Sommario

[**Introduzione** 2](#_Toc48745965)

[**Bluetooth (IEEE 802.15.1)** 3](#_Toc48745966)

[**Bluetooth Low Energy – BLE** 3](#_Toc48745967)

[**Bluetooth Low Energy – Struttura generale** 4](#_Toc48745968)

[**Bluetooth Low Energy - Beaconing** 5](#_Toc48745969)

[**Apple/Google Exposure Notification API** 7](#_Toc48745970)

[BLE Beaconing AG ENAPI – Broadcast 8](#_Toc48745971)

[BLE Beaconing AG ENAPI – Scansione 9](#_Toc48745972)

[**Bluetooth Beaconing BLE – Aspetti implementativi** 9](#_Toc48745973)

[Bluetooth Beaconing BLE – Client (Android Beacon Library) 10](#_Toc48745974)

[Bluetooth Beaconing BLE – Beacon (TODO, Android Beacon Library) 11](#_Toc48745975)

[iBeacon 11](#_Toc48745976)

[Eddystone 12](#_Toc48745977)

[AltBeacon 12](#_Toc48745978)

[**BTPing – Toy case BLE Beaconing/Scanning** 13](#_Toc48745979)

Laboratorio di Mobile Systems M Sperimentazione BLE Beaconing e possibili applicazioni

# **Introduzione**

Nonostante la presenza di progetti pre-esistenti che ne coinvolgono l’implementazione diretta, il BLE Beaconing (Bluetooth Low Energy Beaconing) ha visto la propria implementazione su larga scala con le Google/Apple Exposure Notification API, trasformando i dispositivi di telefonia mobile dotati di Bluetooth in beacons. Il concetto delle G/A ENAPI è quello di far diventare il dispositivo sia ricettore Bluetooth che Beacon, permettendo di interagire con gli altri dispositivi e adempiere alla funzione prevista: il controllo dell’esposizione dell’utente a contatti che possono essere “possibili infetti”, attraverso un sistema di scambio di informazioni, il cui concetto chiave è la “prossimità fisica” tra i dispositivi.  
Come detto, non è l’unica applicazione di questa tecnologia: il Bluetooth Beaconing è utilizzato per fornire navigazione e/o tracciamento degli utenti in luoghi di estensione limitata, similmente alla tecnologia Radar. Ancora, i Beacons possono diventare un sistema di notifica, nel momento in cui un utente si trova in prossimità del beacon, ad esempio per scopi turistici o pubblicitari.  
Quanto illustrato permette di comprendere che il Bluetooth beaconing può avere molte implementazioni e, nel caso delle G/A ENAPI, dimostra essere efficace. L’obiettivo di questo progetto di Laboratorio, dunque, si prefigura l’esplorazione e l’implementazione di semplici toy examples del BLE Beaconing e dimostrarne l’efficacia entro determinati parametri di performance.

# **Bluetooth (IEEE 802.15.1)**

Bluetooth è una tecnologia di comunicazione wireless per lo scambio di dati tra dispositivi fissi e mobili. La comunicazione è a corto raggio, tra i 10 e i 100 metri normalmente, fino a 400 metri nella versione Bluetooth 5.0, con onde radio di frequenza 2.4 GHz (similmente a WiFi) e un consumo energetico che va da 100 mW a 1 mW (per i dispositivi di classe più bassa). A differenza di quest’ultimo, la connessione è Punto-A-Punto e necessita di effettuare Discovery dei dispositivi e utilizza il Frequency-Hopping per la distribuzione della comunicazione sulle frequenze disponibili.  
Il protocollo Bluetooth è un protocollo di scambio a pacchetti ed una rete bluetooth è strutturata come una Piconet Master/Slave: un dispositivo assolverà al ruolo di Master, iniziando il protocollo, creando la rete ed eseguendo una scansione per trovare i nodi vicini. I nodi che desiderano unirsi alla rete inviano al Master il proprio ID, ricevono il clock del master per sincronizzarsi e gli vengono assegnati una frequenza del canale di comunicazione (bidirezionale) e i tagli temporali in cui il dispositivo può comunicare con il Master. Una Piconet può avere al massimo 7 dispositivi connessi (di cui un master), ma più Piconet possono unirsi in Scatternets quando condividono un nodo con qualsiasi ruolo. In quel caso, il nodo diventerà un gateway tra le due reti.  
Bluetooth distingue due tipologie di connessioni: ACL, Asynchronous Connection Less, utilizzato per la trasmissione di pacchetti dati e SCO, Synchronous Connection Oriented, per stream di dati, per esempio stream di dati audio come la voce.  
G/A ENAPI utilizzano una connessione di tipo ACL, mentre, ad esempio, gli auricolari Bluetooth instaurano una connessione di tipo SCO.  
Nelle ultime versioni, presenta anche tecnologie più avanzate, come il supporto a 802.11n, dualità nei ruoli dei nodi, maggiore velocità di trasmissione, l’implementazione dell’Angle of Arrival e Angle of Departure per la localizzazione e il tracciamento.  
Con l’avvento delle tecnologie smart e dell’IoT, che richiedono consumi considerevolmente ridotti, possibilmente a parità di performance, Bluetooth (adesso riferito come Bluetooth “classico” o BR/EDR) è stato espanso con Bluetooth Low Energy.

## **Bluetooth Low Energy – BLE**

BLE è una versione distinta rispetto al Bluetooth “classico”, nonostante le differenze tra i due siano minori.

