decrease power of Trasmitter if SNR is too low or if RSSI is too high

Two pairs of roles

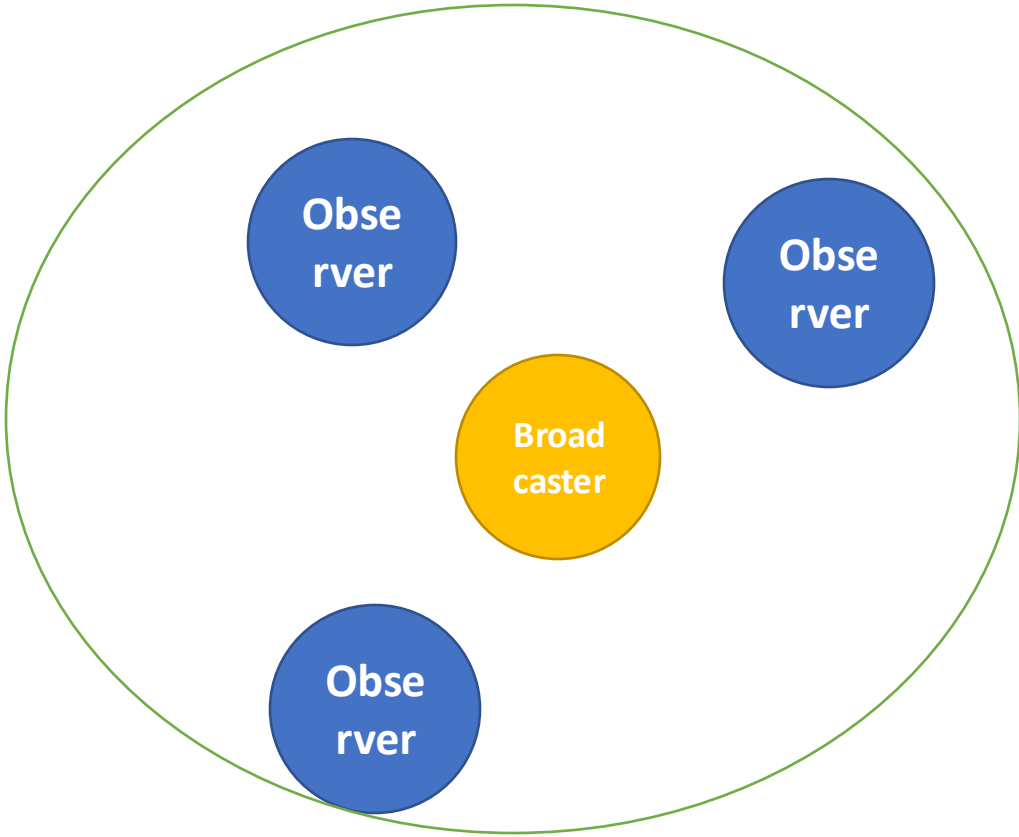
- *Broadcaster/Observer:* unidirectional, connectionless communications

- *Central/Peripheral*:
  - a mobile phone or tablet
  - small, low power devices that can connect to a central device. E.g. a heart rate monitor



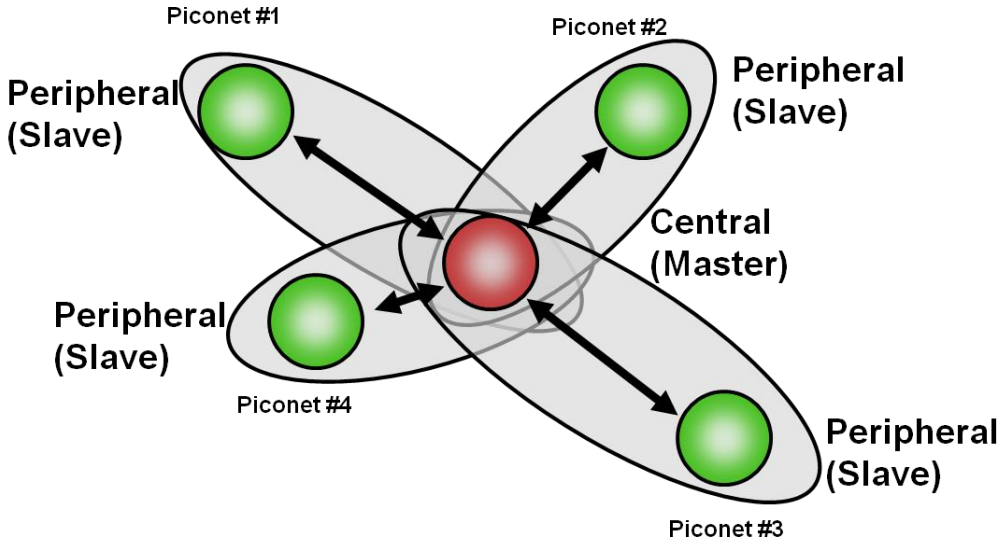
Broadcaster/Observer

- *Broadcaster*: Sends non-connectable advertising packets periodically to anyone willing to receive them
- *Observer*: Repeatedly scans frequencies to receive any non-connectable advertising packets being broadcasted
- The only way for a device to transmit data to more than one peer at time



Central/Peripheral

- **Connection**: permanent, periodical data exchange of packets between two devices
- Needed for transmitting:
  - data in both directions
  - more data than two advertising payloads can contain
- Central (master):
  - repeatedly scans frequencies for connectable advertising packets and starts a connection
  - once connection established, it manages the timing and starts the periodical data exchanges
- Peripheral (slave):
  - periodically sends connectable advertising packets
  - accepts incoming connections
  - when connected the peripheral follows the central's timing and exchanges data with it

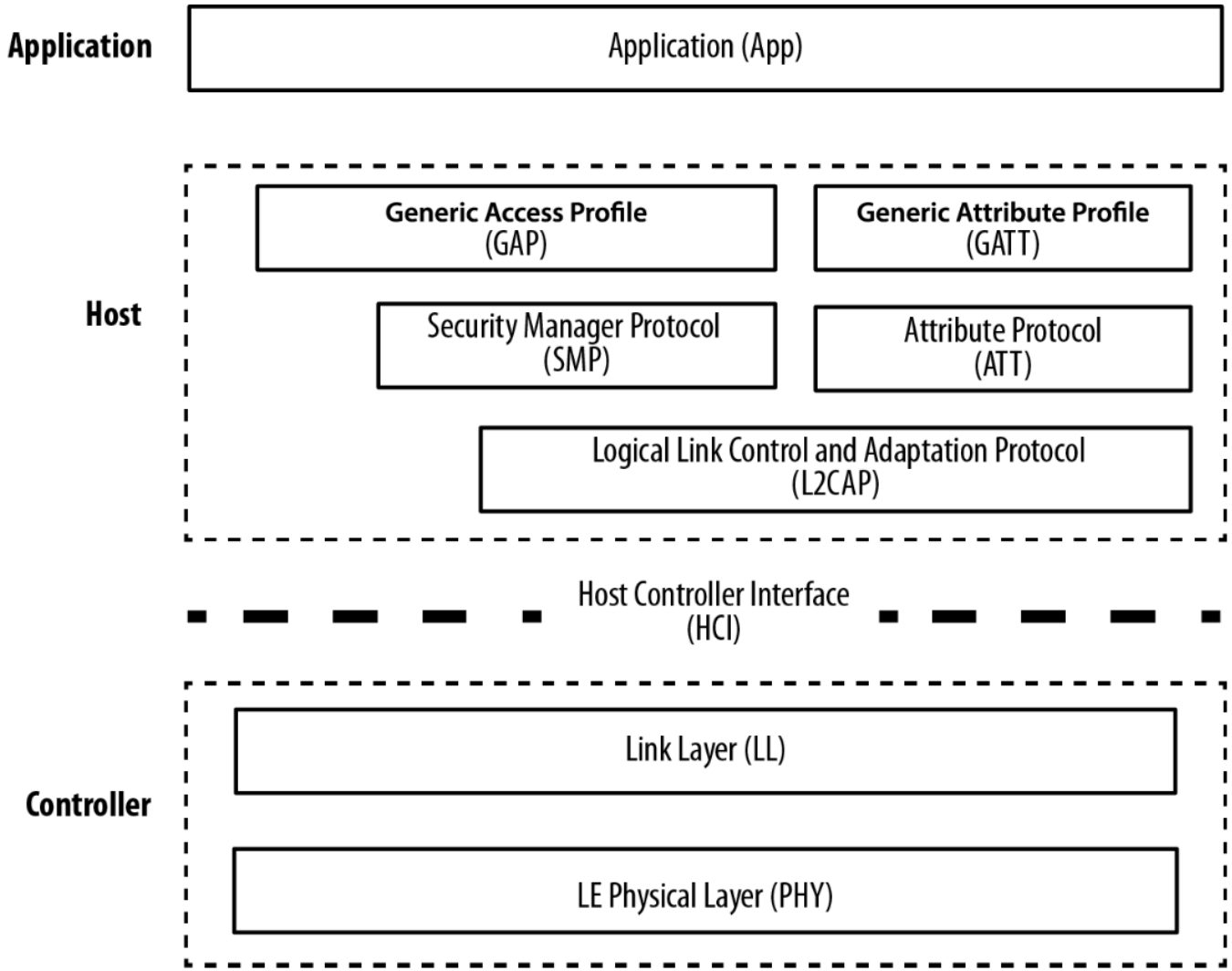


https://microchipdeveloper.com/

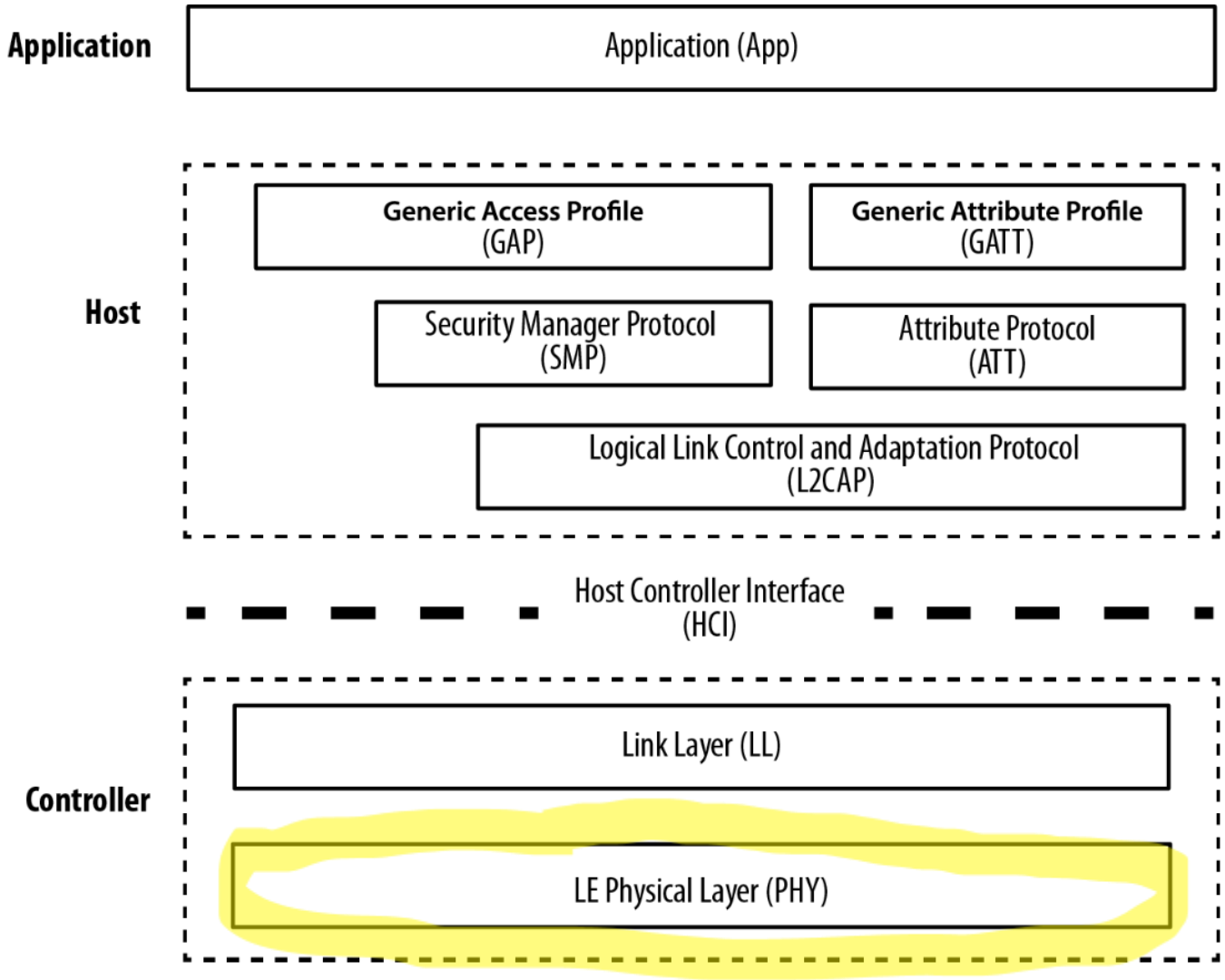
Central/Peripheral

- Topologies can be complex
  - A device can act as a central and a peripheral at the same time
  - A central can be connected to multiple peripheral nodes
  - A peripheral can be connected to multiple central nodes
- Can be more energy efficient than obs/brdcst, depending on communication behavior
- Can extend the delay between connection or send data out only when new values are available, rather than having to continually advertise the full payload at a specific rate without knowing who is listening or how often
- Both peers know when the connection events are going to take place in the future allows the radio to be turned off for longer, potentially saving battery power

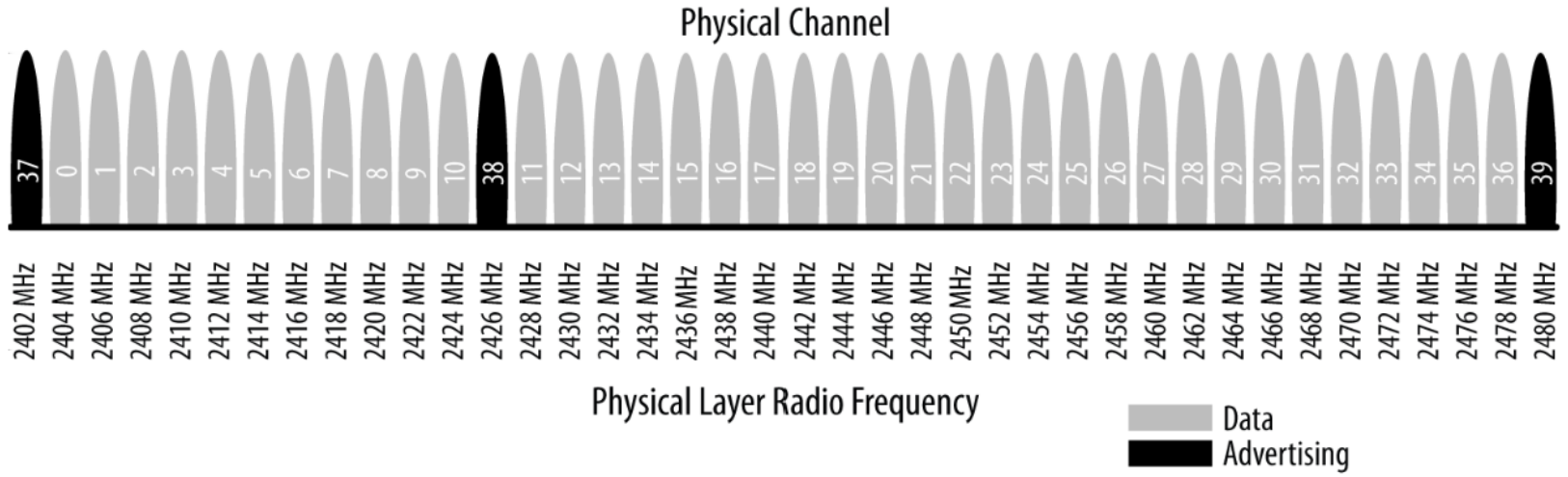
BLE stack


Application
Application (App)
Host
Generic Access Profile
(GAP)
Generic Attribute Profile
(GATT)
Security Manager Protocol
(SMP)
Attribute Protocol
(ATT)
Logical Link Control and Adaptation Protocol
(L2CAP)
Host Controller Interface
(HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

BLE stack


Application
Application (App)
Host
Generic Access Profile
(GAP)
Generic Attribute Profile
(GATT)
Security Manager Protocol
(SMP)
Attribute Protocol
(ATT)
Logical Link Control and Adaptation Protocol
(L2CAP)
Host Controller Interface
(HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

