

BLE

→ progettato per applicazioni IoT con consumo energetico basso



utilizza AFH (Adaptive Frequency Hopping)

e 24-bit CRC Redundancy Checks

- frame brevi
- lunghi periodi di inattività
- ↓
anni e mesi di funzionamento
- connessioni veloci

Supportato da più, sicuro, robusto e affidabile

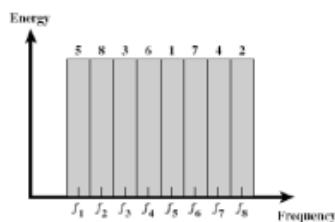
Frequence Hopping \Rightarrow CDMA \Rightarrow durante intervallo T_c la frequenza della portante è costante

il ricevitore
conosce sequenza
di hop e può
ricostruire il segnale

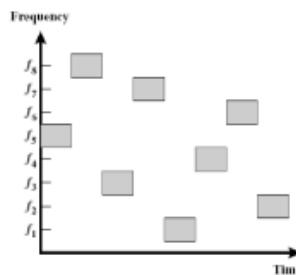
utilizzato da
in modo pseudo
casuale utilizzando
codice dell'utente

salta ad un altro
canale (frequenza)

+ sicurezza
+ resilienza da
interferenze

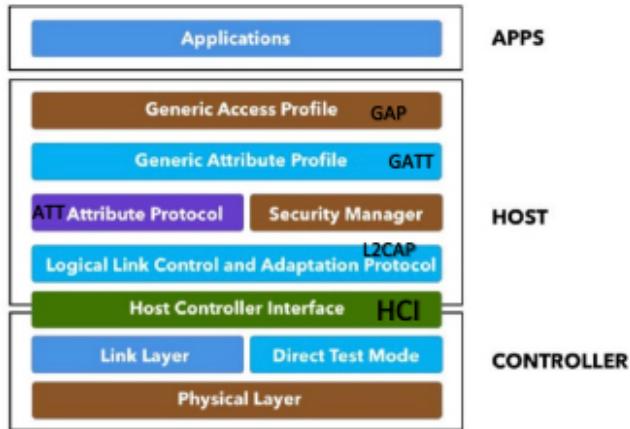


(a) Channel assignment



(b) Channel use

ARCHITETTURA BLE



CONTROLLER \Rightarrow parte dell' architettura che gestisce la parte hardware connessa alle porte I/O

- \rightarrow **livello PHY**:
utilizza AFH per selezionare canale.
BLE \Rightarrow 2.4 GHz \Rightarrow 40 canali
ognuno separato da intervallo di 2 MHz
 \hookrightarrow 3 canali
- \downarrow
runca non nella CPU
- \downarrow
cellulare \Rightarrow chip Bluetooth
- \downarrow
dove host fa
da interfaccia
per interagire
con operazioni
Bluetooth
- \rightarrow **livello LLC**:
si interfaccia con PHY tramite HCI.
Le operazioni sono:
 - encryption
 - stabilire connessione
 - ritrasmissioni
- \rightarrow **livello HCI**: fa da interfaccia per la **parte di host**

HOST → Logic Link Control and Adaptation Protocol (L2CAP)

→ prende i pacchetti da sopra e li spieghetta della dimensione max dei chunks che stanno nel payload supportato per la trasmissione.

↓ quello che riceve
riconverte il messaggio

ATT ⇒ P2P definisce il trasferimento dati tra client e server

↓
dati → formati da attributi
→ questo provvede a leggere e scrivere, indicare e notificare valori di attributi sulla connessione

→ un'app sul telefono

server:

il dispositivo salva
(generando) dati
(come 2 o più attributi)

→ smartwatch

client:

colleziona informazioni
per uno o più server

due ruoli

↓ stato dispositivo

Il client può accedere agli attributi dei server mandando richieste.