BLE nasce come versione a consumi e costi notevolmente ridotti, a parità di performance, di Bluetooth. Infatti, nonostante ci sia uno scarto tra BLE e Bluetooth “classico”, la differenza è molto piccola, rispetto al guadagno in termini di risparmio energetico: se il raggio d’azione è inferiore rispetto a BT, così come è relativamente più lento nel trasferimento dati (3 contro 2 Mbit/s), dall’altra parte consuma da 100 a 500mW con picchi di corrente fino a 15mA (rispetto a 30mA di BT).  
Per quanto riguarda la connessione, mentre la frequenza di trasmissione rimane comunque a 2.4 GHz, si dimezza la distribuzione dei canali, passando da 1MHz a 2MHz, senza però variare dalle tecniche di Frequency Hopping e Time slicing. Una maggiore differenza è rappresentata dal fatto che i dispositivi BLE si possono trovare effettuando Broadcast di pacchetti di annuncio: avendo disponibili fino a 3 canali di trasmissione, per evitare interferenze, i dispositivi trasmettono questi pacchetti su almeno uno dei tre canali, ritrasmettendo ad intervalli di tempo prefissati, o “finestre di annuncio”. Il dispositivo che sta effettuando la scansione si mette in ascolto per un intervallo di tempo definito come “finestra di scansione”.  
L’aspetto più importante è l’utilizzo, mutuato da BT, dei profili per impostare le specifiche di funzionamento del dispositivo (in base al contesto di azione). Normalmente, la maggioranza dei profili a consumo energetico ridotto sono basati su GATT (Profili d’attributo generici). L’implementazione specifica del dispositivo permetterà di inviare dati anche specifici al dispositivo stesso (temperatura, frequenza cardiaca, et cetera).  
L’unico limite ancora presente di questa tecnologia è lo scambio di dati audio in stream, possibile solo per BT, nonostante vi siano effettivamente progetti per un’implementazione di tale funzionalità da parte di Bluetooth SIG.  
  
Le applicazioni notevoli di BLE sono principalmente collegate a sensori personali che permettono di monitorare almeno un evento: come citato sopra, è utilizzato in sensori ambientali (profilo ESP), medici (pressione sanguigna, temperatura corporea, glicemia per i diabetici). Tuttavia, di notevole interesse sono i sensori di prossimità e il tracciamento:

* Prossimità: comunemente noti come “guinzagli elettronici”, consentono di collegare oggetti ad un dispositivo che ne rileva la posizione (profilo PXP), nel caso in cui siano fuori posto o non siano “a vista” (profilo FXP). Per effettuare il calcolo della “distanza” dall’oggetto, usa RSSI.
* Tracciamento: è possibile, attraverso l’utilizzo di BL, costruire sistemi di tracciamento di carattere informativo o posizionale. Possono essere usati per la localizzazione indoor o per tenere traccia di interazioni, soprattutto di carattere commerciale e turistico, per una personalizzazione dell’esperienza utente. Chiaramente il tracciamento, per quanto concerne esclusivamente il Bluetooth e non le applicazioni che lo utilizzano, è limitato al solo contesto di utilizzo, dato che BLE non è adatto a scambiare quantitativi ingenti di dati.

## **Bluetooth Low Energy – Struttura generale**

I componenti principali e generali del BLE sono essenzialmente 5:

* GATT (General Attribute Profile): è un profilo generico per lo scambio di piccole quantità di dati su un collegamento BLE (a basso consumo). Tutti i profili a consumo energetico ridotto sono basati su GATT. I dati scambiati sono definiti come “attributi”
* Attribute Protocol (ATT): gli attributi sono i dati d’interesse scambiati. GATT è costruito su ATT ed ATT è ottimizzato per i dispositivi BLE, in quanto usa quanti meno bytes possibili. Un attributo è composto dai dati e da un UUID, un ID di stringa a 128 bit per l’identificazione univoca di informazioni. Gli attributi sono formattati come “caratteristiche” e “servizi”.
* Caratteristiche: le caratteristiche contengono un valore singolo e possono contenere uno o più descrittori che arricchiscono il valore della caratteristica. Possono essere intesi come gli equivalenti delle classi.
* Descrittori: sono attributi definiti che descrivono il valore di una caratteristica. Possono essere descrizioni in linguaggio umano, range di valori che il valore della caratteristica può assumere, o anche un’unità di misura.
* Servizio: è una collezione di caratteristiche. In questo senso, è possibile avere un servizio che identifica la funzione del dispositivo e possiede caratteristiche contestuali. Un esempio è il monitor del battito cardiaco (servizio) che conterrà il battito cardiaco (caratteristica). I servizi, similmente ai profili, vanno definiti ed implementati.

Riguardo ai profili e ai servizi, Bluetooth SIG fornisce implementazioni basate su GATT già esistenti.

BLE non ha la stessa struttura architetturale del Bluetooth “classico”, nonostante ci sia una parziale “gerarchia”: invece del paradigma M/S, i dispositivi BLE possono assumere i ruoli di dispositivo “centrale” e “periferico”. Un dispositivo “centrale” effettua scansioni per cercare messaggi di annuncio, mentre i dispositivi “periferici” sono quelli che inviano messaggi di annuncio. Oltre a questa distinzione, si distingue in BLE un dispositivo come GATT Server o GATT Client: il GATT Client è il dispositivo che riceve dati (generalmente tramite un’applicazione), mentre il GATT Server è il dispositivo che li invia. Riprendendo il caso del monitor del battito cardiaco, il monitor è il dispositivo periferico e assume il ruolo di GATT Server, mentre il dispositivo che riceverà informazioni dal monitor è un dispositivo centrale e GATT Client.

## **Bluetooth Low Energy - Beaconing**

BLE permette a qualsiasi dispositivo che monti un circuito Bluetooth (4.0 o superiori) di diventare un “beacon”. Come suggerisce il nome, il dispositivo diventa un trasmettitore per i dispositivi Bluetooth nelle vicinanze, fornendo informazioni di un determinato tipo o generando eventi, senza necessità di “accoppiamento” dei dispositivi.   
Tuttavia, va specificato che, inizialmente, l’architettura beaconing iniziale prevedeva il beacon come un dispositivo “dedicato”: fino ad ora, l’architettura prevedeva uno o più dispositivi single-purpose adibiti a trasmettitori e un ricevitore, costituito dal dispositivo mobile che avesse in dotazione un circuito B4.0+.   
La rivoluzione dell’architettura, con le AG ENAPI, consiste nel fatto che il dispositivo mobile può diventare a sua volta beacon, assumendo entrambi i ruoli di ricevitore o trasmettitore in maniera alternata. Di base, un beacon effettua comunicazioni broadcast mono-direzionali (non può ricevere, può solo trasmettere). Un dispositivo “beacon”, locato in un Punto di Interesse, utilizza il sensore di prossimità per inviare l’UUID e i dati relativi al contesto di azione del dispositivo che, tramite un’applicazione o un sistema operativo compatibile, vengono riconosciuti e permettono di eseguire azioni contestuali al beacon rilevato. Successivamente, come già accennato e come spiegato a breve, il dispositivo mobile diventa capace di assumere i due ruoli: la “dualità” consisterebbe nella possibilità di effettuare broadcast e, con lo stesso dispositivo, poter porsi in ascolto di possibili broadcast provenienti da altri dispositivi.  
Quindi si possono individuare problematiche intrinseche che hanno portato ad evolvere il modello di base, tutte legate alla scalabilità di determinati aspetti relativi al ciclo di vita del sistema in uso. Per capire meglio queste problematiche tra il modello del “beacon come dispositivo single purpose” e il modello “beacon e listener nello stesso dispositivo”, osserviamo alcuni esempi notevoli d’utilizzo di questi dispositivi:

* Mobile commerce: un negozio può disporre un beacon per essere riconosciuto ed effettuare promozioni e marketing, inviando notifiche al dispositivo compatibile. Solitamente può essere usato anche per effettuare pagamenti elettronici.
* Turismo smart: alcuni POI possono essere dotati di beacon e trasmettono informazioni sul POI ai dispositivi compatibili, soprattutto di carattere informativo.
* Localizzazione in ambienti interni: disponendo dispositivi beacons all’interno di una stanza o di un edificio è possibile fornire un sistema di posizionamento. In base alla densità dei beacons, variano anche le tecniche di localizzazione, dalla triangolazione RSSI ai percorsi predittivi.
* Tracciamento: un dispositivo beacon, solitamente indossabile, permette di rilevare la propria presenza nelle vicinanze. Esempi implementativi sono già stati presentati in passato, come i braccialetti di tracciamento per bambini della Nivea nel 2014. Attualmente sono attivi diversi progetti di braccialetti di tracciamento, come i-Feel-You dell’IIT.

Mentre per il Mobile Commerce e alcune istanze del problema di tracciamento il problema può non essere posto, perché i dispositivi beacon in uso possono essere pochi e direttamente sostituibili senza costi onerosi, il problema si pone in scenari come il Turismo Smart, la localizzazione in ambienti interni e l’attuale tracciamento applicato alle procedure di “contact tracing”: in questi casi, il numero di dispositivi di beaconing potrebbe aumentare, in quantità importanti. Basti pensare agli esempi del Gatewick Airport (UK) e del Tom Jobin International Airport (BZ), che utilizzano più di 2000 dispositivi per effettuare la localizzazione e navigazione attraverso le strutture: considerando che i beacons sono stati pensati come dispositivi modesti, a limitate risorse e a basso consumo energetico, quindi sono stati progettati per avere solo funzionalità Bluetooth e solo di trasmissione, la manutenzione e gli interventi di riparazione e/o sostituzione di questi dispositivi non è scalabile con sistemi centralizzati e telematici, ma deve essere effettuata manualmente. Questo è un limite importante, per quanto riguarda dispiegamenti di sistemi su larga scala. Successivamente, l’episodio pandemico di Covid-Sars-19 ha portato alla necessità di un sistema di tracciamento intelligente che prescindesse dal dover possedere due dispositivi diversi, in quanto un sistema a sostegno del contact tracing doveva avere caratteristiche particolari:

* Semplicità di utilizzo
* Basso costo per l’utente, in termini economici e di risorse, specie energetiche
* Accessibilità massima

Tutte e tre caratteristiche necessarie per favorirne quanto più facilmente possibile la diffusione, condizione necessaria per un funzionamento adeguato. Per abbattere i costi, garantire semplicità e accessibilità, non era possibile prevedere l’uso di due dispositivi diversi (un beacon e il proprio smartphone). Grazie alla grande diffusione dei circuiti B4.0+, quindi predisposti al BLE, dovuta al fatto che la quasi totalità delle aziende produttrici di dispositivi mobili ne prevede la presenza nei propri prodotti, ha portato dunque a pensare che un dispositivo potesse essere trasmettitore e ricevitore. Questo è il concetto chiave alla base delle AG ENAPI.

# **Apple/Google Exposure Notification API**

La pandemia globale di SARS-CoV-19 ha mostrato il limite della prassi per quanto concerne il contact tracing, rendendo in alcuni casi estremamente difficile risalire a una lista completa di contatti con un possibile infetto. In risposta alla problematica e alla necessità di una procedura più efficiente, in uno sforzo congiuto, Google ed Apple hanno sviluppato il sistema delle Exposure Notification API, nella speranza di fornire uno strumento efficace di contact tracing a disposizioni dei Governi nazionali e degli enti di sanità, pur preservando quanto più possibile la privacy degli individui.

Attraverso le AG ENAPI, un’applicazione permette, tramite l’utilizzo della tecnologia Bluetooth Low Energy, di scambiare con altri dispositivi su cui è presente questa app degli ID casuali, generati da una chiave personale, che rappresentano il “contatto” tra due dispositivi. In più, l’applicazione scarica dal server la lista delle chiavi di pazienti trovati infetti e cerca nella propria lista di id incontrati di recente degli id che corrispondano ad una delle chiavi. Se trova un risultato, l’applicazione notifica l’utente del fatto che un risultato è stato trovato, implicitamente che “si sia entrati in contatto con un individuo infetto”. L’applicazione poi provvederà ad indicare ulteriori istruzioni sulle operazioni da effettuare.  
Nel caso in cui invece l’utente dell’applicazione possa essere infetto, solo tramite autorizzazione (e in collaborazione con un operatore sanitario, in base alla regolamentazione delle autorità di salute pubblica a capo del progetto), può decidere di fornire le proprie chiavi relative agli ultimi 14 giorni di attività al server, in modo che gli altri dispositivi possano verificare se sono entrati in contatto con un individuo infetto.

Nel dettaglio, le AG ENAPI, utilizzano il Beaconing tramite Bluetooth BLE per scambiare messaggi. Tali messaggi sono rappresentati dal servizio bluetooth “Exposure Notification”, che contiene al proprio interno sia il Rolling Proximity Identifier, sia gli Associated Encrypted Metadata:

* Rolling Proximity Identifier: sono degli identificatori che rappresentano in maniera anonima il dispositivo comunicante. Questi sono generati a partire da una chiave, la Temporary Exposure Key, e sono passati tramite il servizio “Exposure Notification”
* Associated Encrypted Metadata: metadati aggiuntivi, come la potenza di segnale (per l’RSSI) e la versione del servizio
* Temporary Exposure Key: è la chiave che permette di creare i Rolling Proximity Identifier e che rappresenta in maniera anonima il dispositivo. La TEK viene cambiata dopo un certo periodo di tempo e vengono generati nuovi identificatori

La scelta del Beaconing tramite BLE è giustificata dal basso consumo energetico e dalla possibilità del BLE di una comunicazione bidirezionale, da intendersi come possibilità sia di fare annunci broadcast, sia di ascoltare per eventuali messaggi di annuncio di altri dispositivi, senza dover ricorrere alla connessione diretta (accoppiamento) tra i due dispositivi.