Physical layer

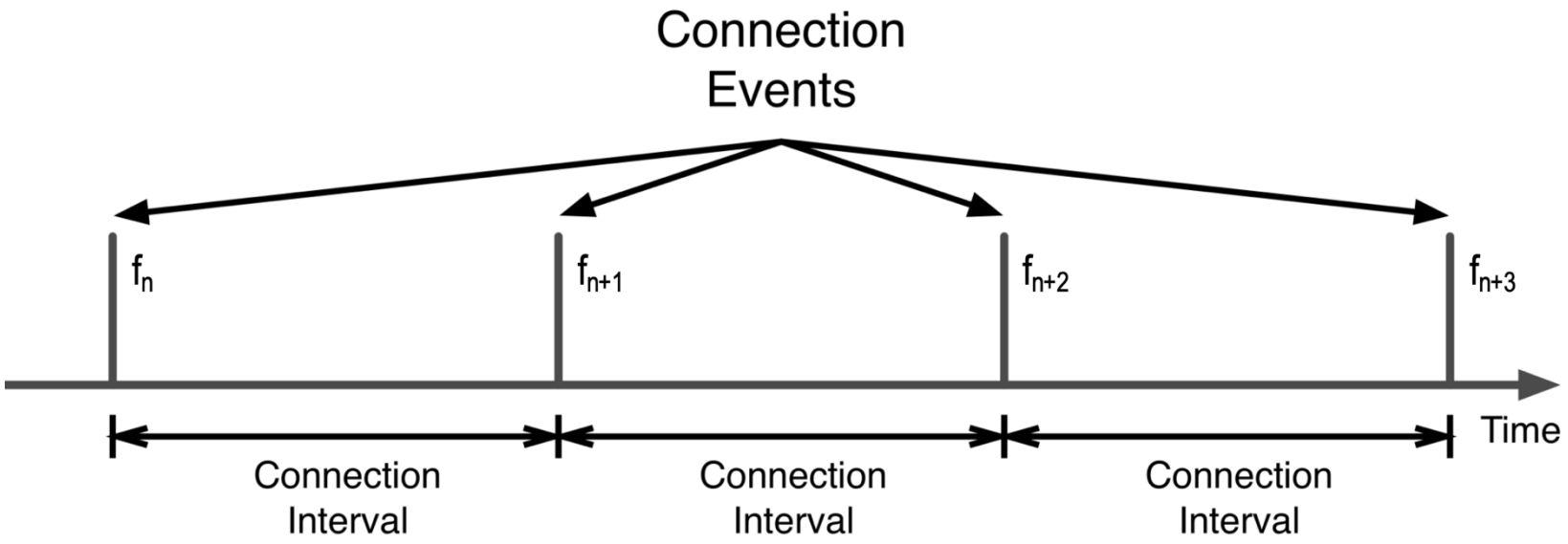


- Uses the 2.4 GHz ISM band
- Divided into 40 channels
  - 37 channels used for connection data
  - 3 channels (37, 38, and 39) are used as advertising channels to set up connections and send broadcast data
- Frequency hopping spread spectrum: radio hops between channels on each connection event using the following scheme:

  next_channel = (channel + hop) mod 37
- The value of the hop is communicated when the connection is established

- At every connection event, a different channel is used

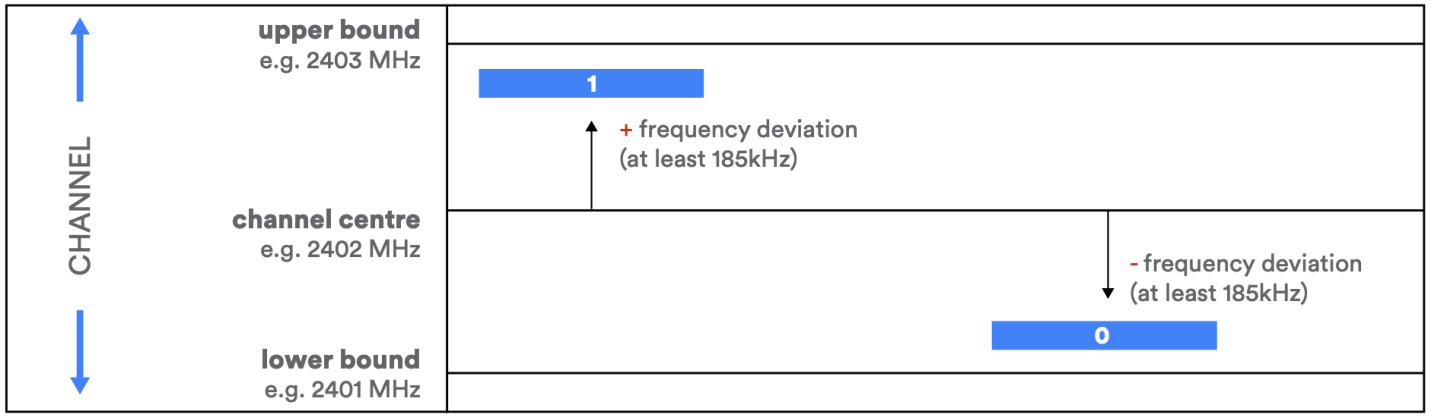
$$f_{n+1} = (f_n + hop) \bmod 37$$



Frequency	LL
2402 MHz	37
2404 MHz	0
2406 MHz	1
2408 MHz	2
2410 MHz	3
2412 MHz	4
2414 MHz	5
2416 MHz	6
2418 MHz	7
2420 MHz	8
2422 MHz	9
2424 MHz	10
2426 MHz	38
2428 MHz	11
2430 MHz	12
2432 MHz	13
2434 MHz	14
2436 MHz	15
2438 MHz	16
2440 MHz	17
2442 MHz	18
2444 MHz	19
2446 MHz	20
2448 MHz	21
2450 MHz	22
2452 MHz	23
2454 MHz	24
2456 MHz	25
2458 MHz	26
2460 MHz	27
2462 MHz	28
2464 MHz	29
2466 MHz	30
2468 MHz	31
2470 MHz	32
2472 MHz	33
2474 MHz	34
2476 MHz	35
2478 MHz	36
2480 MHz	39

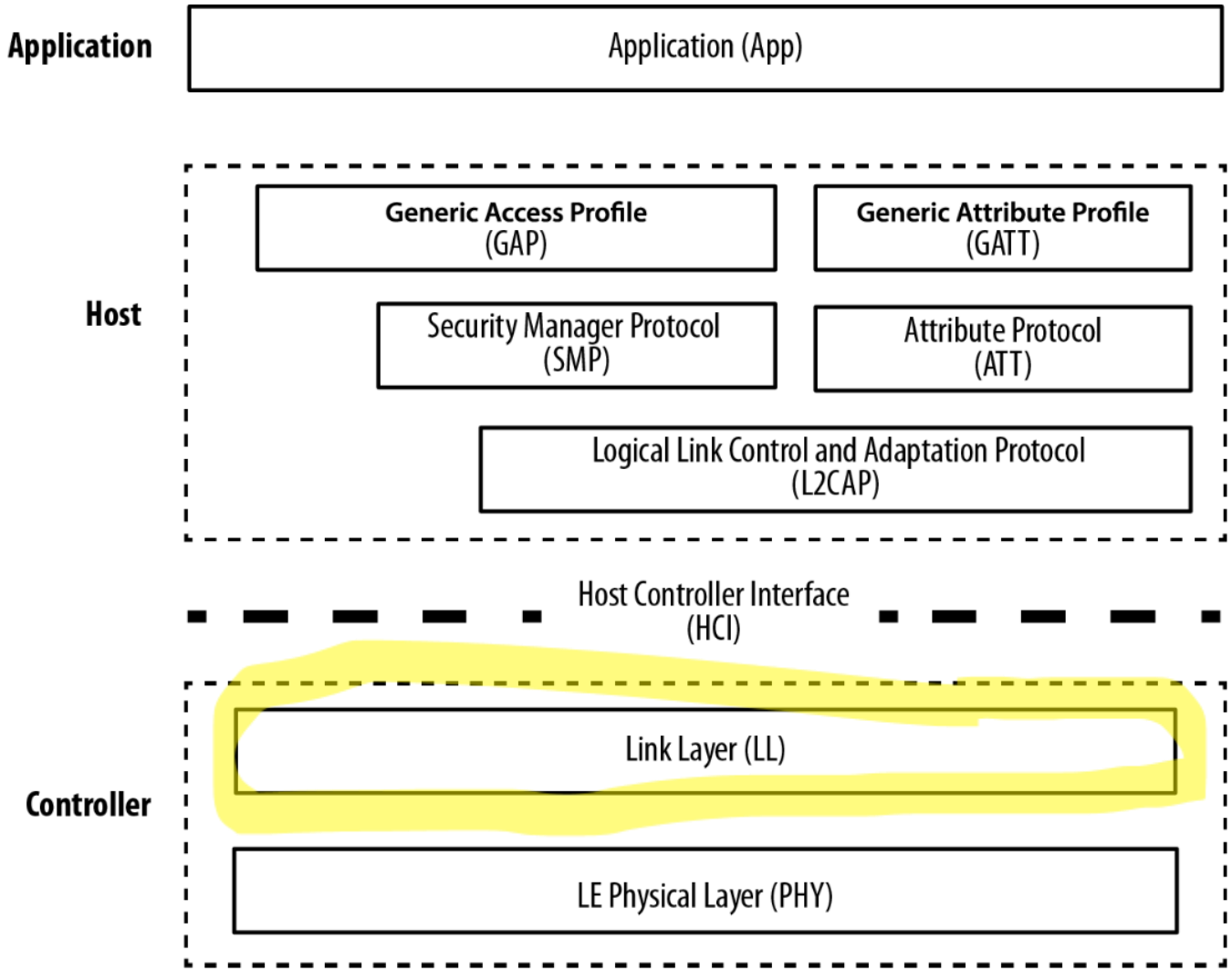
Physical layer

- Modulation scheme called Gaussian Frequency Shift Keying (GFSK)
- GFSK works by taking a signal with the central frequency of the selected channel (the *carrier*) and shifting it up by a specified amount to represent a digital value of 1 or down by the same amount to represent a binary value of 0.



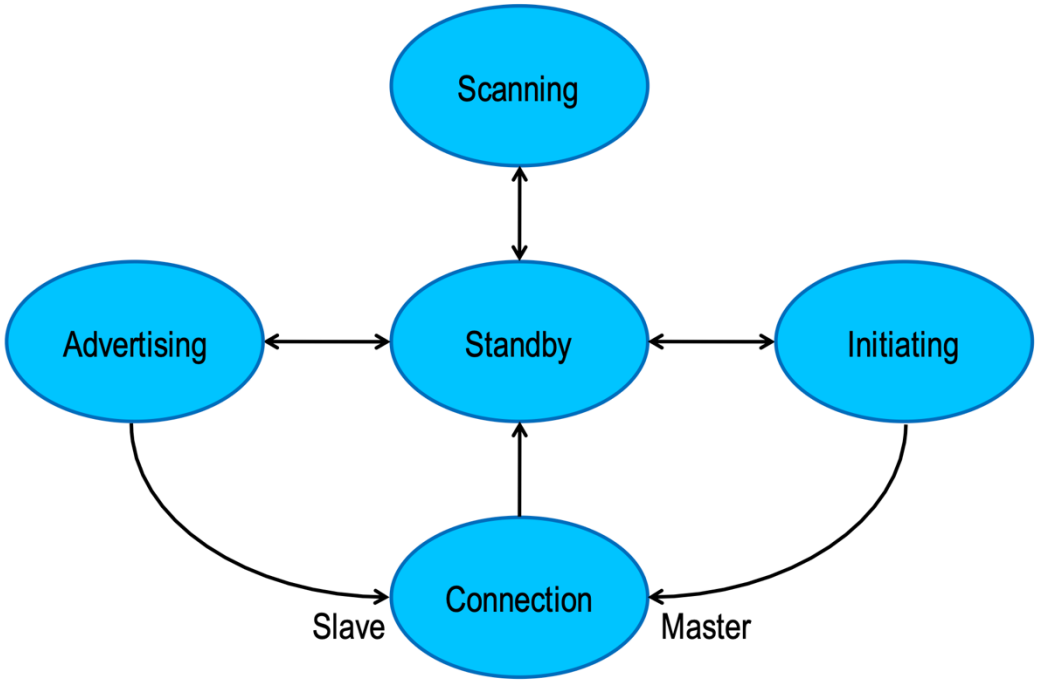
- The LE 1M PHY uses a symbol rate of 1 Msym/s with a required frequency deviation of at least 185 kHz. All devices must support the LE 1M PHY
- The LE 2M PHY is similar to LE 1M but uses a symbol rate of 2 Msym/s and has a required frequency deviation of at least 370 kHz. Support for the LE 2M PHY is optional.