↓ manda comandi per scrivere attributi

il server può inviare

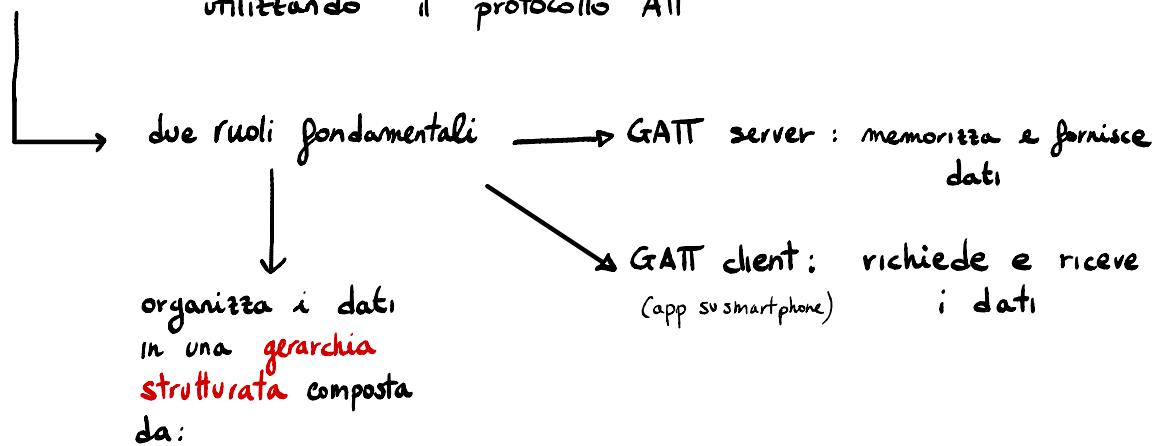
due tipi di messaggi → indicazioni
che contengono contenuti
(il cliente manda conferma)

Read, Write, Notification

Struttura di un attributo → nel server

2 Octets	2 or 16 Octets	Variable Length	Implementation Specific Length
Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
↓	UUID	↓ 16 bit identificativo assegnato dal server per riferirsi a quegli attributi ↓ funziona come indice!	↑ per ogni UUID specifico ↓ è un identificativo universalmente univoco (alcuni custom) ↓ exe: 0x2A37 rappresenta la misura del battito del cuore ↓ ci pensa GATT ↓ valore dell'attributo

GATT → organizza e gestisce comunicazione tra dispositivi BLE utilizzando il protocollo ATT



1. Profilo ⇒ insieme di servizi che supporta il dispositivo (cosa può fare)
2. Servizi ⇒ contiene caratteristiche, identif. da UUID
3. Caratteristiche

GATT server \Rightarrow salva i dati, manda risposte a richieste e se configurato manda indicazioni e notifiche quando occorre evento nel Gatt server.



Servizi \Rightarrow contengono una collezione di caratteristiche

Caratteristiche \Rightarrow contengono singole informazioni

ognuno di esso ha:

- UUID
- valore
- descrittori

permessi

formato

notify attivo o no

Security Manager Protocol (SMP) \Rightarrow protocollo P2P per creare, gestire e assegnare chiavi per criptare

GAP \Rightarrow Generic Access Profile \rightarrow specifica **se e come** due dispositivi possono interagire tra di loro.

↓
stati:

connectable: se si può connettere

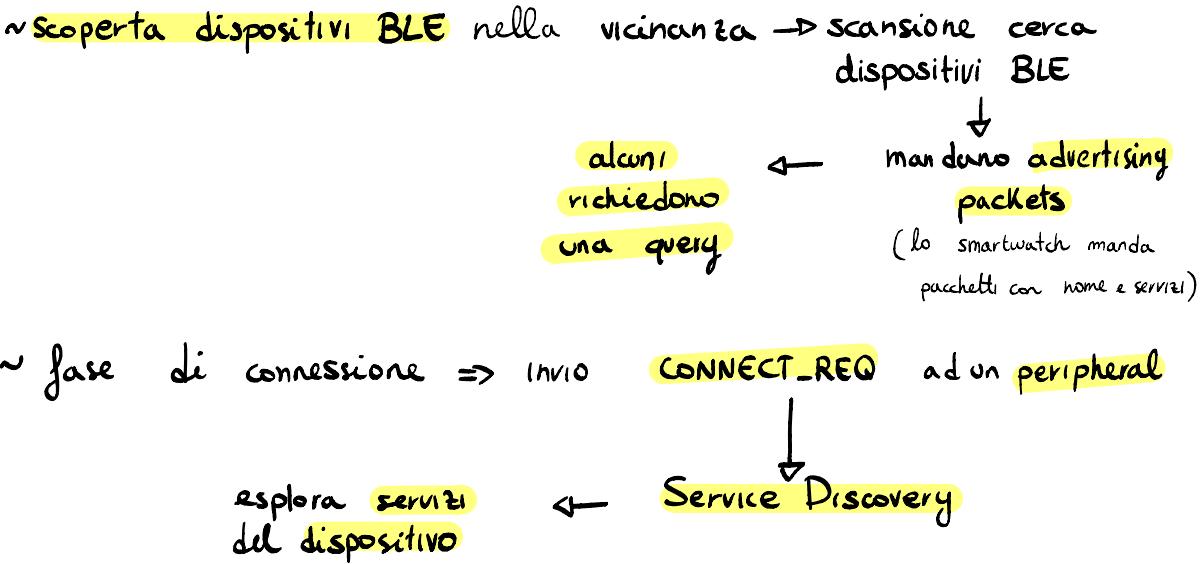
discoverable: può essere scoperto
None, limited, general

bondable: se è connettabile il dispositivo, si accoppierà con il dispositivo per un tempo lungo.