### BLE Beaconing AG ENAPI – Broadcast

La fase di Broadcast di un dispositivo con un’applicazione che sfrutta le AG ENAPI funziona come segue:

* Dopo l’approvazione del permesso all’uso dell’Exposure Notification, viene generata una Temporary Exposure Key. Da questa chiave vengono derivati il Rolling Proximity Identifier e gli Associated Encypted Metadata.
* Dopo la generazione dei precedenti, il sistema Bluetooth locale manda in broadcast annunci di tipo “Non-connectable undirected”, ossia diretti verso qualsiasi dispositivo in ascolto e senza necessità di connessione per la ricezione, con un intervallo di polling consigliato (“Exposure Notification – Bluetooth Specification v1.2.2” di Google) tra i 200 e 270 millisecondi.  
  Gli annunci sono composti di UUID dell’Exposure Notification Service, l’Identificatore e i Metadata associati.
* Il dispositivo in ascolto che può ricevere, salva sia l’Identifier che i Metadata.
* Dopo un intervallo di tempo, definibile manualmente (idealmente presentato come di 15 minuti, “Exposure Notification – Bluetooth Specification v1.2.2” di Google), vengono generati nuovi identificatori e metadati associati.  
  Ricomincia nuovamente il periodo di Broadcast.
* Dopo un periodo di tempo, definibile manualmente (idealmente presentato come di 24 ore, “Exposure Notification – Bluetooth Specification v1.2.2” di Google), viene generata una nuova Temporary Exposure Key, ripetuto il processo di derivazione di Identifiers e Metadata e ri-avviato il broadcasting.

### BLE Beaconing AG ENAPI – Scansione

Per quanto riguarda invece la scansione:

* Dopo l’approvazione del permesso all’uso dell’Exposure Notification, viene avviata una scansione per l’UUID del servizio Exposure Notification (durata massima di 5 minuti per catturare un messaggi, “Exposure Notification – Bluetooth Specification v1.2.2” di Google)
* Se catturato un broadcast Exposure Notification, vengono estratti e salvati sia il Rolling Proximity Identifier che gli Associated Encrypted Metadata sul dispositivo.
* Periodicamente, dall’applicazione, viene scaricata la lista delle Diagnosis Key, le chiavi dei dispositivi al cui utente è stato diagnosticato il virus.
* Una volta acquisite, vengono risolte le Diagnosis Keys con i Rolling Proximity Identifiers per verificare se, tra gli identificatori incontrati, c’è almeno un match con una Diagnosis Key.
* In base al risultato, viene calcolato il rischio associato all’esposizione e, in presenza di un rischio d’esposizione, viene generata una notifica.

# **Bluetooth Beaconing BLE – Aspetti implementativi**

Come mostrato dal lavoro congiunto di Apple e Google, è possibile utilizzare un dispositivo mobile (smartphone, per ragioni di semplicità, a meno di specificare diversamente) come beacon e ricevitore. Questo potrebbe rappresentare una svolta nel campo delle implementazioni Bluetooth, specie in relazione all’avanzamento nella diffusione dell’IoT.

Per quanto riguarda l’aspetto implementativo, considereremo solo il sistema operativo Android: BLE è disponibile da Android 4.1 in poi, ma soltanto da Android 4.3 c’è la possibilità di far operare il dispositivo sia in modalità periferica, come trasmettitore, che in modalità centrale, come ricevitore.  
Un dispositivo Android può effettuare scansioni per trovare dispositivi nelle vicinanze. In più, da Android 6.0, è possibile anche applicare dei filtri alle scansioni, in modo da ridurre il consumo energetico. Se il dispositivo effettua scansioni per dispositivi BLE, non può effettuare scansioni per Bluetooth classico e viceversa. Per esaminare come produrre un client e un beacon tramite BLE, utilizzeremo la libreria Android Beacon Library (Terze parti), che permette di rilevare e trasmettere come beacon, con compatibilità verso tutti gli standard maggiori di beaconing.

## Bluetooth Beaconing BLE – Client (Android Beacon Library)

Per quanto riguarda i permessi, l’app deve dichiarare i permessi:

* BLUETOOTH: per l’utilizzo della periferica Bluetooth
* BLUETOOTH\_ADMIN: per effettuare discovery e per la modifica dei parametri di configurazione del Bluetooth.
* ACCESS\_COARSE\_LOCATION: dato che la scansione può identificare oggetti usati per la geolocalizzazione, è necessario avere questi permessi per poter attivare la localizzazione. Disattivando la localizzazione, si disattiva lo scanner Bluetooth.  
  Quest’autorizzazione (e l’uso del servizio di localizzazione) può essere evitata se si utilizzano le Companion Device Manager API, che fanno discovery al posto del dispositivo stesso. Tale strumento è disponibile da Android 8.0 in poi.

Per poter usufruire dei servizi BLE è necessario che BLE sia supportato e che il servizio Bluetooth sia attivo. Se BLE è supportato, ma non attivo, è attivabile tramite il BluetoothAdapter: una volta istanziato un BluetoothAdapter, si può controllare lo stato di Bluetooth con “isEnabled()”. In caso di esito positivo, è possibile avviare un’attività, con un intent generabile a partire dal BluetoothAdapter: startActivityForResult(Intent in, Int i), con secondo argomento REQUEST\_ENABLE\_BT, una costante che indica i servizi Bluetooth.  
Il Client, per poter ricevere segnali dai Beacon, deve implementare BeaconConsumer, un’interfaccia che identifica il “consumatore” dei messaggi inviati dai beacon. Inoltre, deve istanziare un beaconManager, che si occuperà di gestire per intero le attività di monitoring. Il manager è composto di diversi componenti:

* Beacon Parsers: sono dei parser che permettono di comprendere i pacchetti di advertising fatti dai Beacons. Considerando che ogni protocollo ha il proprio modo di comporre i pacchetti, ogni parser che viene aggiunto al manager contiente il “layout” dei pacchetti che vogliamo poter ricevere. Il metodo “setBeaconLayout”, invocato alla creazione di un BeaconParser, permette di selezionare il layout che il parser deve catturare.
* Beacon Layout: come già specificato, sono i layout con cui sono strutturati i pacchetti in base ai protocolli usati dai beacon. I layout sono individuati da stringhe di termini, separati da righe, che individuano i campi e il loro significato. Prendendo l’esempio di AltBeacon:   
  ALTBEACON\_LAYOUT: "m:2-3=beac,i:4-19,i:20-21,i:22-23,p:24-24,d:25-25"  
  Altri layout supportati sono iBeacon e i frame di Eddystone.
* Region: un oggetto che contiene i criteri per catturare i pacchetti advertising inviati dai beacon. E’ composto di un UID che serve per distinguerla nel sistema. Quando si vuole effettuare monitoring usando una region, questa viene usata come riferimento per attivare e disattivare il monitoring. E’ composto anche da campi che servono per identificare possibili ID d’interesse per il ricevitore.
* MonitorNotifier: componente che funge da notificatore nel caso in cui il manager rilevi beacons che fanno matching con la Region dichiarata per cui si sta facendo monitoring. I metodi del notifier, da implementare, specificano il comportamento dell’applicativo nel momento in cui cambia lo stato del beacon rispetto a monitoraggio. E’ assegnato al Manager nel metodo “onBeaconServiceConnect()”, che viene implementato dall’interfaccia BeaconConsumer, per specificare come consumare i messaggi rilevati dai beacon.

Precedentemente si è accennato agli UUID di “profili SIG(-adopted)”: come specificato spiegando il funzionamento di GATT, gli UUID indicano “che servizio viene offerto dal GATT Server” o, in questo caso, a quale servizio fa riferimento il beacon. In genere ci sono due tipi di UUID:

* SIG-adopted: sono UUID registrati presso Bluetooth-SIG e fanno riferimento a profili di interesse generale. Un UUID SIG-adopted, oltre a contenere dettagliate informazioni circa l’utilizzo, si riconoscono in quanto sono identificati solamente per 16 bit: normalmente utilizzano una base speciale da 128-bit, che è la lunghezza standard degli UUID, composta come UUID:00000000-0000-1000-8000-00805F9B34FB. L’UUID SIG-adopted, avrà gli ultimi 2 bytes del primo termine settati per indicare l’UUID del profilo in maniera “abbreviata”.
* Custom UUID: si possono creare anche UUID custom, sempre nella forma standard specificata e sempre di 128-bit di lunghezza. La differenza sostanziale tra i SIG-adopted e i Custom UUID è la dimensione del pacchetto scambiato: poiché i SIG-adopted necessitano solo di 16 bit, i pacchetti inviati con profili aventi tali UUID saranno meno grandi.

## Bluetooth Beaconing BLE – Beacon (TODO, Android Beacon Library)

Da Android 5.0, se i dispositivi supportano BLE in modalità periferica, è possibile far funzionare il proprio dispositivo personale come beacon. Ma per prima cosa dobbiamo identificare che tipo di Beacon vogliamo creare e ciò dipende dal protocollo Beacon scelto.

## iBeacon

iBeacon nasce come protocollo Apple per il beaconing, nel 2013, ma è stato espanso per poter funzionare anche con dispositivi Android. Sempre basato su GATT, il beacon è definito da quattro valori:

* UUID: stringa di 16 byte per identificare univocamente il servizio del beacon.
* Major: stringa di 2 byte per identificare un sottoinsieme di beacons. Per capire meglio, un insieme di beacon che ha lo stesso Major è identificabile come “appartenente allo stesso servizio (o allo stesso individuo/entità).
* Minor: stringa di 2 byte per identificare il beacon individualmente. Ad esempio, in un insieme di beacon caratterizzati dallo stesso major, il Minor individua la singola unità a cui si è interessati. In un caso pratico, si hanno due beacon, uno all’ingresso (Minor = 1) di un negozio e uno all’interno (Minor = 0). Il Minor permette di identificare specificatamente l’identità del beacon stesso: se cerchiamo il beacon con Minor pari a 0, indichiamo il beacon all’interno del negozio.
* TX: o Transmission Power, indica quanta potenza avrà il segnale del beacon. Intuitivamente, viene incorporato nel pacchetto di advertising per fornire informazioni utili per l’RSSI (ossia per capire la potenza di segnale del beacon, fornendo informazioni importanti sulla prossimità del dispositivo).

## Eddystone

Eddystone è il protocollo di beaconing di Google. Sempre basato su UUID, si differenzia da iBeacon per avere dei “frames” che definiscono il contenuto del pacchetto. In particolare, ci sono 4 tipi di frames:

* Eddystone-UID: broadcast di un codice identificativo che permette all’app di prelevare informazioni dall’app server.
* Eddystone-EID: come un frame UID, ma l’identificativo è criptato.
* Eddystone-TLM: frame complementare, fornisce infomazioni circa lo stato del Beacon.
* Eddystone-URL: broadcast di un URL di massimo 18 caratteri, che ridirige il dispositivo ricevente verso un sito web.

## AltBeacon

AltBeacon è un protocollo open che mira all’interoperabilità tra i vari protocolli beacon. Un messaggio trasmesso con AltBeacon utilizza un unico ID che identifica il beacon e specifica il servizio e contiene informazioni RSSI.  
  
Ciò che è importante sapere di AltBeacon è che, nonostante il layout mostri come la struttura sia molto simile ad iBeacon, con spazi per tre ID, questi non sono propriamente “identificatori” di per sé. Per intenderci, il layout mostra la presenza di 3 Id:

* i:4-19 : ID primario, usato principalmente per l’UUID
* i:20-21 : ID secondario. Può corrispondere al “Major” del protocollo iBeacon
* i:22-23 : ID terziario. Può corrispondere al “Minor” del protocollo iBeacon

Questi tre identificatori sono usati anche per confrontare tramite “equals” due beacon AltBeacon. Tuttavia, è solo “formalità” che questi corrispondano ad UUID, Major e Minor.

In realtà negli ID possono essere codificate informazioni, purché rientrino nello spazio assegnato (due byte per ID 2 e 3, ad esempio) e purché vengano trasformate tramite parsing in “Identifier”.