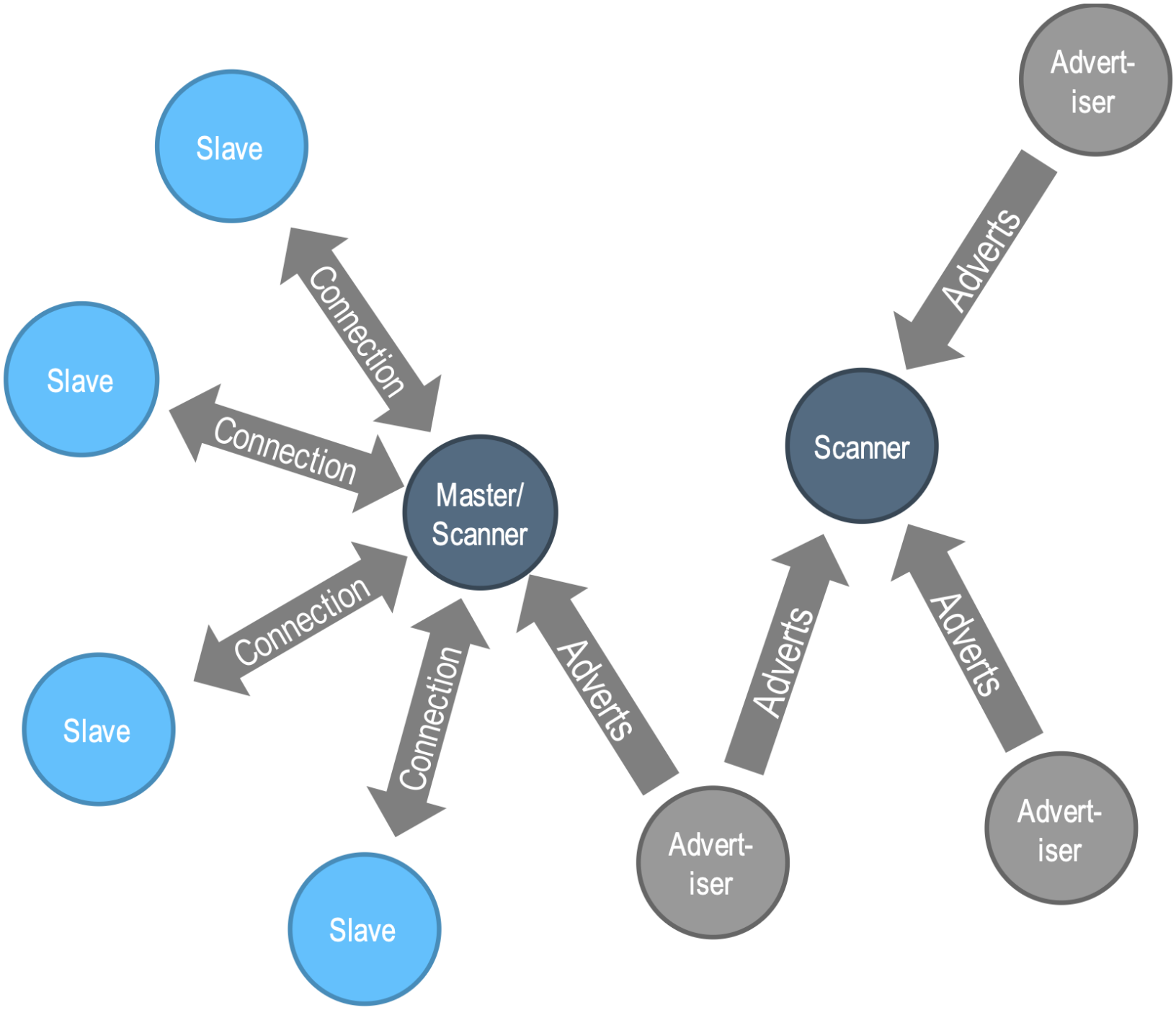
BLE stack

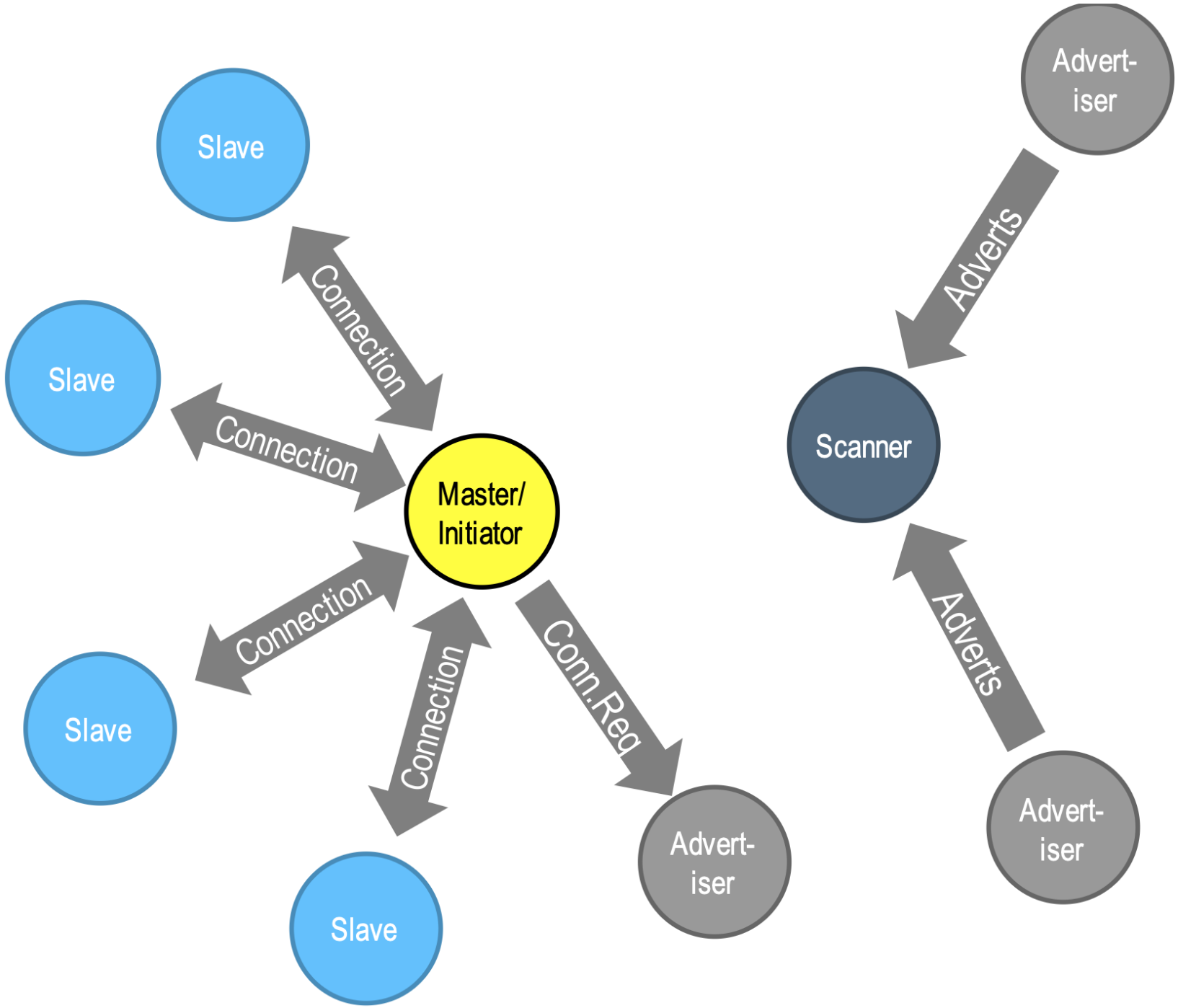

Application
Application (App)
Host
Generic Access Profile
(GAP)
Generic Attribute Profile
(GATT)
Security Manager Protocol
(SMP)
Attribute Protocol
(ATT)
Logical Link Control and Adaptation Protocol
(L2CAP)
Host Controller Interface
(HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

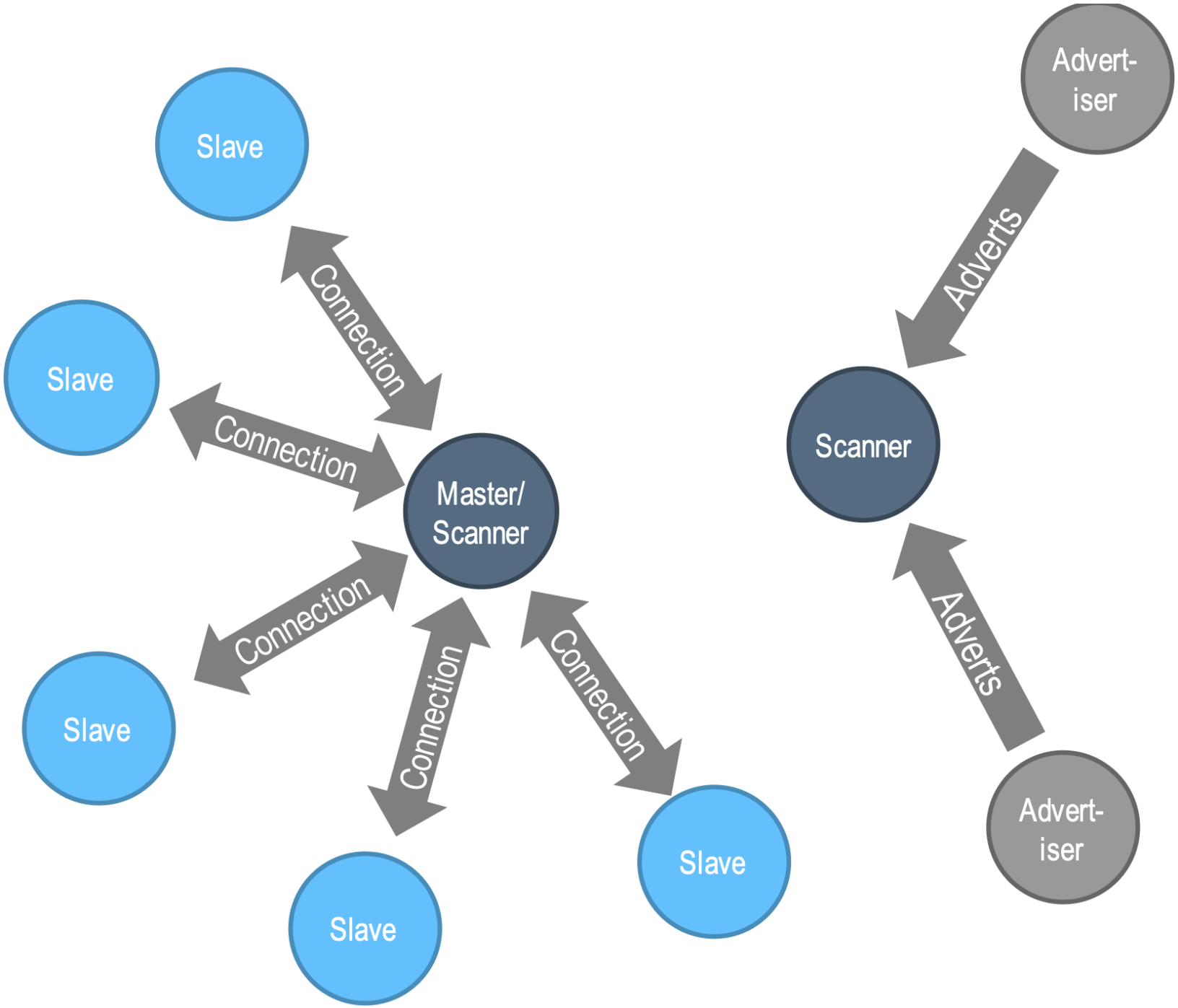
Link layer

- Layer defines the following roles:
  - **Advertiser**: device sending advertising packets
  - **Scanner**: device scanning for advertising packets
  - **Master (Central)**: device that initiates and manages a connection
  - **Slave (Peripheral)**: device that accepts a connection request and follows the master's timing
- Defines format of frames
- Bluetooth device address: 48-bit, uniquely identifies a device among peers
- *Public device address*: equivalent to a fixed, BR/EDR, factory-programmed device address; must be registered with the IEEE, never changes
- *Random device address*: can either be programmed on the device or dynamically generated at runtime




Slave
Slave
Slave
Slave
Connection
Connection
Connection
Connection
Master/
Scanner
Adverts
Adverts
Adverts
Adverts
Scanner
Advert-
iser
Advert-
iser
Advert-
iser


Slave
Slave
Slave
Slave
Connection
Connection
Connection
Connection
Master/
Initiator
Conn.Req
Advert-
iser
Advert-
iser
Adverts
Scanner
Adverts
Advert-
iser

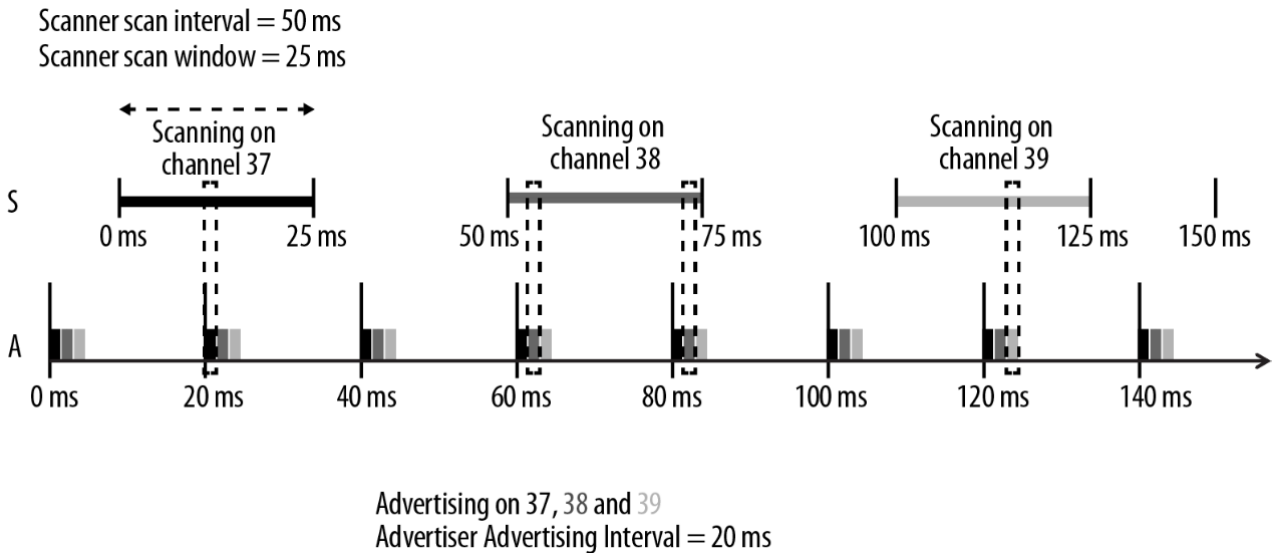

Slave
Slave
Slave
Slave
Slave
Connection
Connection
Connection
Connection
Connection
Master/
Scanner
Advert-
iser
Adverts
Scanner
Adverts
Advert-
iser

Link layer

- Advertisements are sent at a fixed rate defined by the advertising interval, which ranges from 20 ms to 10.24 s
- Advertisements are received successfully by the scanner only when they randomly overlap

- *Connectable*: scanner can start a connection
- *Non-connectable*: scanner cannot start a connection (this packet is meant for broadcast only)

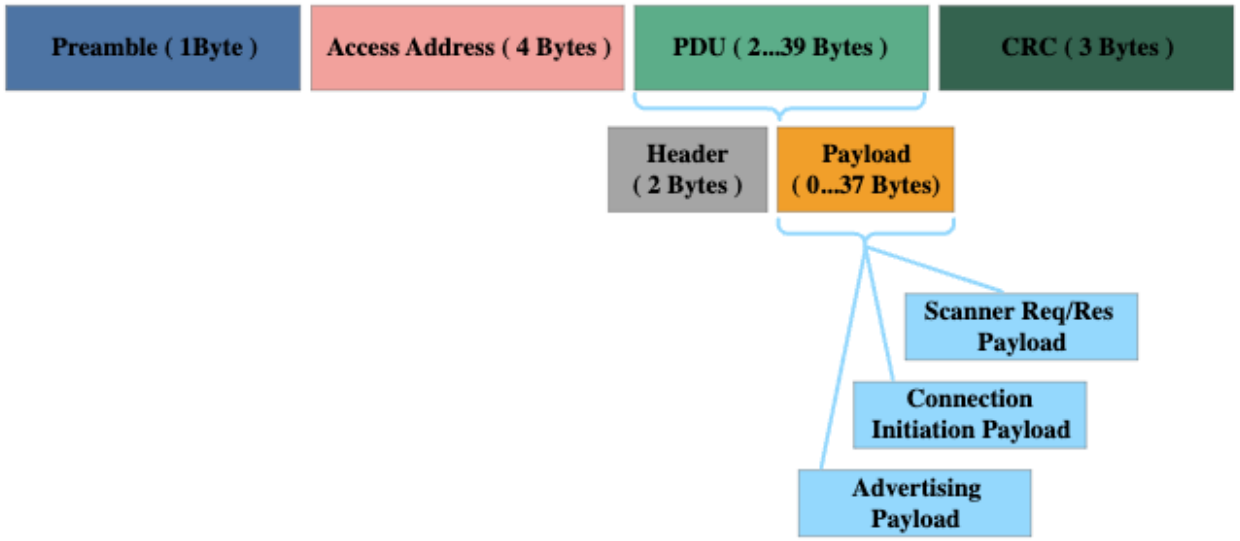
- *Scannable*: scanner can send a scan request upon reception of such an advertising packet
- *Non-scannable*: scanner cannot send a scan request upon reception of such an advertising packet



Link layer

- ADV_IND
  - Advertising Indication (ADV_IND), a peripheral device requests connection to any central device (i.e., not directed at a particular central device).
  - Example: A smart watch requesting connection to any central device.
- ADV_DIRECT_IND
  - Similar to ADV_IND, but the connection request is directed at a specific central device.
  - Example: A smart watch requesting connection to a specific central device.
- ADV_NONCONN_IND
  - Non connectable devices, advertising information to any listening device.
  - Example: Beacons in museums defining proximity to specific exhibits.
- ADV_SCAN_IND
  - Similar to ADV_NONCONN_IND, with the option additional information via scan responses.
  - Example: A warehouse pallet beacon allowing a central device to request additional information about the pallet.

LL, Advertisements



- LL packet format
  - AA: 0x8E89BED6, random generated
- Protocol Data Unit: 2B header + payload (up to 37 B
- Standard advertising packet contains a max of 31-byte payload (37 – 6 B for address)
- Include data that describes the broadcaster and its capabilities
- Can also include any custom information you want to broadcast to other devices
- ADV_IND:

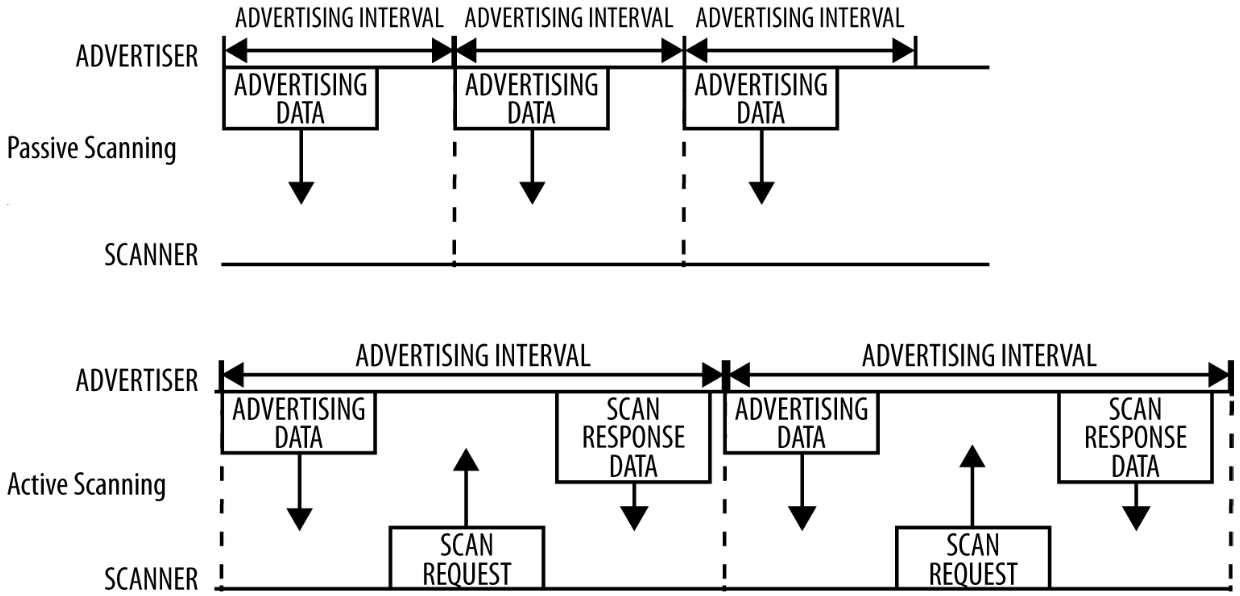
Payload	
Advertisement Address (AdvA) (6 Bytes)	Advertisement Data (AdvData) (0 to 31 Bytes)

In PDU header: 4 bits for PDU type, 7 types of adv

Packet	Usage
ADV_IND	Connectable undirected advertising event
ADV_DIRECT_IND	Connectable directed advertising event
ADV_NONCONN_IND	Non-connectable undirected advertising event
ADV_SCAN_IND	Scannable undirected advertising event
SCAN_RSP	Response to Scan Request from Scanner
CONNECT_REQ	Connect Request by Initiator
SCAN_REQ	Scan Request for further information from Advertiser