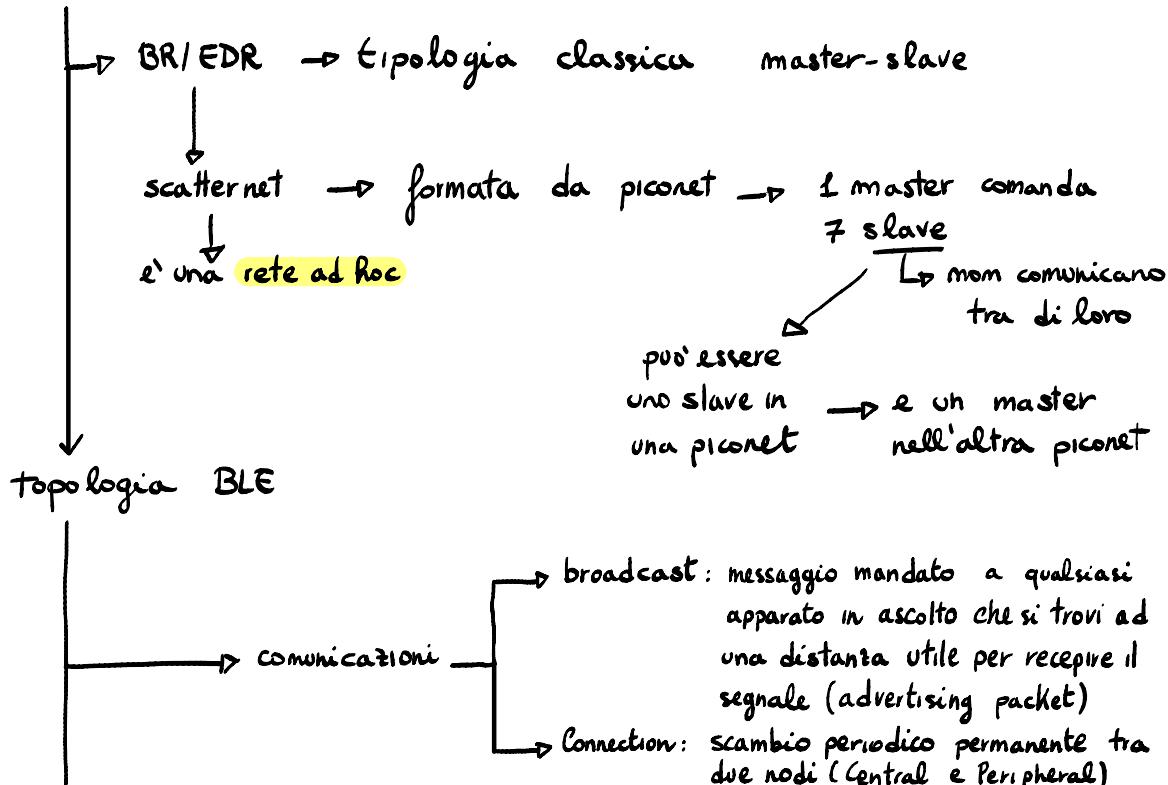
↓
come un dispositivo è

visibile e **connettibile**
con gli altri device

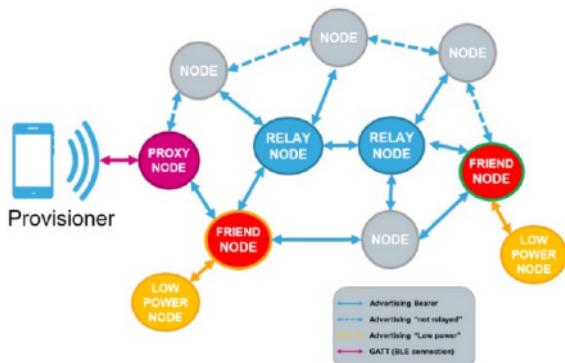
Procedura:



TOPOLOGIA DELLE RETI BLUETOOTH



BLE - MESH → Comunicazione tramite flooding → messaggi trasmessi in broadcast ai dispositivi vicini
 ↓
 TTL per evitare congestioni



Self-Healing
 messaggi trovano percorsi alternativi
 i nodi vicini replicano il messaggio

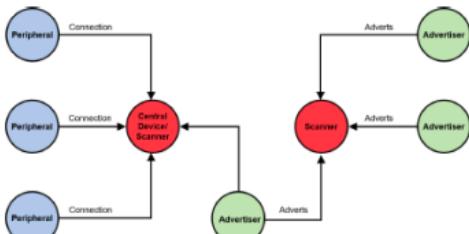
~ relay node ⇒ inoltra messaggi agli altri nodi aumentando copertura di rete.

~ Proxy node ⇒ permette comunicazione tra dispositivi non mesh e la rete mesh.

~ Friend node ⇒ Memorizza temporaneamente messaggi destinati ai nodi Low Power

~ Low Power Node ⇒ minimizza consumo energetico, rimanendo inattivo il più possibile.

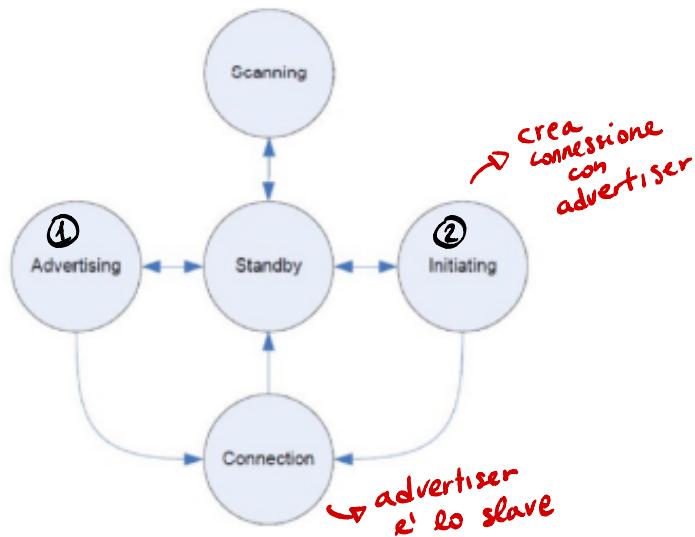
STAR BUS → peripheral non è sempre in ascolto per connettersi con il centrale → ma invia pacchetti advertising



↓
 master decide a quale connettersi
 Peripheral si disconnette
 inizia a pubblicare advertising per ridurre consumo