Per la creazione di un esempio di questo sistema utilizzeremo le Android Beacon Libraries, che usano lo standard AltBeacon e permettono di interagire con diversi protocolli di Beaconing, compreso l’attuale sistema delle AG ENAPI.

# **BTPing – Toy case BLE Beaconing/Scanning**

## BTPing – Sintesi e permessi

BTPing è un toy case d’utilizzo del BLE per il Beaconing e Scanning, che utilizza la libreria open source “Android Beacon Library”. Il funzionamento è molto semplice: una volta scaricata e installata, l’app permette allo smartphone di fare scanning di beacon con un UUID fisso e/o di fare beaconing per farsi trovare da altri BTPing.

Una volta attivato lo scanner, i beacon in vista vengono aggiunti ad una lista e vengono mostrati all’utente con una tripla (UUID, ID2 e ID3, risp. Major e Minor). Nel momento in cui questi beacon scompaiono dal range dello scanner, tramite una notifica, viene avvisato l’utente della loro scomparsa e vengono eliminati dalla lista dei beacon in vista. Per ora, il funzionamento core dell’app è semplicemente questo.

Come ogni app che utilizza BLE, andranno aggiunti i permessi per Bluetooth e Localizzazione nel Manifest.xml:

<uses-permission android:name="android.permission.BLUETOOTH"/>

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN"/>

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>

<uses-permission android:name="android.permission.ACCESS\_BACKGROUND\_LOCATION"/>

<uses-feature android:name="android.hardware.bluetooth"

android:required="true" />

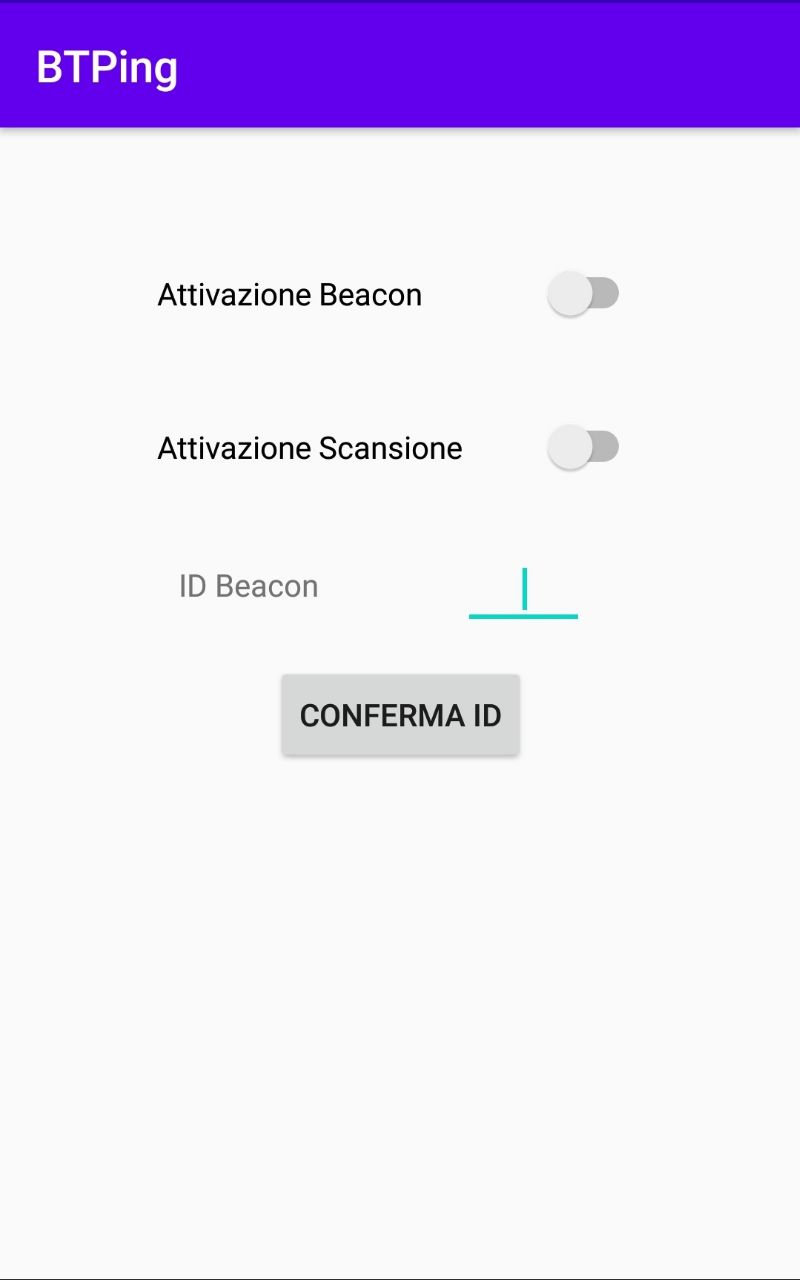
<uses-feature android:name="android.hardware.bluetooth\_le"

android:required="true" />

FINE\_LOCATION e BACKGROUND\_LOCATION sono necessari per i dispositivi con Android 10+ (API 29+). Questo perché sia il Beacon che lo Scanner sono posizionati in Services che computano in background e, a partire da Android 10+, si fa distinzione tra i processi in background e in foreground per quanto riguarda i permessi, per evitare comportamenti indesiderati. Dato che per l’uso di Bluetooth è necessaria la localizzazione (Bluetooth viene usato per aumentare la precisione del GPS e può raccogliere dati di RSSI, o potenza di segnale, per effettuare altri tipi di calcoli sempre relativi alla localizzazione).