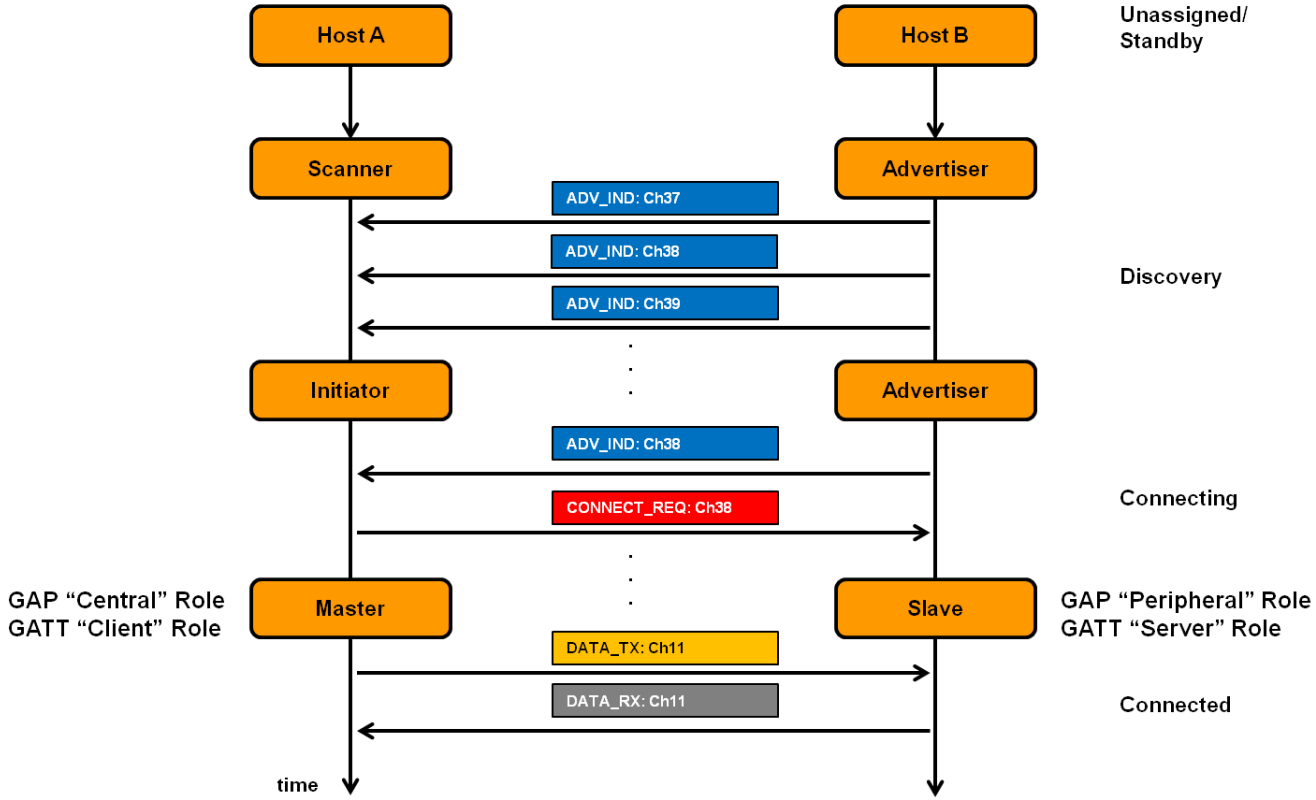
Advertiser/scanner

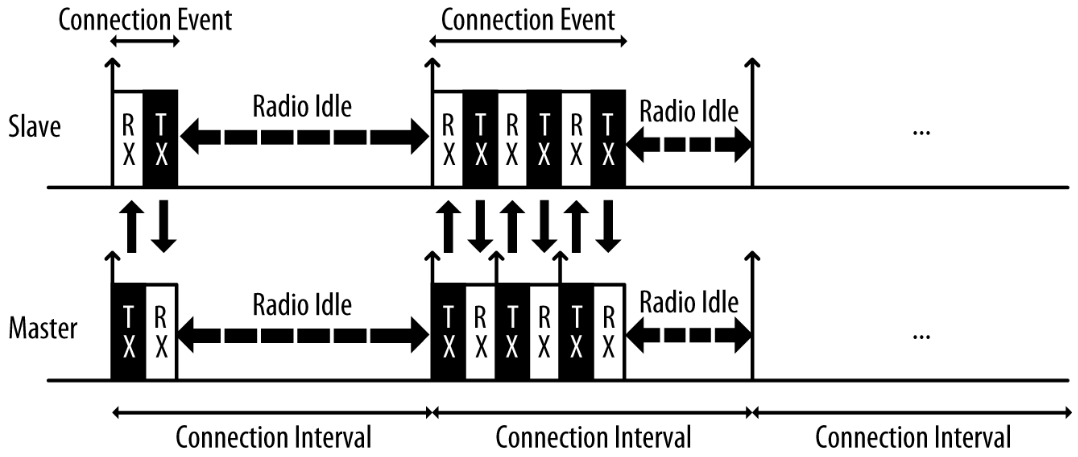
- If 31-byte payload is not enough: secondary advertising payload (*Scan Response*) to request a second advertising frame with another 31-byte payload (up to 62 bytes total)
- *Passive scanning*:
  - scanner listens for advertising packets
  - advertiser is never aware of the fact that one or more packets were received by a scanner
- *Active scanning*
  - scanner sends a *Scan Request* after receiving an advertising packet
  - advertiser responds with a *Scan Response* packet
  - Communication is still unidirectional



LL, Connections

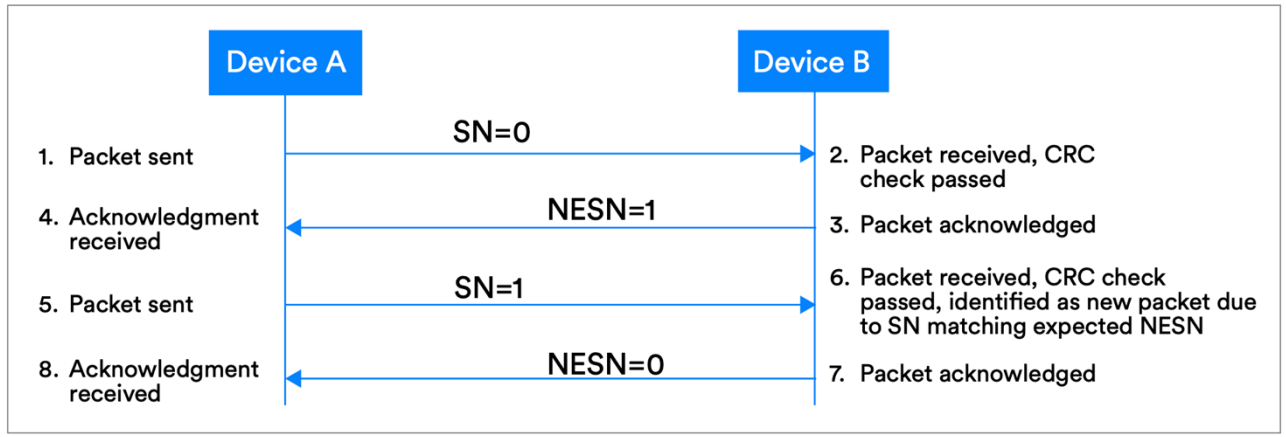
- Data packets are used to transport user data bidirectionally between master and slave
- Usable data payload of 27 bytes, but additional protocols limit the actual amount of user data to 20 bytes per packet
- Link layer is reliable
  - all packets received are checked against a 24-bit CRC
  - retransmissions are requested when the error checking detects a transmission failure

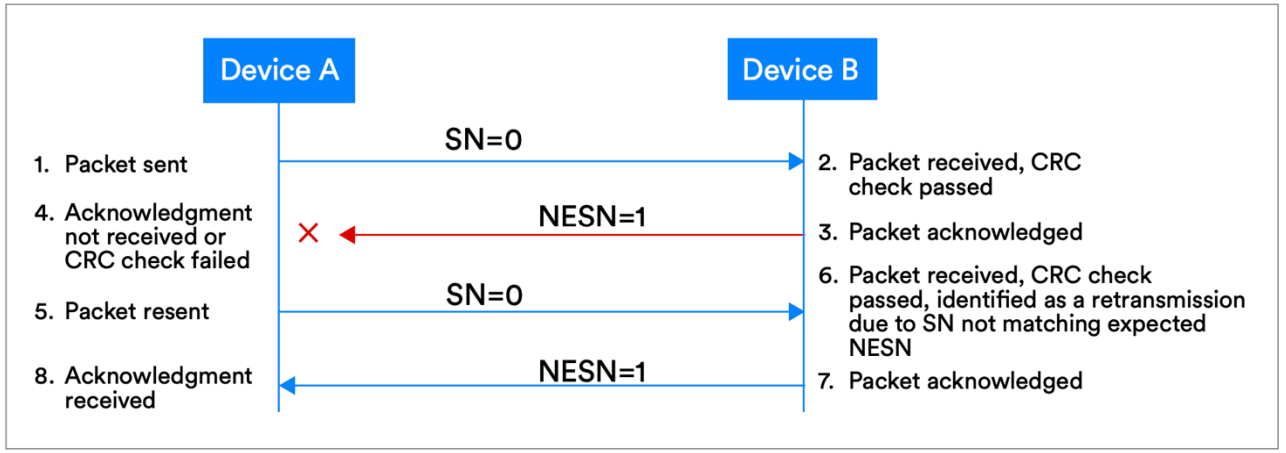


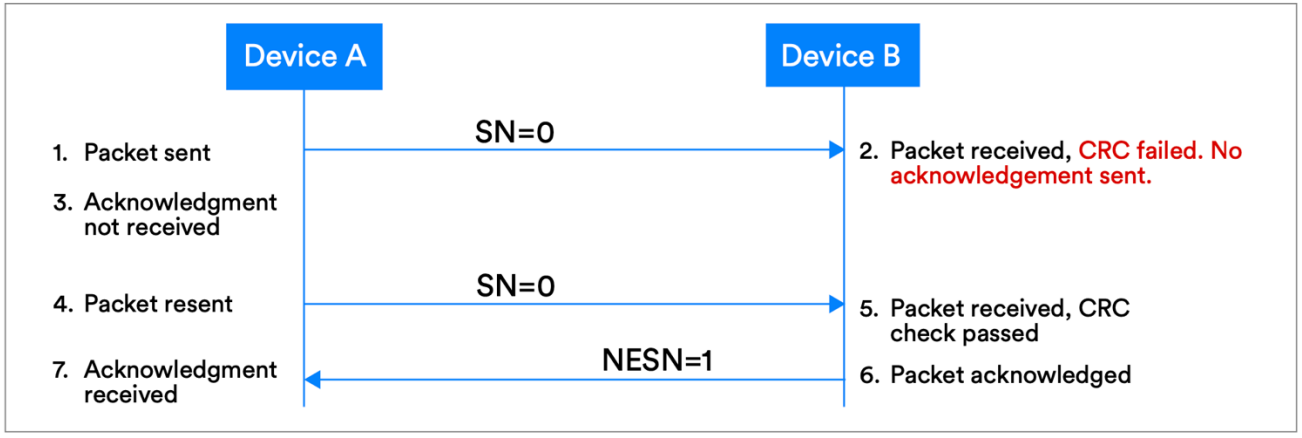


LL: acknowledgments

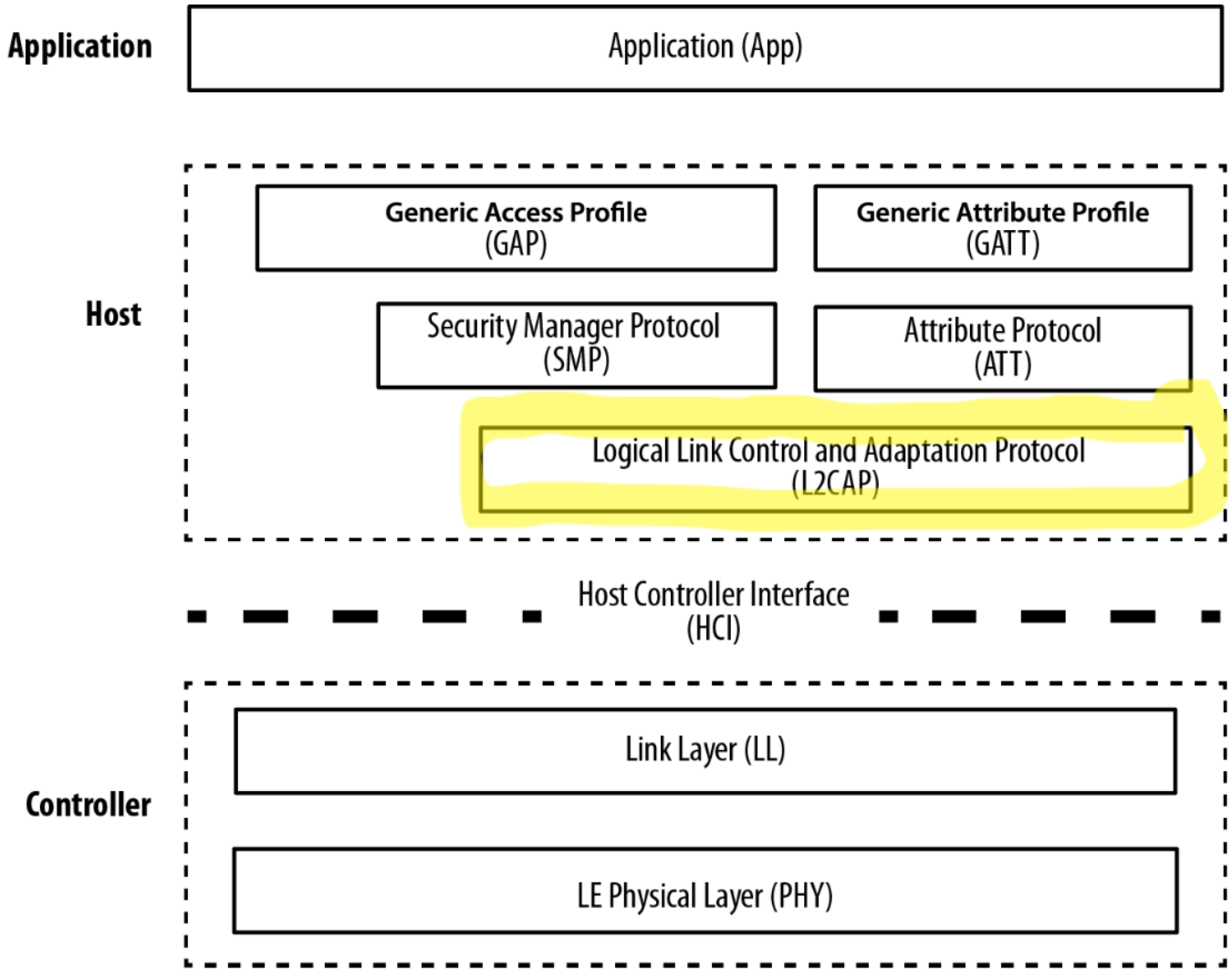
- Data packets contain fields (single bits) Sequence Number (SN), Next Expected Sequence Number (NESN)







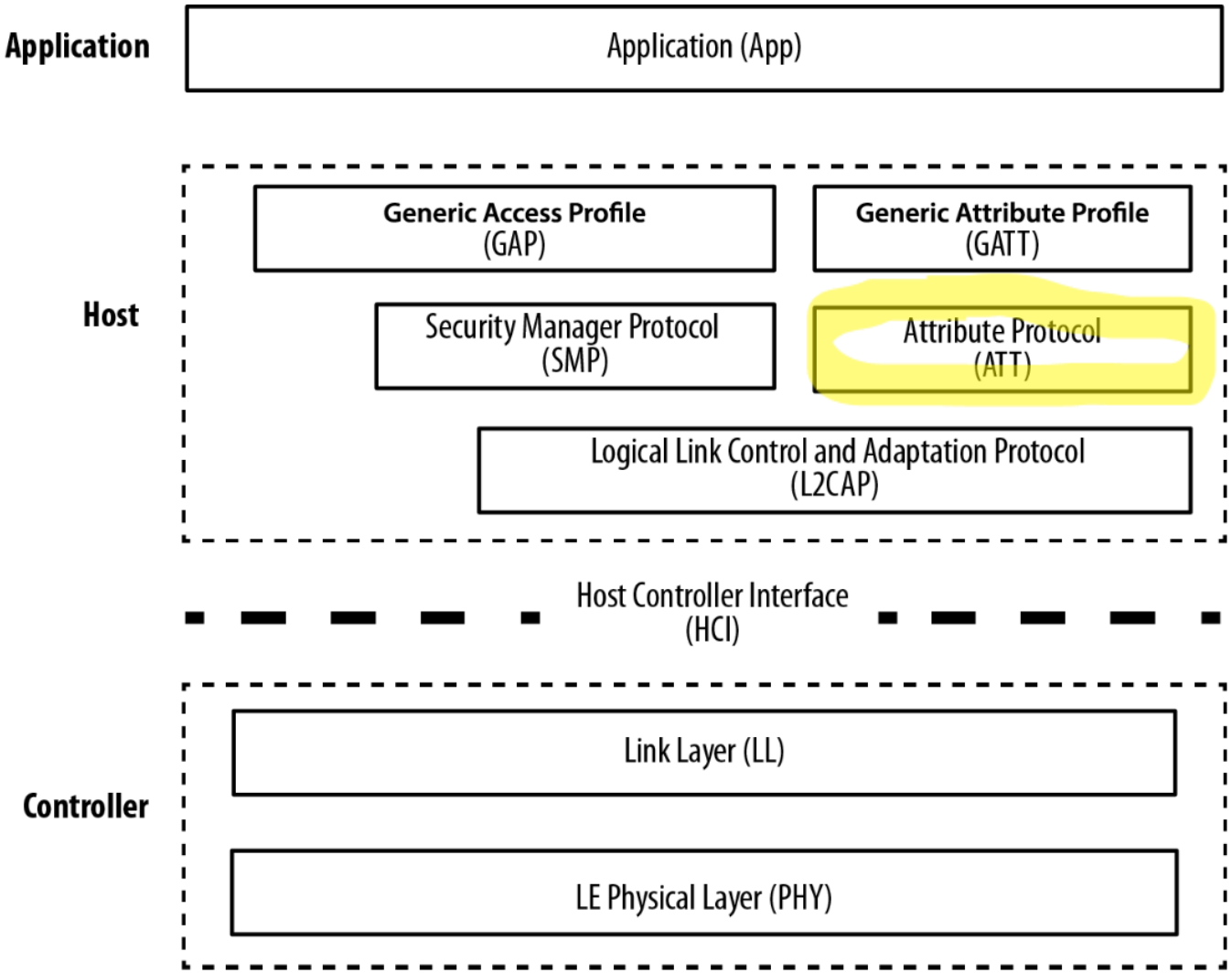
BLE stack


Application
Application (App)
Host
Generic Access Profile
(GAP)
Generic Attribute Profile
(GATT)
Security Manager Protocol
(SMP)
Attribute Protocol
(ATT)
Logical Link Control and Adaptation Protocol
(L2CAP)
Host Controller Interface
(HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