- **Advertiser** ⇒ invia pacchetti advertising con info utili (ID univoco, temperatura o posizione)
- **Scanner** ⇒ ascolta pacchetti advertising e inoltrarli ad un server.
- **Central Device (Master)** ⇒ ascolta pacchetti advertising e crea connessioni (Smartphone) con i peripheral.
⇒ gestisce più peripheral contemporaneamente
- **Peripheral Devise (smartwatch)** ⇒ invia periodicamente advertising packet per segnalare possibili connessioni
⇒ quando connesso, niente adv e segue comandi master
- **Hybrid Device** ⇒ sia Advertiser che Scanner

BLE STATES



Lo swap da attivo a stand by permette di diminuire l'energia consumata

prende luogo tra gli stati 1 e 2

LINK LAYER \Rightarrow instaurare connessioni
(parte controller)

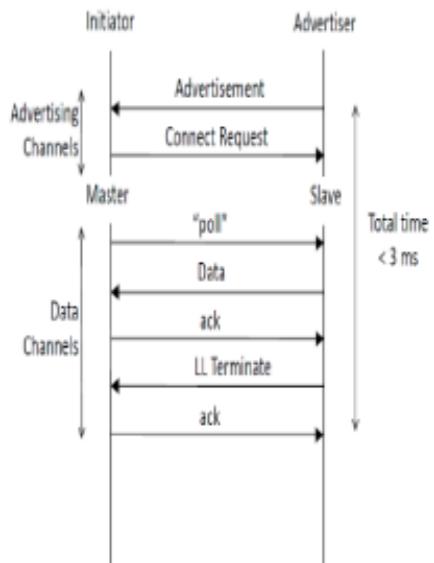
due stati importanti:

Advertising State

Scanning State

i pacchetti sono inviati \rightarrow su 3 canali
a un intervallo fissato mentre nel caso tradizionale
32 canali

da 20 ms fino a 10.24 s



Attivo
fa richiesta
e ottiene
lista di servizi

device discovery
round robin
scannerizza
sequentialmente
ogni canale

Passivo
un dispositivo
può solo ricevere
dati da advertiser
ma non fa nessuna
richiesta

\rightarrow connessione \rightarrow stesso trap
stabilità pattern

Advertising packet

Preamble (8 bits)	Access Address (32 bits)	PDU (2 to 39 bytes)	CRC (24 bits)
----------------------	-----------------------------	------------------------	------------------

per sincronizzazione

negli advertising
e' sempre lo stesso
per garantire che
tutti i BLE possano
riconoscere questi
tipi di pacchetti

nei pacchetti dati
viene scelto uno
univoco e casuale
per quel collegamento

Header 2 Byte	Advertiser Address Adva 6 Byte	Advertising Data AdvData max 31 Byte
------------------	--------------------------------------	--

publico → randomico
lunghezza PDU

payload

si ha di due
tipi:

publico
non cambia
mai associato
alla vita del
dispositivo

Randomico:
generato dinamicamente
↓
generato ogni volta
che il dispositivo
riparte

Trovare posizione e direzione



RTLS
e i o y
a m c j
l e t e
+ +
n n
g s



l'applicazione
puo' determinare
la posizione del
suo host trovando
la direzione dal
quale arriva il segnale

approssima
distanza dal beacon
e la posizione



almeno due riferimenti ai beacon Bluetooth devono essere saputi e la loro distanza deve essere saputa

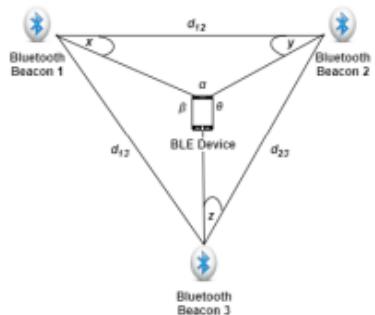
accuratamente = 3 beacon

usa il received signal strength indicator (RSSI)

per calcolare distanza tra beacon e il device

non si sa la direzione quindi si assume che il dispositivo sia in qualsiasi punto della circonferenza

intersezione delle 3 circonference



necessario 3 beacons

misurazione angoli \Rightarrow AoA

\Downarrow AoD angle of arrival
angle of departure

Con angoli misurati e distanze tra i beacon si calcola la posizione esatta del dispositivo

→ possono non avere punto di intersezione