## BTPing – Funzionamento e interfaccia

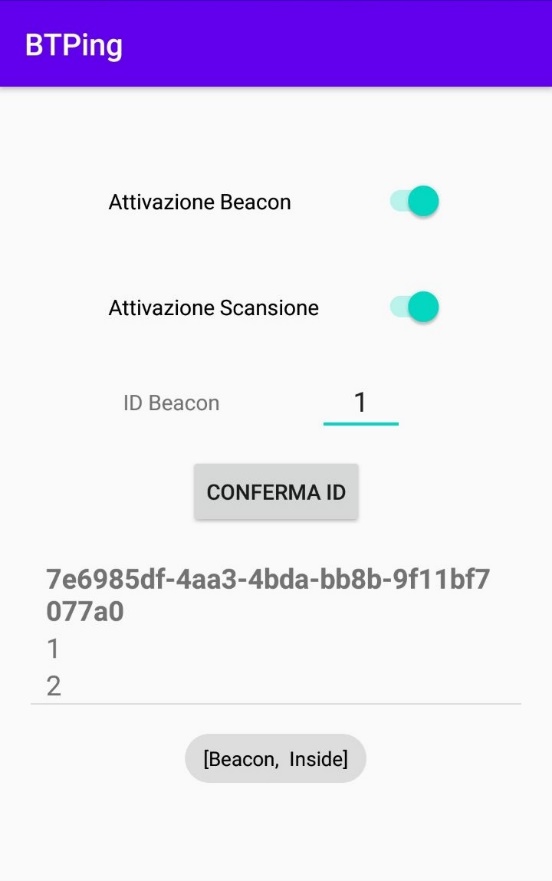
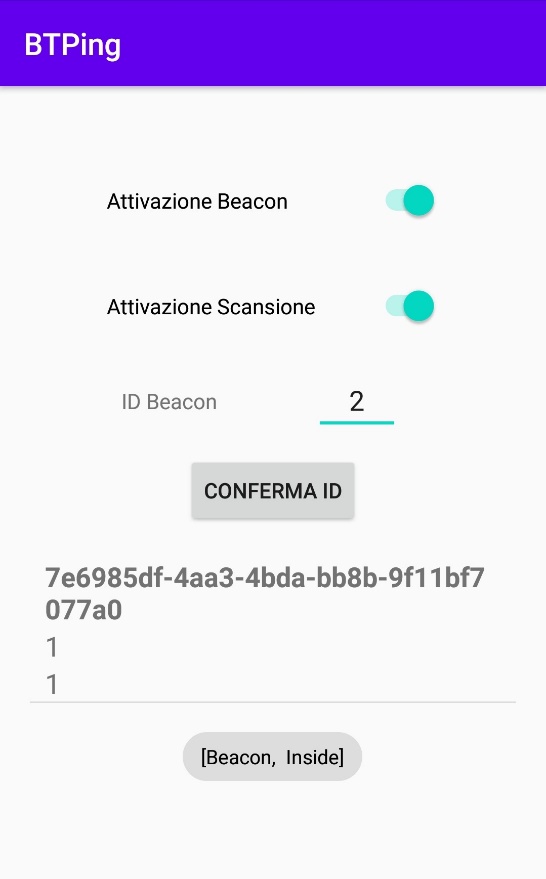
L’interfaccia dell’app è relativamente semplice e asciutta, data la semplicità in sé :



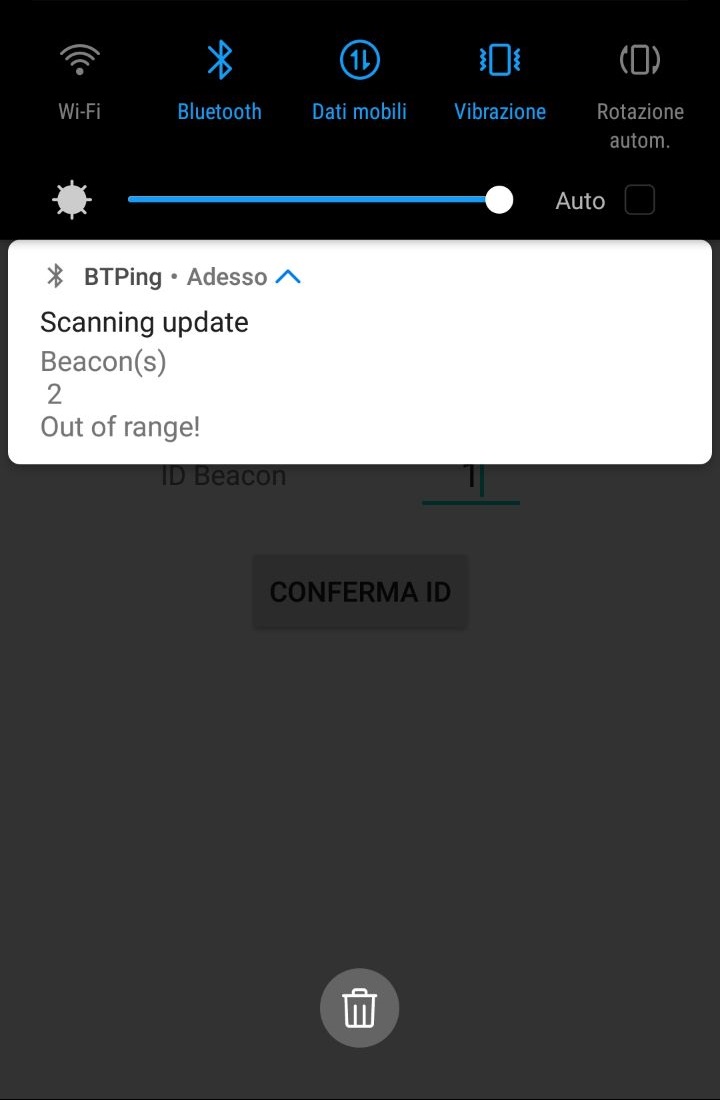
Come si può vedere sono disponibili due switch che permettono di interagire con i Servizi di Beaconing e Scanning: il primo attiva il servizio del beacon e l’advertising, il secondo attiva il servizio di scansione per i beacon. Per necessità di testing ed evitare di generare ID casualmente, rischiando che più beacon abbiano gli stessi ID, è fornita una text box dove inserire un numero che diventerà l’ID 3 del Beacon del dispositivo. Per confermarlo, è presente il pulsante “Conferma ID”. Se non è specificato nulla, l’ID 3 default è “1”.

Sebbene non visibile, al di sotto del pulsante “Conferma ID” è presente una List View che mostra i risultati delle scansioni.

Infatti, nel momento in cui è operativa, l’interfaccia grafica dell’applicazione si presenterà come segue:



In questo momento, i due dispositivi si “vedono” tra loro. Nel momento in cui uno dei due dispositivi disattiverà il beacon o spegnerà l’interfaccia Bluetooth, una notifica avviserà che un dispositivo (o i dispositivi) sono andati fuori range e specificherà il (o i) “minor” relativi.