L2CAP

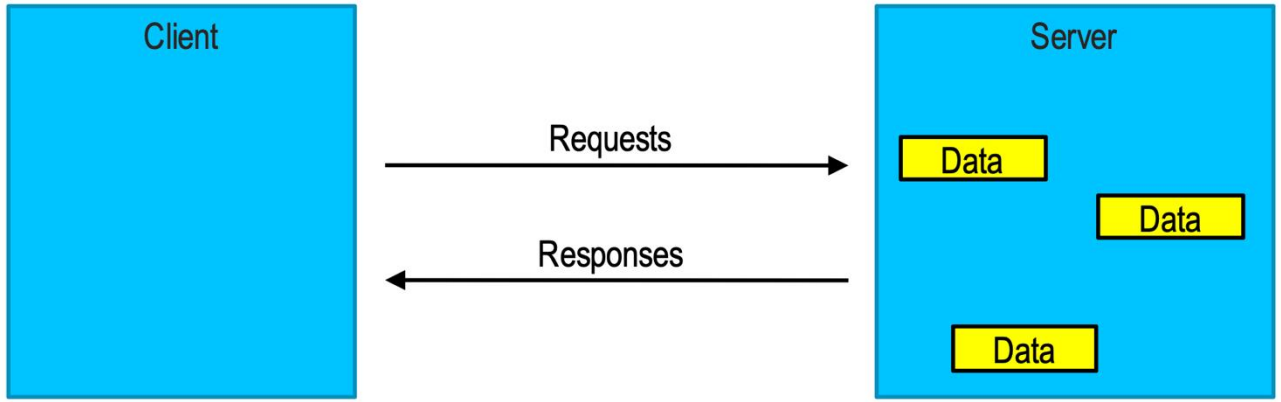
- Logical Link Control and Adaptation Protocol (L2CAP)
- Main functionalities
  - Multiplexing/demultiplexing from/to upper layers
  - Fragmentation/reassembling
- L2CAP packet header takes up four bytes, which means that the effective user payload length is 27 - 4 = 23 bytes
- Interacts with upper-level protocols
  - Attribute Protocol (ATT)
  - Security Manager Protocol (SMP)

BLE stack


Application
Application (App)
Host
Generic Access Profile
(GAP)
Generic Attribute Profile
(GATT)
Security Manager Protocol
(SMP)
Attribute Protocol
(ATT)
Logical Link Control and Adaptation Protocol
(L2CAP)
Host Controller Interface
(HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

ATT

- Attribute Protocol (ATT) is a simple client/server stateless protocol based on attributes presented by a device
  - client requests data from a server
  - server sends data to clients
- Servers contains data organized in the form of attributes



ATT

- Each attribute has
  - a 16-bit attribute handle
  - a universally unique identifier (UUID)
  - a set of permissions
  - a value
- The attribute handle is an identifier used to access an attribute value
- The UUID specifies the type and nature of the data contained in the value

ATT

- Attributes have values
  - array of octets
  - 0 to 512 octets in length
  - can be fixed or variable length

Value
0x54656d70657261747572652053656e736f72
0x04
0x0802

ATT

- Each attribute has a “handle”
  - used to address an individual attribute by a client
  - Read (0x0022) => 0x04 ; Read (0x0098) => 0x0802

Handle	Value
0x0009	0x54656d70657261747572652053656e736f72
0x0022	0x04
0x0098	0x0802

ATT

- Attributes have a type
  - type is a «UUID», determines what the value means
  - Types are defined by “Characteristic Specifications”
  - or Generic Access Profile or Generic Attribute Profile

Handle	Type	Value
0x0009	«Device Name»	0x54656d7065726174757265205365e736f72
0x0022	«Battery State»	0x04
0x0098	«Temperature»	0x0802

ATT

- «Device Name»
  - defined by GAP
  - formatted as UTF-8
  - 0x54656d7065726174757265205365e736f72 =
  - “Temperature Sensor

Handle	Type	Value
0x0009	«Device Name»	“Temperature Sensor”
0x0022	«Battery State»	0x04
0x0098	«Temperature»	0x0802

ATT

- «Battery State»
  - defined by “Battery State Characteristic” specification
  - enumerated value
  - 0x04 = Discharging

Handle	Type	Value
0x0009	«Device Name»	“Temperature Sensor”
0x0022	«Battery State»	Discharging
0x0098	«Temperature»	0x0802

ATT

- «Temperature»
  - defined by “Temperature Characteristic” specification
  - Signed 16 bit Integer in 0.01 ºC
  - 0x0802 = 2050 * 0.01 ºC = 20.5 ºC

Handle	Type	Value
0x0009	«Device Name»	“Temperature Sensor”
0x0022	«Battery State»	Discharging
0x0098	«Temperature»	20.5 ºC

ATT

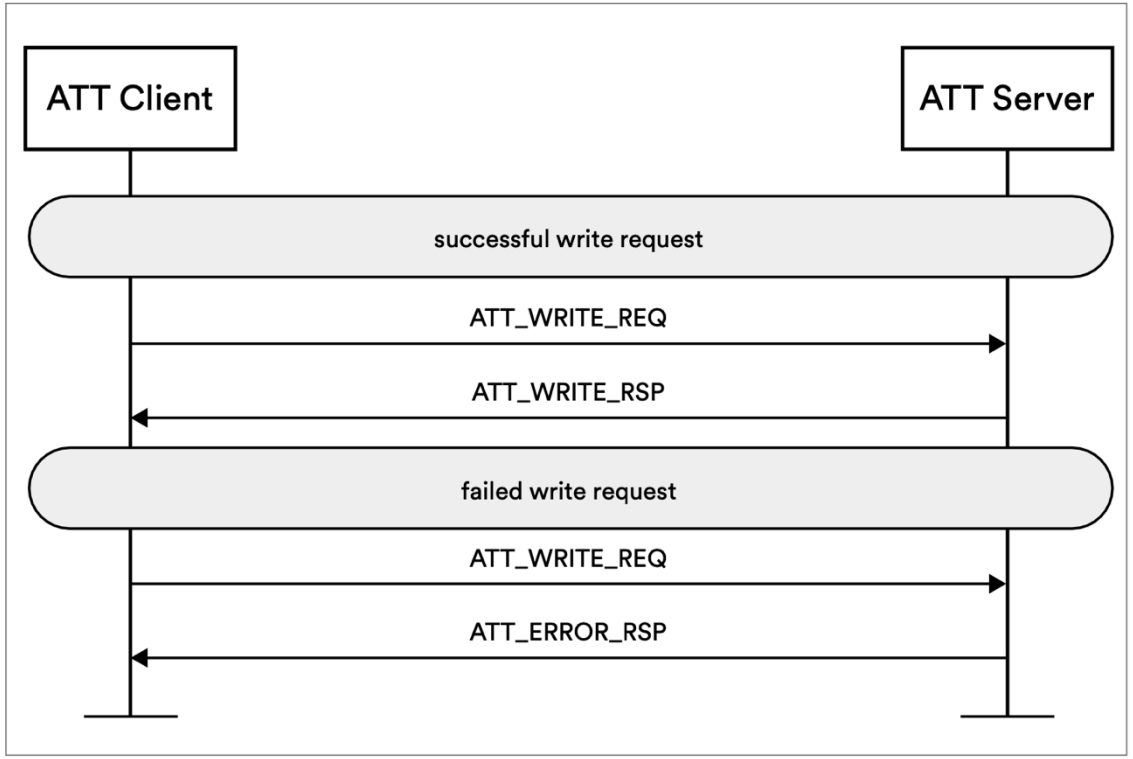
- The handle is a sort of row number in a table
- The value depends on the attribute type
- The attribute type is represented by a UUID

Heart Rate Profile	Handle	Type of attribute (UUID)	Attribute permission	Attribute value
Service Declaration	0x000E	**Service declaration** Standard UUIDservice 0x2800	Read Only, No Authentication, No Authorization	Heart Rate Service 0x180D
Characteristic Declaration	0x000F	**Characteristic declaration** Standard UUIDcharacteristic 0x2803	Read Only, No Authentication, No Authorization	Properties (Notify) Value Handle (0x0010) UUID for Heart Rate Measurement characteristic (0x2A37)
Characteristic Value Declaration	0x0010	**Heart Rate Measurement Characteristic** UUID found in the Characteristic declaration value 0x2A37	Higher layer profile or implementation specific.	Beats Per Minute E.g "167"

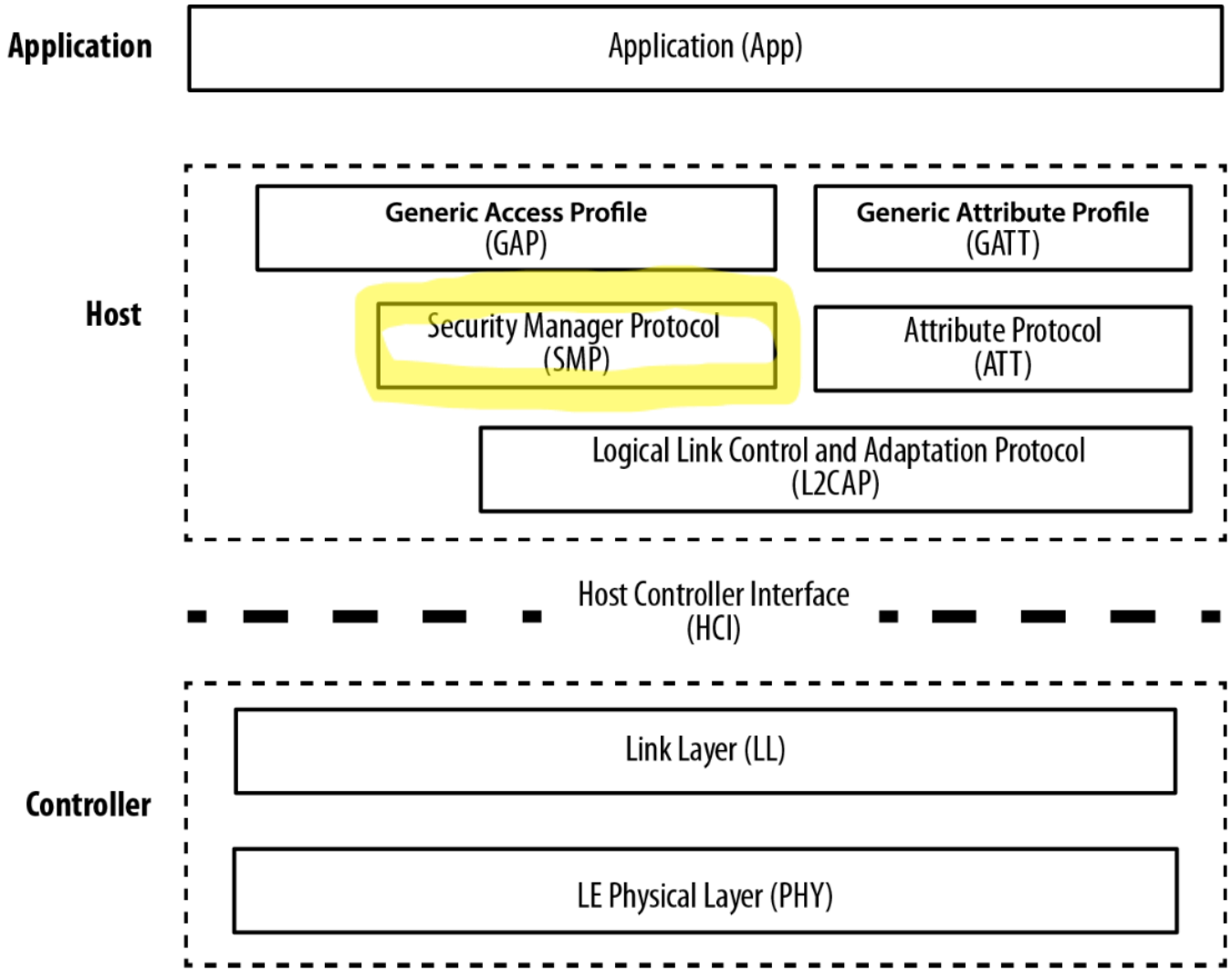
Source: Nordic semiconductor

ATT

- The ATT protocol defines 30+ PDUs
- Sometimes client initiated, other times server initiated
- Can be with response or w/o resp



BLE stack


Application
Application (App)
Host
Generic Access Profile
(GAP)
Generic Attribute Profile
(GATT)
Security Manager Protocol
(SMP)
Attribute Protocol
(ATT)
Logical Link Control and Adaptation Protocol
(L2CAP)
Host Controller Interface
(HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

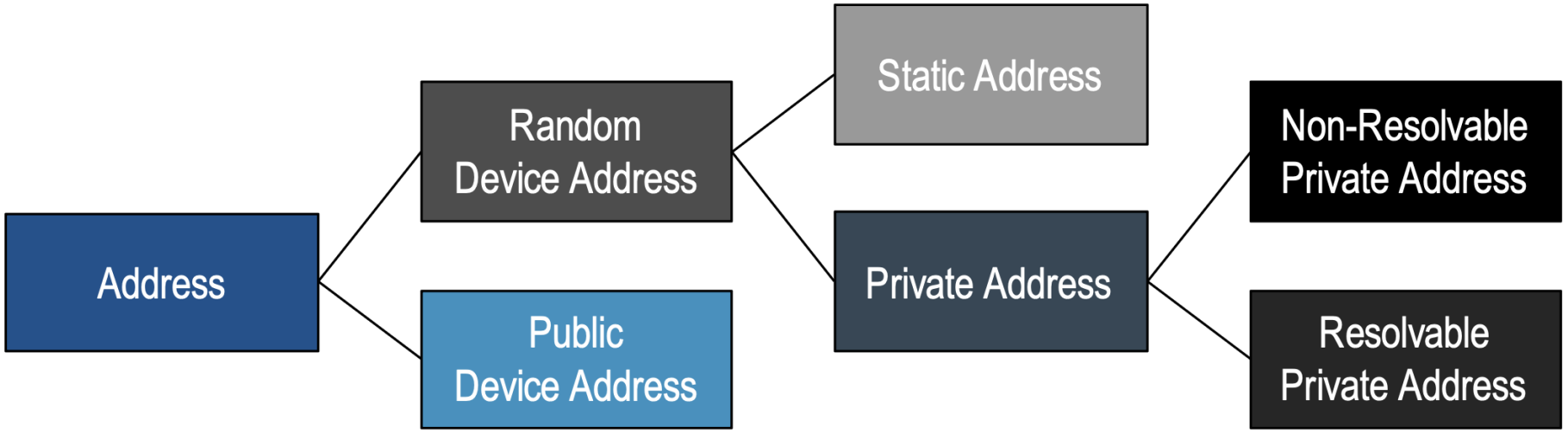
Security manager

- Defines security procedures
- **Pairing**:
  - a temporary common security encryption key is generated
  - switch to a secure, encrypted link
  - temporary key is not stored and is not reusable in future connections
- **Bonding**:
  - pairing + the generation and exchange of permanent security keys
  - stored in nonvolatile memory, and creating a permanent bond between two devices
  - allows to create a secure link in subsequent connections without having to perform a bonding again
- **Encryption re-establishment**: after bonding
  - keys generated in bonding are stored on both sides of the connection
  - defines how to use keys in subsequent connections to re-establish a secure, encrypted connection without having to go through pairing and bonding again

Pairing

- Possible pairing procedures
- **Just works**:
  - STK is generated on both sides, based on the packets exchanged in plain text
  - no security against MITM attacks
- **Passkey display**:
  - one of the peers displays a randomly generated, six-digit passkey
  - the other side is asked to enter it
  - protects against MIM attacks
- **Out of band**:
  - additional data is transferred without using BLE, such as NFC
  - protects against MIM attacks