Come è possibile vedere, non solo appare la notifica, ma anche la lista dei beacon verrà aggiornata di conseguenza.

Nella notifica, saranno presenti tutti gli ID dei beacon non più visibili. In questo modo, si potrà sapere quali dispositivi sono usciti dal raggio di ricezione del modulo BLE.

## BLE – Premesse su sviluppi sperimentali

Nonostante sia una semplice toy case, il proposito è quello di testare la funzionalità di Beaconing e Scanning, esattamente come per le A/G ENAPI. Lo sviluppo della tecnologia BLE è relativamente recente e l’avvento delle reti IoT potrebbe rappresentare un’occasione per la tecnologia BLE. Prima di procedere oltre, tuttavia, va analizzato ciò che rappresenta questo toy case:

Questo toy case ha mostrato, anche per dispositivi relativamente “vecchi” (Android 8.0.0), che il doppio comportamento BLE è una risorsa molto utile. Questo era già stato mostrato dalle App nazionali per lo smart contact tracing, che ha rappresentato il caso per eccellenza in cui il binomio BLE e Smartphone riesce a diventare estremamente utile, se non fondamentale. Tuttavia, la possibilità di replica di questo comportamento, nonostante le forti limitazioni dell’ambiente Android, comporta che sia possibile attuarlo su vasta scala, anche per propositi differenti rispetto allo smart contact tracing: BTPing nasceva con l’idea di poter controllare chi fosse nell’abitazione in ogni dato momento. Un impiego immediato era poter controllare dispositivi BLE equipaggiati a persone mentalmente instabili (quindi non-indipendenti), in modo da allarmare immediatamente in caso di evasione dal domicilio. Ma l’impiego non si limita solo a questo: i beacons possono esporre anche informazioni di modeste dimensioni (ordine massimo: 2 byte) e la libreria Android Beacon Library permette di implementare anche altri protocolli di beaconing, come iBeacon e Eddystone, diventando anche “cross-piattaforma” e incastrandosi anche con altri sistemi simili, come le Nearby API (Google). Per fare un esempio pratico di “cross-piattaforma”, specificando l’UUID corretto, è possibile vedere anche i beacon che trasmettono con UUID delle G/A ENAPI.

Un problema maggiore durante lo sviluppo, è relativo al come le interazioni all’interno dell’app si svolgono e alla necessità di “reattività automatica” dell’applicazione. Si pensi anche al solo aggiornamento della lista di beacon in tempo reale. Beacon e Scanning sono posti su due Background Service, per non appesantire l’Attività principale. Ciò impone che le informazioni vengano scambiate dai Servizi all’Attività tramite Broadcast, quindi abbiamo un Receiver che utilizzerà una callback nel momento in cui riceve un messaggio. Ma anche i componenti di beaconing e scansione (rispettivamente BeaconTransmitter e BeaconManager), nel momento in cui attivano i loro servizi, dispongono callback in risposta a determinati eventi. Ciò ha imposto l’utilizzo di Thread all’interno dei metodi di callback, in quanto il lavoro necessario era troppo pesante (lungo) da poter eseguire sia dentro le callback del BeaconManager, che è il componente dello scanner, quindi il responsabile primo del contenuto delle lista di beacon e del loro individuamento, sia dentro la callback del Receiver.  
In più, il sistema di scansione è molto più complesso: dispone di due componenti, una che esegue operazioni di “ranging”, ossia di scansione generale per trovare beacon visibili, e una di “monitoring”, che osserva se i beacon entrano od escono dalla Regione controllata. Potrebbe sembrare uguale, ma il monitoring si occupa del solo evento di entrata/uscita, il ranging si occupa di “vedere” i beacons (non vederli non è “un problema” della componente di ranging.

Quanto mostrato è l’esempio del Receiver che riceve i broadcast dei servizi. La logica nel caso “Beacon” dello switch potrebbe essere al “limite”, rispetto alla velocità e leggerezza richiesta dalla callback “onReceive”.

**[Da “Main Activity” – BroadcastReceiver receiver]**  
  
[---]

public void onReceive(Context context, Intent intent) {

[…]

switch (action) {

case SCANNING\_ACTION: {

String[] message = Objects.requireNonNull(intent.getStringExtra("SCANNING\_UPDATE")).split(":");

switch (message[0]) {

case "Info": {

if (message[1].contains("off")) scanSw.setEnabled(true);

break;

}

case "Beacon": {

ArrayList<Beacon> b = Objects.requireNonNull(intent.getParcelableArrayListExtra("BEACON"));

if (message[1].contains("Inside")) {

sauce.clear();

sauce.addAll(b);

} else sauce.removeAll(b);

listCustomAdapter.notifyDataSetChanged();

}

[…]

}

case BEACON\_ACTION: {

String[] message = Objects.requireNonNull(intent.getStringExtra("BEACON\_UPDATE")).split(":");

if (message[1].contains("off")) beaconSw.setEnabled(true);

Toast.makeText(getApplicationContext(), Arrays.toString(message), Toast.LENGTH\_SHORT).show();

break;

}

[…]

Ed è solo l’esempio di quanto le callback siano fragili, esaminando il codice contenuto nello ScanningService, queste operazioni sono state inserite in Thread in quanto, senza questi, erano troppo pesanti per essere gestite.

**[Da “ScanningService” – Operazioni contenute nei Thread]**

[---]

@Override

public void didExitRegion(Region region) {

new Thread(() -> {

if (!lostBeacons.isEmpty()) {

tellMain("Beacon: outside", lostBeacons);

StringBuilder sb = new StringBuilder();

for (Beacon b : lostBeacons) {

String beaconString = " " + b.getId3().toString() + "\n";

sb.append(beaconString);

}

showNotification("Scanning warning!", "Beacon(s)\n" + sb.toString() + "Out of range!");

}

}).start();

lostBeacons.clear();

}  
[---]

private void listBeaconUpdate(ArrayList<Beacon> list) {

lostBeacons = listBeacons;

lostBeacons.removeAll(list);

tellMain("Beacon: Inside", list);

listBeacons.clear();

listBeacons.addAll(list);

}  
  
[---]

Tuttavia, rimangono problemi risolvibili tramite utilizzo di pattern e best practices per quanto riguarda l’ambiente di sviluppo mobile.

## BLE – Possibili sviluppi sperimentali