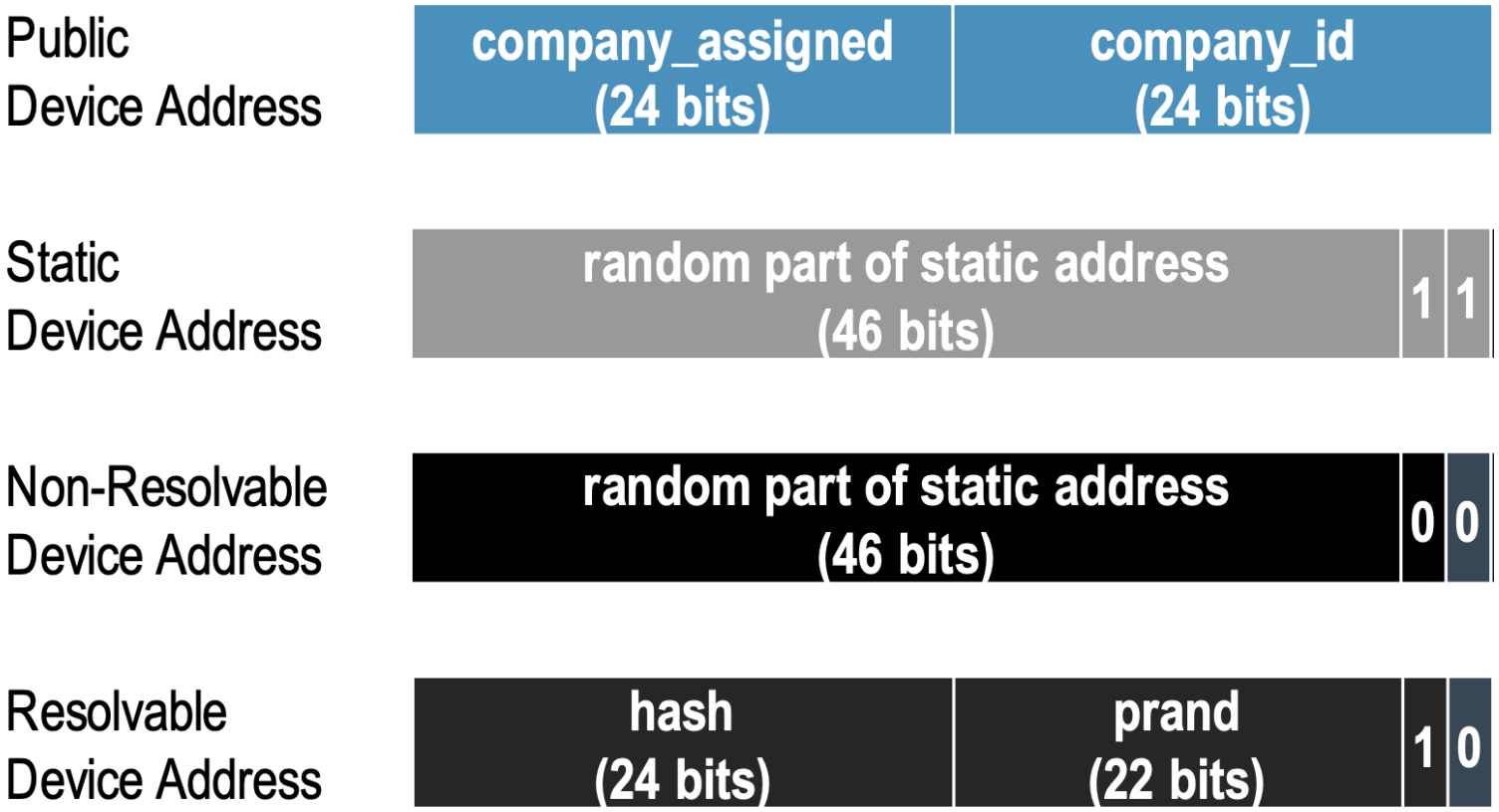
Addresses


Address
Random Device Address
Public Device Address
Static Address
Private Address
Non-Resolvable Private Address
Resolvable Private Address

Link layer: random addresses

- **Static addresses**: A static device address is randomly generated but in no way disguised in order to safeguard the privacy of the device. A device using a static address is permitted to regenerate the address every time it is power-cycled but is not required to do so. It is not permitted to change its address at any other time.
- **Non-resolvable Private Addresses**: a non-resolvable private address is randomly generated and typically changes at each reconnection. It can be used to offer privacy but without the cost associated with processing resolvable private addresses
- **Resolvable Private Addresses**: A resolvable private address (RPA) is changed periodically. The Bluetooth Core Specification recommends an interval of about 15 minutes but ultimately the timing governing RPA changes is an implementation decision.
  - The process involves a security key called the Identity Resolution Key (IRK) which is exchanged between devices when they are bonded and the application of a hash function.
  - A bonded peer device may resolve an RPA in a received packet by applying the same hash function with each of the IRK values it possesses from the devices it has bonded with, one at a time.
  - When the process yields a match with the received RPA the peer knows it has resolved the address and the true identity of the remote device.
  - Devices which have not bonded with the transmitting device cannot resolve the RPA.

Addresses


Public
Device Address
company_assigned
(24 bits)
company_id
(24 bits)
Static
Device Address
random part of static address
(46 bits)
1
1
Non-Resolvable
Device Address
random part of static address
(46 bits)
0
0
Resolvable
Device Address
hash
(24 bits)
prand
(22 bits)
1
0

Addresses

hash = func (IRK, prand)

Identity Resolving Key (**IRK**)= 128-bit key used for generating a private Bluetooth address of the devices involved in the pairing. Shared beteen devices.

When a device receives a packet with an RDA, it uses its stored IRKs to resolve the RDA and understand the identity of sending device.

The "*func*" hash function is executed using

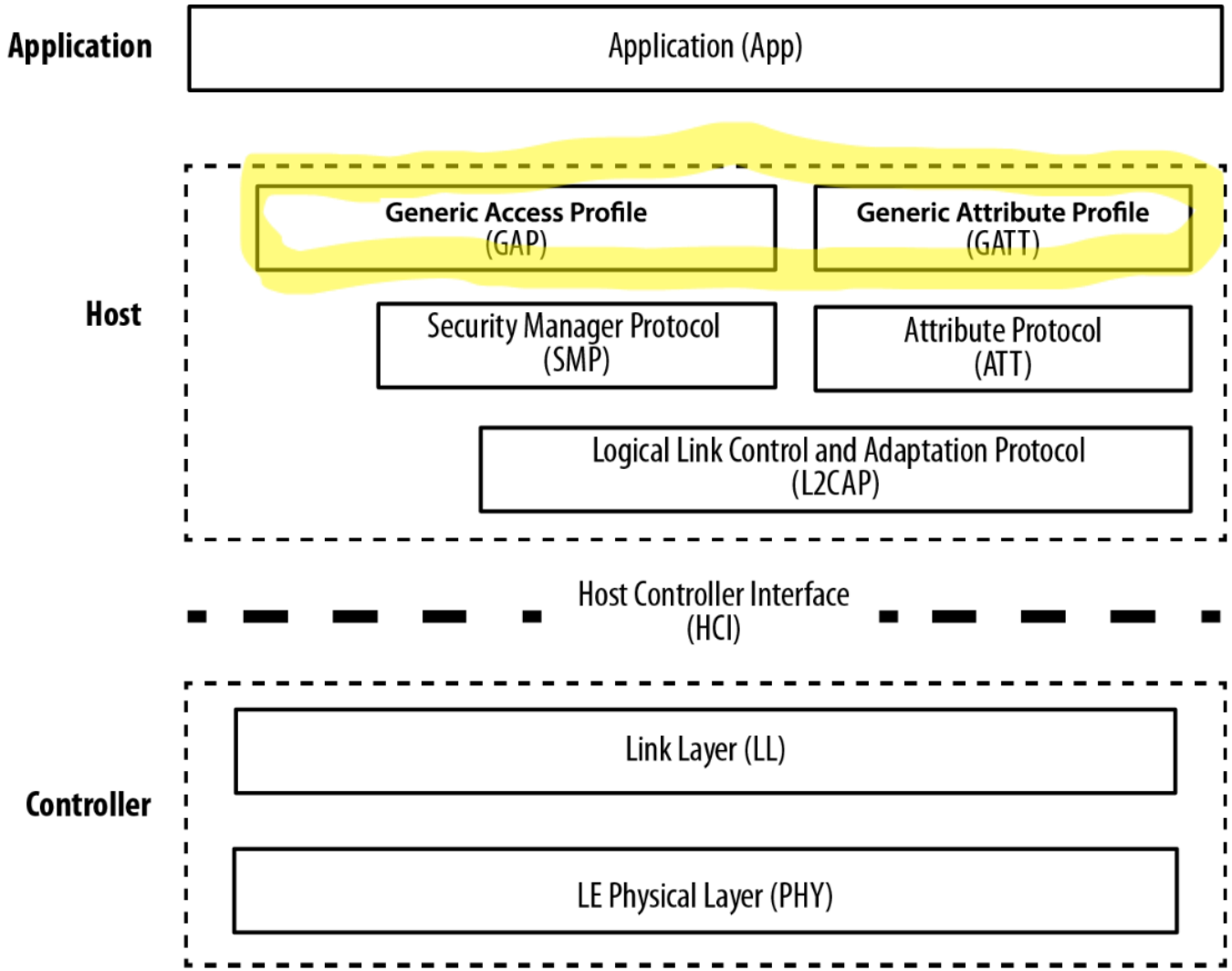
- the received, *prand*
- each stored IRK.

If match with hash device identity is determined.

Link layer: filter accept list

- The link layer includes a feature known as the Filter Accept List.
- This list may be populated with the addresses of devices which the host is interested in receiving packets from.
- One benefit of the Filter Accept List is that it allows the controller to efficiently select those packets that need to be passed up the stack to the host and to discard those which do not.
- Resolvable private addresses may be added to the Filter Accept List but their processing by the controller incurs a cost. Since an RPA will change periodically, it is the resolved address that is stored in the Filter Accept List when necessary and therefore to check for the presence of an RPA in the list, the controller must first resolve the received RPA.

BLE stack


Application
Application (App)
Host
Generic Access Profile
(GAP)
Generic Attribute Profile
(GATT)
Security Manager Protocol
(SMP)
Attribute Protocol
(ATT)
Logical Link Control and Adaptation Protocol
(L2CAP)
Host Controller Interface
(HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

GAP and GATT

- **GAP**: defines roles, procedures to broadcast data, discover devices, establish and manage connections, and negotiate security levels
  - Roles: broadcaster/observer, central/peripheral
  - Discoverable/non discoverable
  - Connectable/non connectable
- **GATT**: defines a data model and procedures to discover, read, write, and push data
  - Data is organized in services
- Example: heart rate monitor paired with a smartphone
  - HRM:
    - GAP role is peripheral
    - acts as a GATT server when the phone requests data from sensors
    - sometimes acts as a GATT client when it requests configuration information from the smartphone
  - Smartphone
    - GAP role is central
    - Generally a GATT client, but sometimes can be GATT server
- The GATT client/server roles depend exclusively on the direction in which the data requests and responses flow
- GAP always peripheral for the HRM and central for the smartphone

UUID

- A universally unique identifier (UUID) is a 128-bit (16 bytes) number that is guaranteed (or has a high probability) to be globally unique

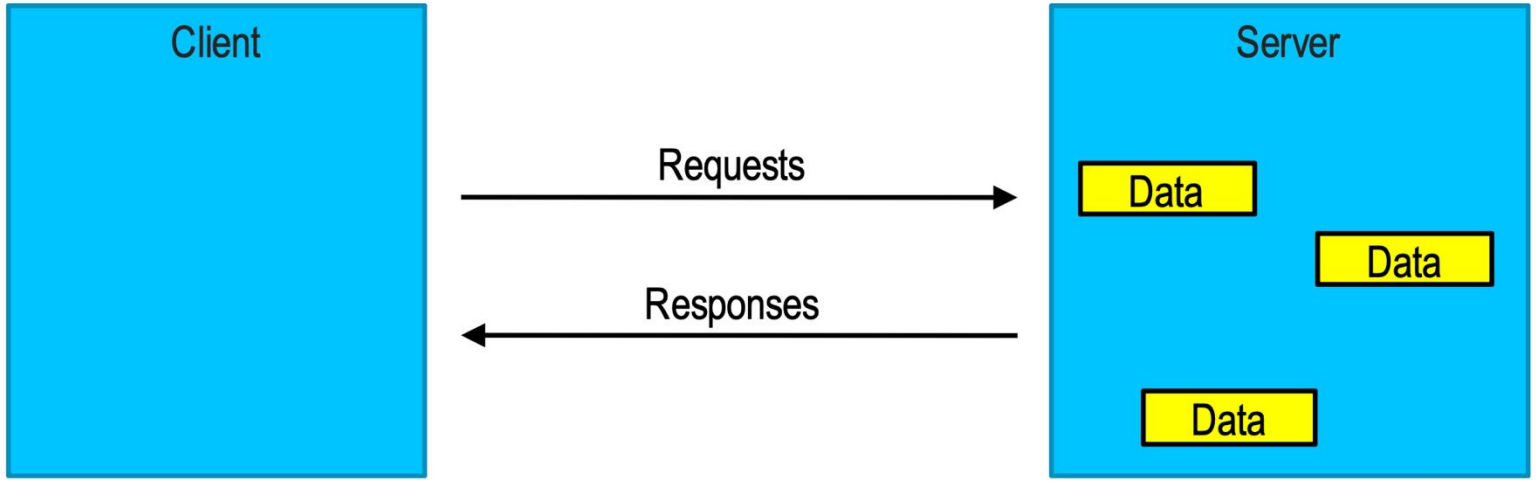
517f05c4-67b5-4983-ad3b-61f911bd29b5

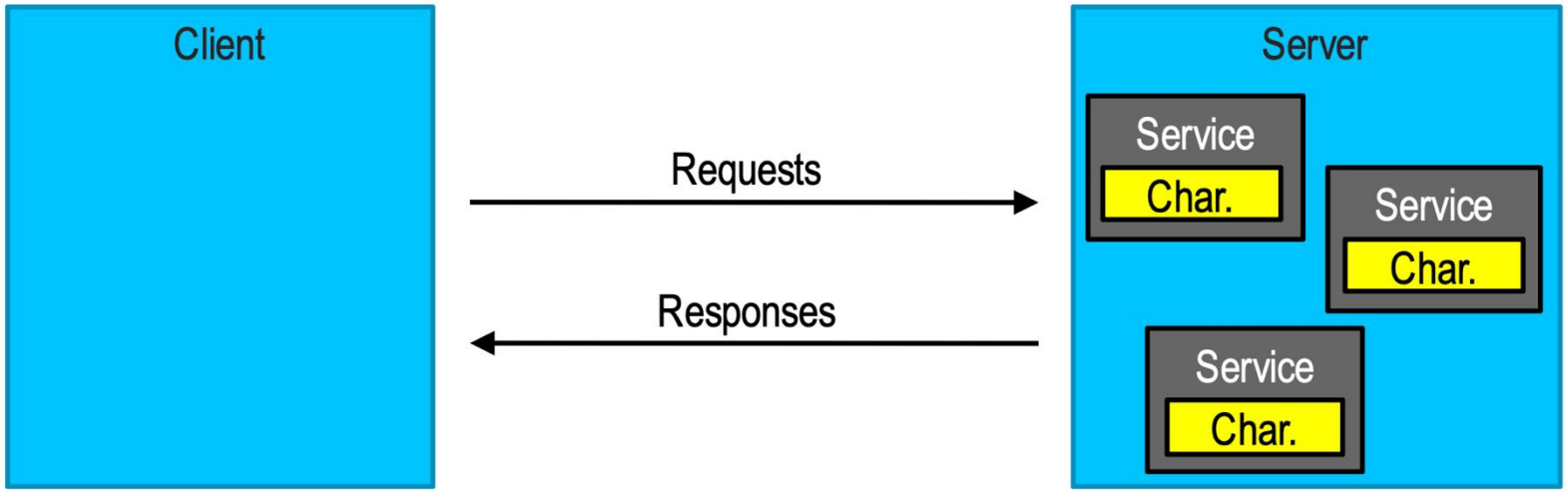
- 16 bytes can be large for BLE small packets
- Specification adds short 16-bit UUID format, e.g. 0x180c
- Short formats can be used only with UUIDs that are defined in the BLE specification
- To obtain the 128-bit UUID from 16-bit ones:

`0000xxxx-0000-1000-8000-00805F9B34FB`

GATT

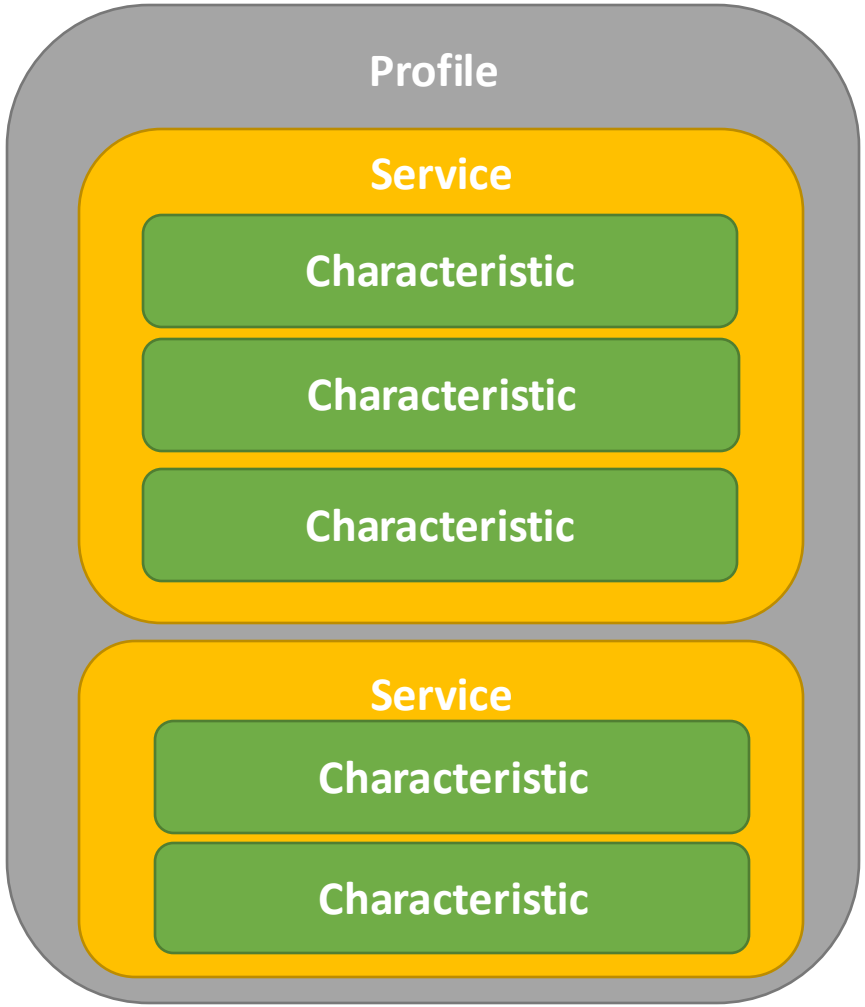
- Same client server architecture as ATT
- but data is encapsulated in “Services”
- and data is exposed in “Characteristic





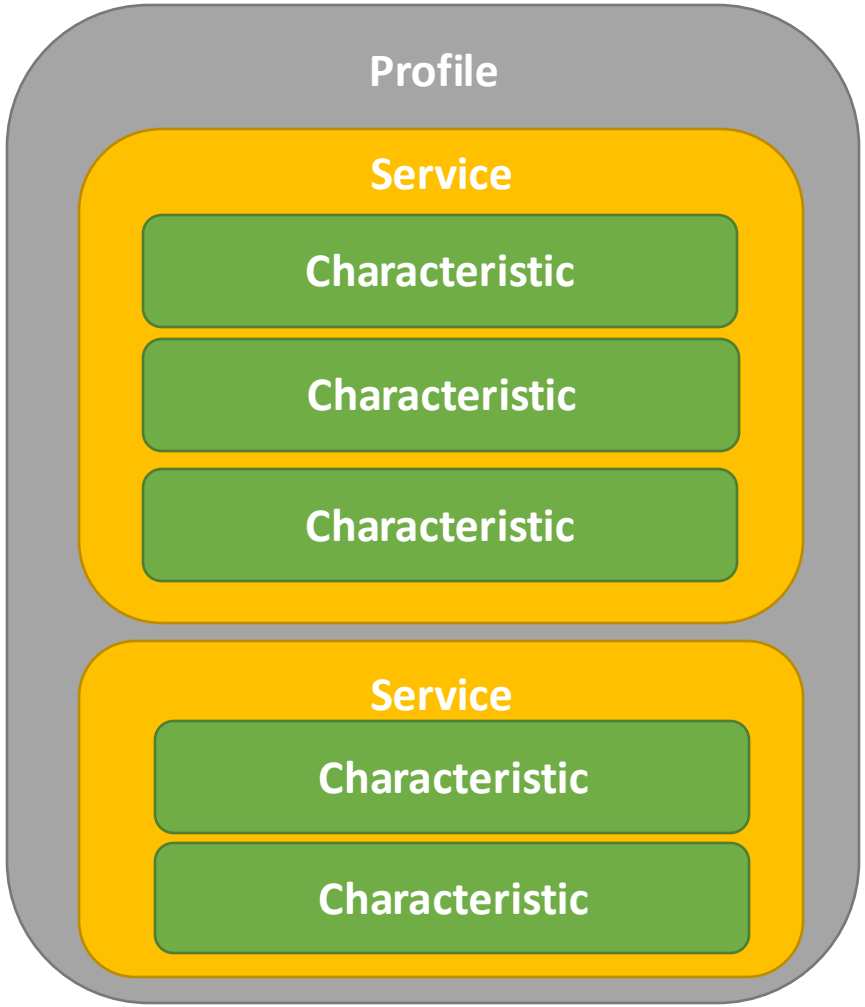
GATT profiles, services, characteristics

- GATT organizes information in profiles, services, and characteristics
- **Profile**: a collection of Services that has been defined by either the Bluetooth SIG or by the peripheral vendor
  - The *Heart Rate Profile* combines the *Heart Rate Service* and the *Device Information Service*



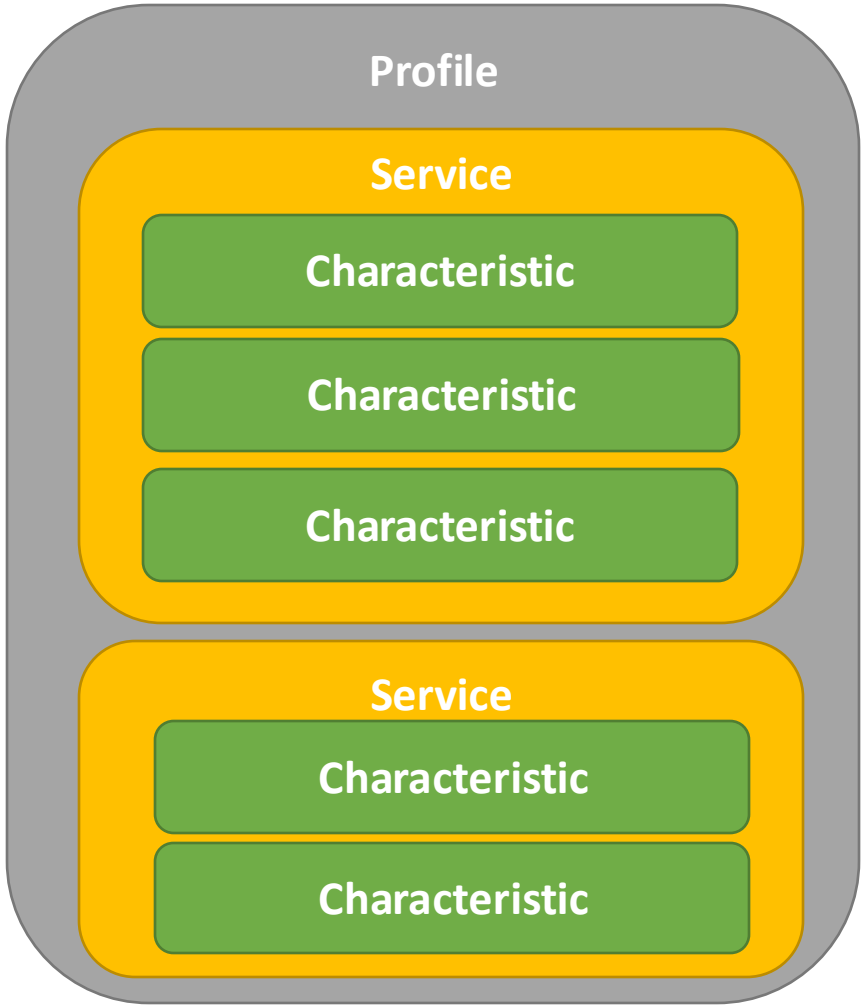
GATT services

- A **service** provides information through one or more characteristics
- Each service can be recognized by means of a unique numeric ID, the UUID
  - 16-bit (for officially adopted BLE Services)
  - 128-bit (for not officially adopted services)
  - E.g.: Heart Rate Service is officially adopted and has a 16-bit UUID equal to 0x180D
- You can design your own non-standard services



GATT characteristics

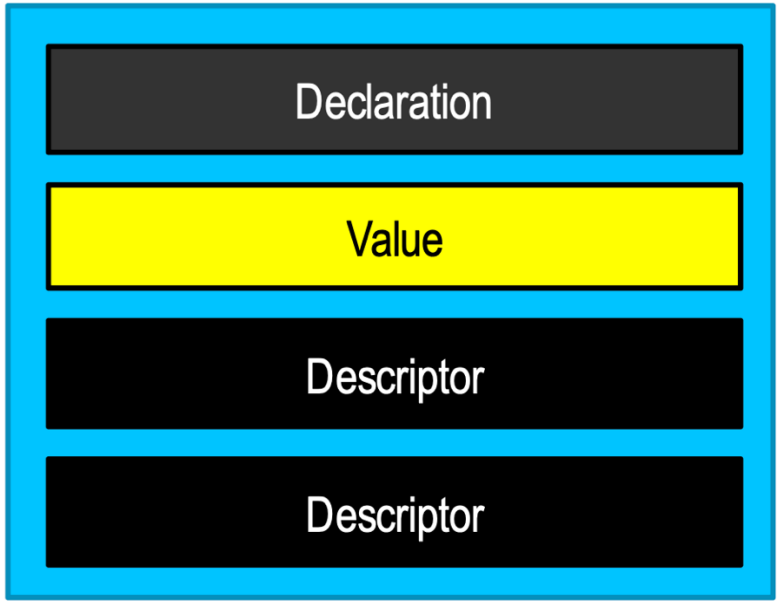
- Characteristics encapsulate data
- Heart Rate Service has three characteristics
  - *Heart Rate Measurement*
  - *Body Sensor Location* (optional)
  - *Heart Rate Control Point* (optional)
- Characteristics are identified by 16-bit or 128-bit UUID, if standard or not
- Standard characteristics defined by the Bluetooth SIG
- Heart Rate Measurement characteristic: UUID of 0x2A37, defines how data is encoded e.g. uint8_t



Characteristics

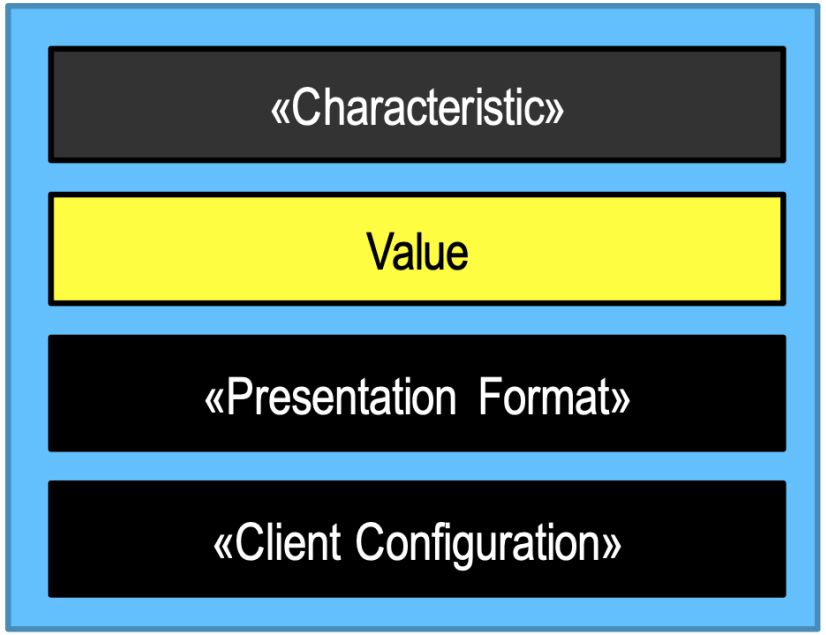
- What is a characteristics?

- It’s a value, with a known type, and a known format defined in a “Characteristic Specification”
  - Characteristic Declaration
  - Characteristic Value
  - Characteristic Descriptors



Characteristics

- Characteristics are grouped by «Characteristic»
  - Value attribute is always immediately after «Characteristic»
  - followed by descriptors
- Descriptors
  - additional information
  - any number
  - any order
  - can be vendor specific



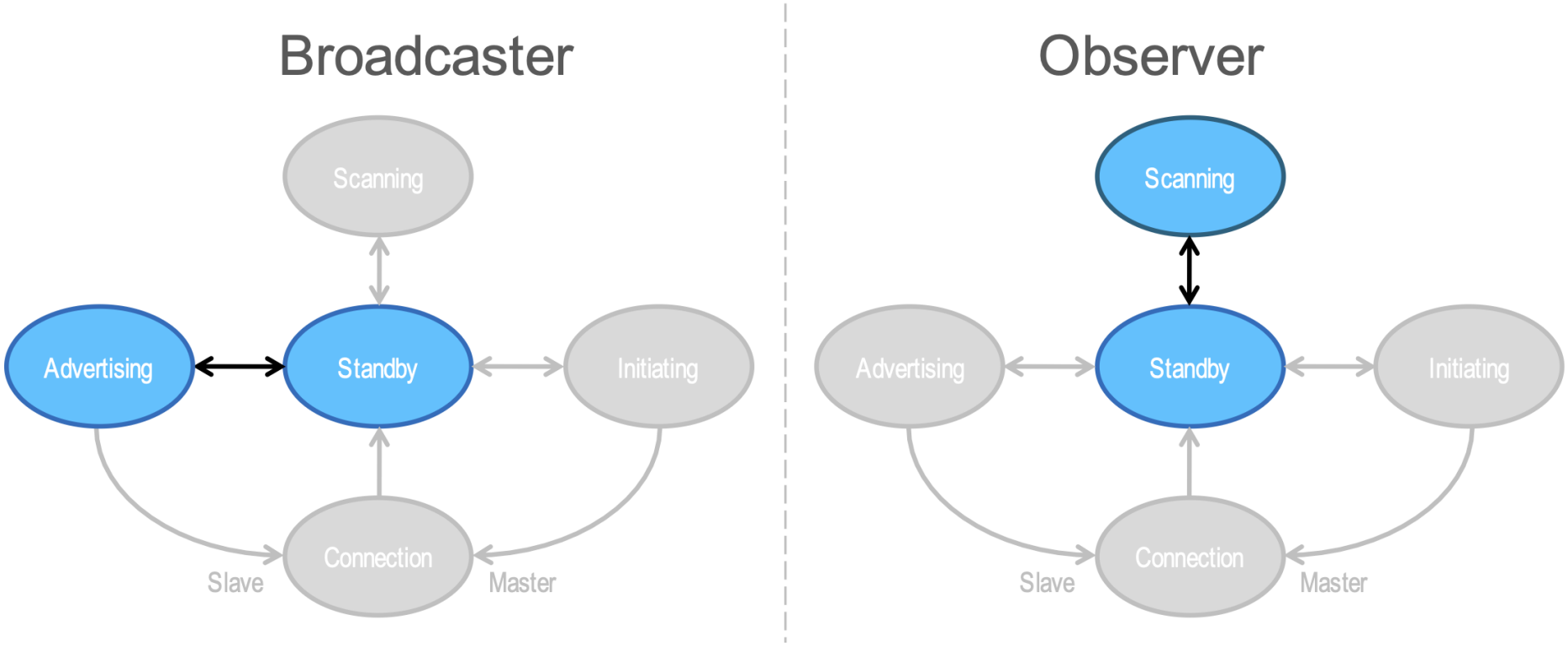
Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

Handle	Type	Value	Permissions
0x0001	«Primary Service»	«GAP»	R
0x0002	«Characteristic»	{r, 0x0003, «Device Name»}	R
0x0003	«Device Name»	“Temperature Sensor”	R
0x0004	«Characteristic»	{r, 0x0006, «Appearance»}	R
0x0006	«Appearance»	«Thermometer»	R
0x000F	«Primary Service»	«GATT»	R
0x0010	«Characteristic»	{r, 0x0012, «Attribute Opcodes Supported»}	R
0x0012	«Attribute Opcodes Supported»	0x00003FDF	R
0x0020	«Primary Service»	«Temperature»	R
0x0021	«Characteristic»	{r, 0x0022, «Temperature Celsius»}	R
0x0022	«Temperature Celsius»	0x0802	R*

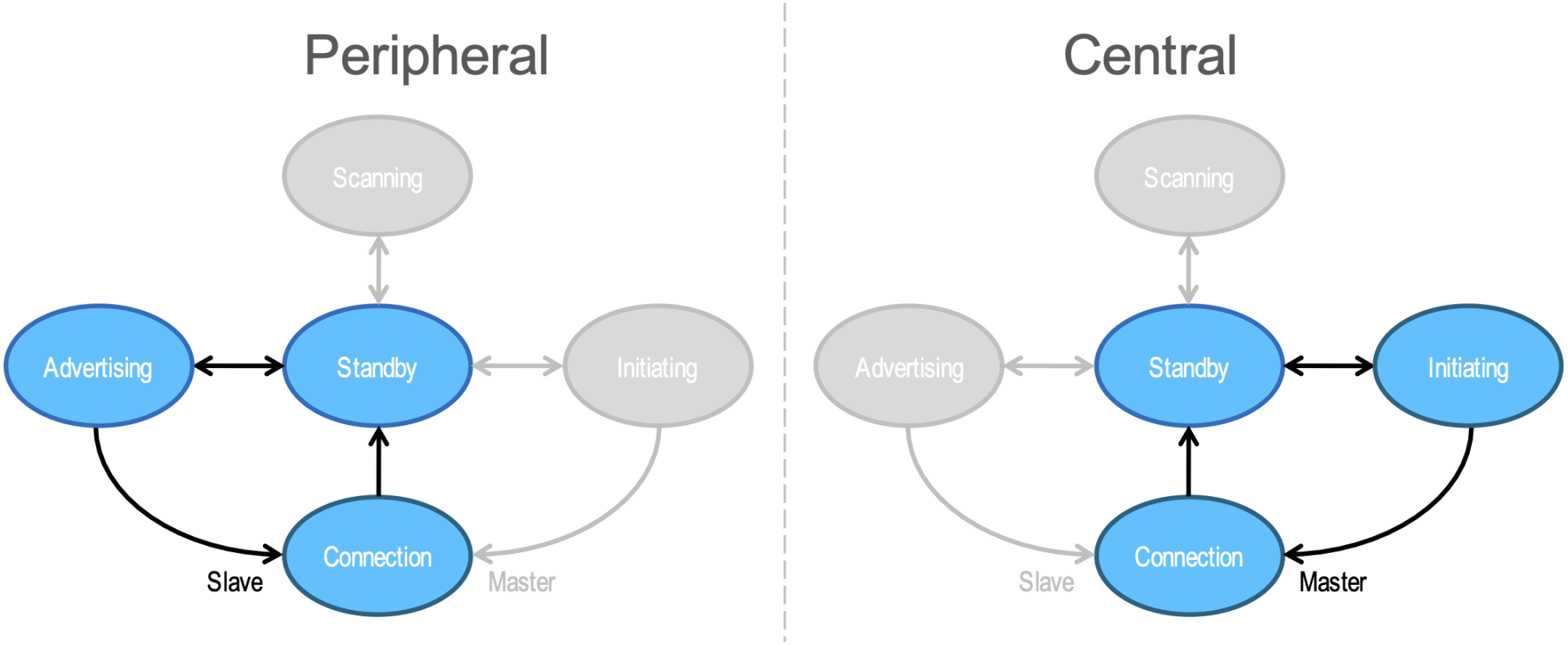
GAP

- LL FSM


Broadcaster
Scanning
Advertising
Standby
Initiating
Slave
Connection
Master
Observer
Scanning
Advertising
Standby
Initiating
Slave
Connection
Master

GAP

- LL FSM

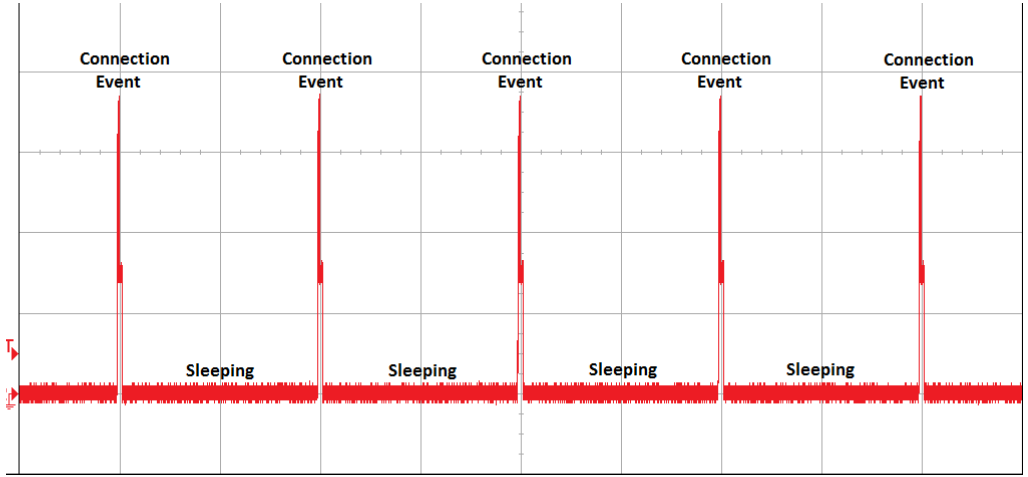

Peripheral
Scanning
Advertising
Standby
Initiating
Slave
Connection
Master
Central
Scanning
Advertising
Standby
Initiating
Slave
Connection
Master

Beacons

- Beacon: broadcasting small pieces of information
  - Information can be absent
- BLE Beacons include iBeacon and Eddystone
- Can be used for indoor localization
- Notify user when close to shops
- iBeacon:
  - a protocol developed by Apple
  - smartphones, tablets perform actions when close to an iBeacon
- Eddystone:
  - made by google, URLs are broadcasted
  - at the base of the Physical Web
  - automatic notifications even without app
  - discontinued at the end of 2018 (many spam notifications)
  - now replaced by google beacon

Power consumption

- Let’s compute the duration of a coin cell battery when powering a TI CC2541
- The analysis is limited to the BLE stack, without including possible other HW elements, such as accelerometers, etc
- Even in high throughput systems, a BLE device is transmitting only for a small percentage of the total time that the device is connected



Power consumption

- In addition to transmitting, a BLE device will most likely go through several other states, such as receiving, sleeping, waking-up from sleep which must be taken into account

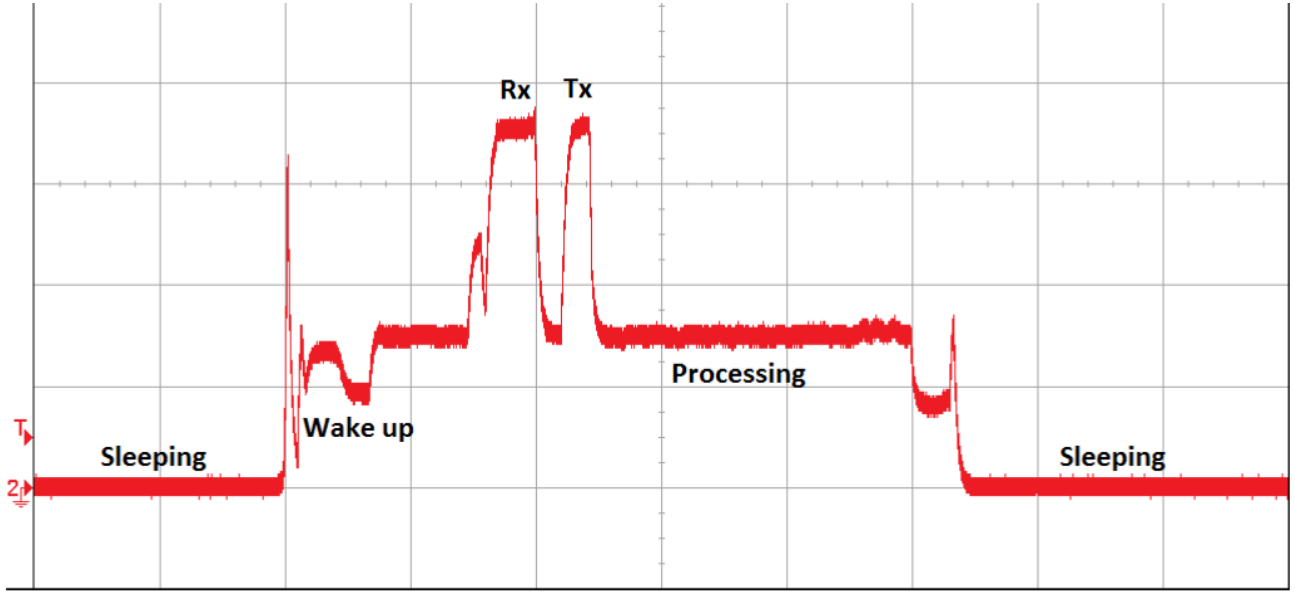
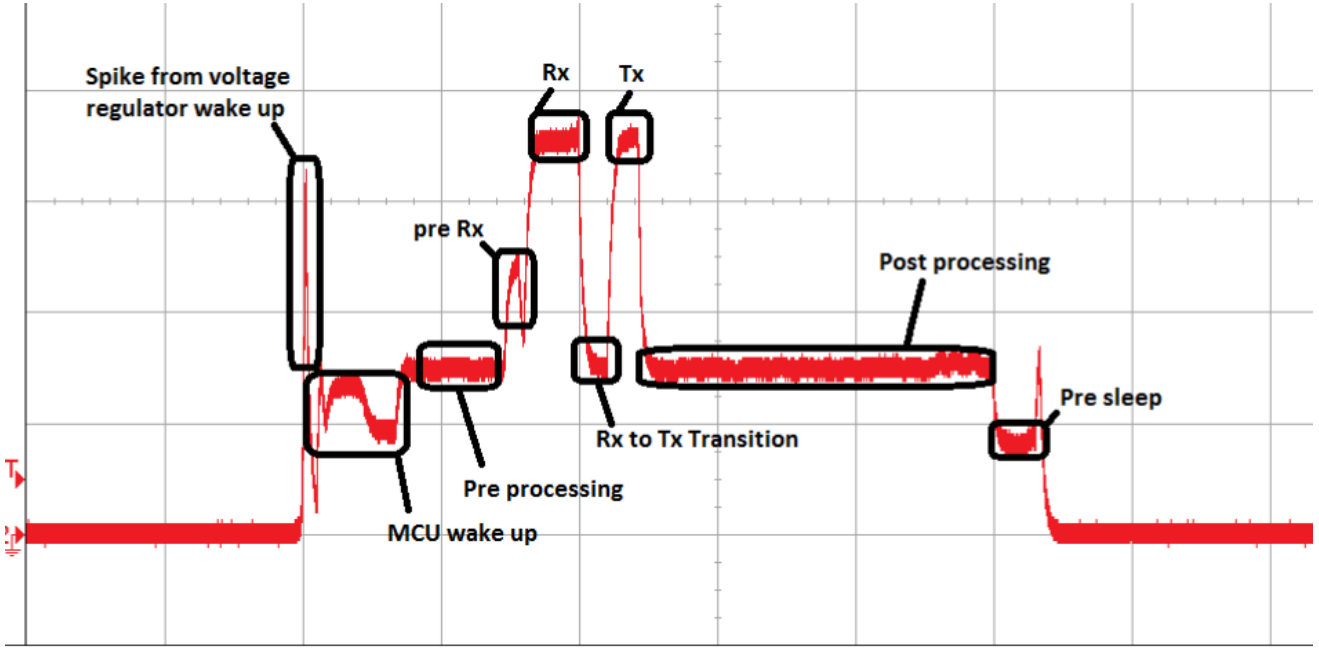


Figure 2- Current Consumption versus Time during a single Connection Event

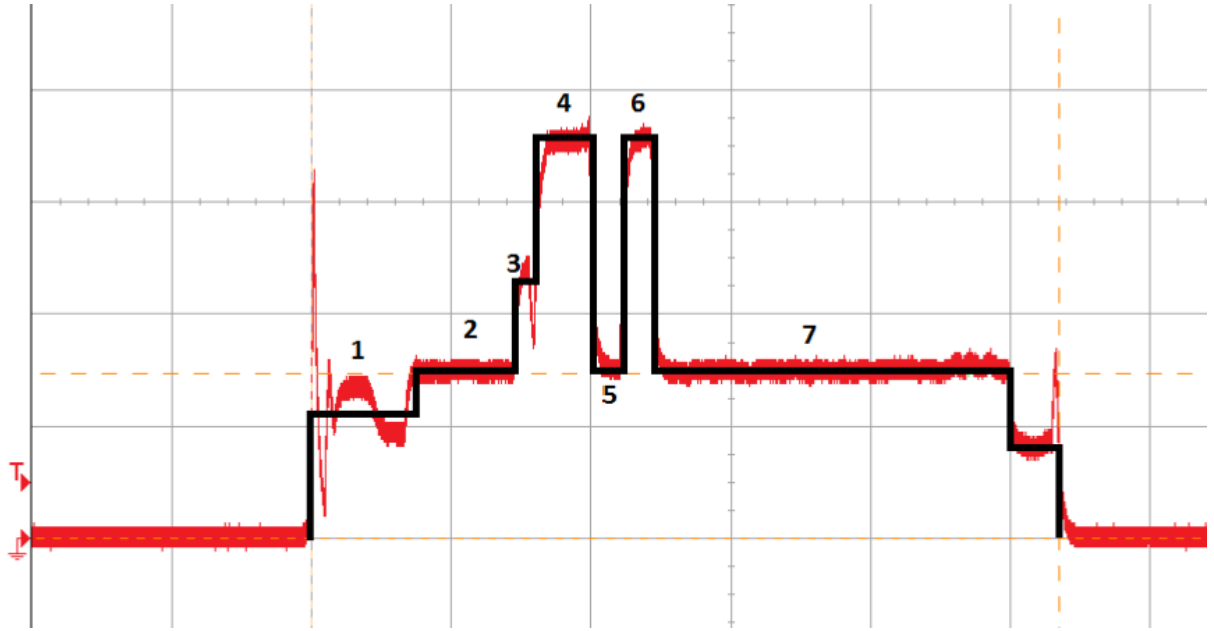
Power consumption

- More in detail
- **MCU wake-up:** upon waking up, the current level drops slightly
- **Pre-processing:** the BLE protocol stack prepares the radio for sending and receiving data
- **Pre-Rx:** the CC2541 radio turns on in preparation of Rx and Tx
- **Rx:** the radio receiver listens for a packet from the master
- **Rx-to-Tx** transition: the receiver stops, and the radio prepares to transmit a packet to the master
- **Tx**: the radio transmits a packet to the master
- **Post-processing:** the BLE protocol stack processes the received packet and sets up the sleep timer in preparation for the next connection event.
- **Pre-Sleep:** the BLE protocol stack prepares to go into sleep mode



Power consumption

	Time [µs]	Current [mA]
State 1 (wake-up)	400	6.0
State 2 (pre-processing)	340	7.4
State 3 (pre-Rx)	80	11.0
State 4 (Rx)	190	17.5
State 5 (Rx-to-Tx)	105	7.4
State 6 (Tx)	115	17.5
State 7 (post-processing)	1280	7.4
State 8 (pre-Sleep)	160	4.1

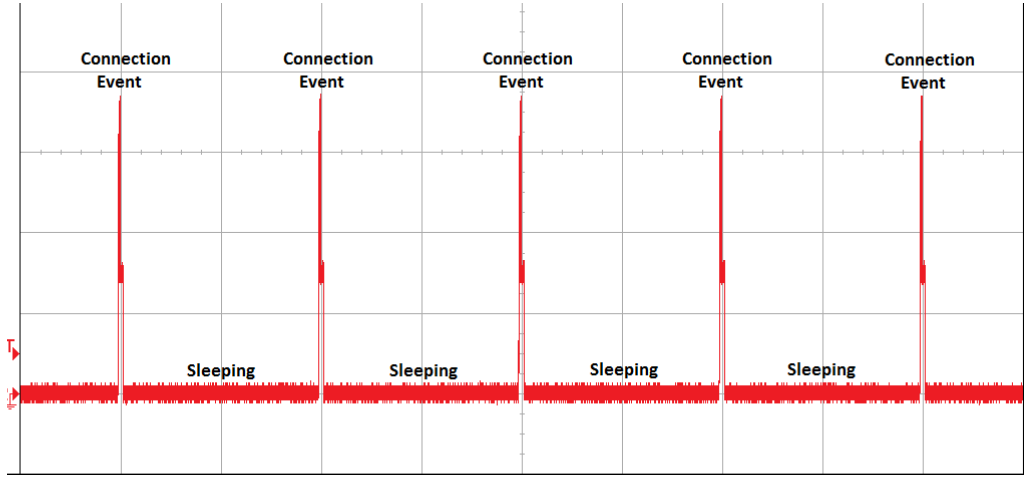


Average current during connection event = [(State 1 durat.)*(State 1 current) + (State 2 durat.)*(State 2 current) + …] / (Total awake time)

8.24 mA

Power consumption

- Between connection events CC2541 goes into low power state PM2 (Power Mode 2)
  - internal digital voltage regulator is turned off, along with 32 MHz crystal oscillator. The 32 kHz sleep timer remains active while the RAM and registers are retained
- Current absorbed in sleeping state can be measured: 1 uA



Power consumption

- Final step: calculate average current for the entire connection interval, which takes into account the time during which the device is sleeping

Average current while connected = [(Connection Interval – Total awake time) *(Average sleep current) + (Total awake time)*(Average current during connection event)] / (Connection Interval)

- [(1000 ms – 2.675 ms)*(0.001 mA) + (2.675 ms)*(8.24 mA)] / (1000 ms) = 0.0230 mA

- The average current consumption while the device is in a connected state is approximately 0.023 mA (23 uA)

Power consumption

- A coin cell battery CR2032 ha a capacity of 230mAh
- Expected battery life = 230 mAh / (0.023mA) = 10000 h
- Approximately 400 days

- Always in connection state, 1 s connection interval, no other sources of consumption

References

- Getting Started with Bluetooth Low Energy, O'Reilly
- Measuring Bluetooth Low Energy Power Consumption, Sandeep Kamath & Joakim Lind, Texas Instruments
- Introduction to Bluetooth Low Energy, R. Heydon, Qualcomm