UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

Progetto e Sviluppo di un Sistema per la Geolocalizzazione Indoor basata su tecnologia Bluetooth Low Energy

Luca Betti

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2016/2017

RINGRAZIAMENTI

Ringrazio l'Ing. Riccardo Martoglia, mio relatore, per la disponibilità e i consigli ricevuti per questo lavoro di tesi.

Ringrazio Chiara che ha sopportato le mie ansie durante gli ultimi esami ed è sempre stata al mio fianco.

Infine un ringraziamento va ai miei genitori e mia sorella che hanno sempre creduto in me e mi hanno dato la possibilità di arrivare fino a questo punto.

PAROLE CHIAVE

Bluetooth

BLE

Beacon

Wearable Device

Antenna

Indice

Elenco delle figure

Introduzione

Le connessioni wireless ormai da decenni hanno rivoluzionato il nostro modo di vivere, comunicare e viaggiare, portando ogni singola azione da “a portata di click” ad “a portata di smartphone”.
In particolare l’evoluzione della tecnologia *Bluetooth Classic* ha consentito un exploit di un nuovo mercato, quello dei wearable device. Infatti la tecnologia *Bluetooth 4.1*, chiamata anche *Bluetooth Low Energy* (BLE), non va vista tanto come una evoluzione dei precedenti standard Bluetooth ma come una tecnologia che si affianca per essere applicata in certe situazioni. Questo lo dimostrano le sue applicazioni su praticamente la totalità dei device indossabili e non solo, come nel caso di mobile advertising, proximity marketing and notice, healthcare & fitness application su smartphone e smartwatch.

Il lavoro di tirocinio svolto in Lapsy, start-up incentrata sullo sviluppo di sistemi e soluzioni Bluetooth per differenti segmenti di mercato, ha portato alla realizzazione di un sistema di geolocalizzazione indoor basata sulla tecnologia BLE. In particolar modo vengono analizzati i dispositivi iBeacon, che costituiscono i wearable device utilizzati per il progetto, e l’ utilizzo della tecnologia Bluetooth Low Energy.
Il lavoro di tirocinio nella start-up è stato pensato e modellato per un Hotel di Roma per tenere traccia degli spostamenti, all’interno dell’edificio, del personale delle pulizie. Viene approfondito quindi il comportamento e l’implementazione delle antenne, per l’individuazione dei dispositivi, in questo caso preso in analisi: per ogni beacon che entra o esce dalla zona, l’antenna notifica il server dell’avvenuto, anche nel caso di spostamenti bruschi o rilevanti all’interno della zona stessa.

Deve inoltre garantire il suo funzionamento anche dopo casi di blackout, potersi connettere automaticamente alla wifi/ethernet lan che è stata messa a disposizione ed aggiornarsi automaticamente alla versione più recente, quando necessario.

La tesi è strutturata su due parti principali, costituiti da 4 capitoli :

- *Caso di studio* comprendente:
  - Capitolo 1 - Tecnologie utilizzate, in cui vengono studiate le tecnologie ed analizzati gli strumenti messi a disposizione da Lapsy;

  - Capitolo 2 - Progetto Lapsy, per la descrizione delle scelte comportamentali dell'antenna e testing delle board per la sua costruzione.

- *L'applicazione* comprendente:
  - Capitolo 3 - Progettazione, prevede la raccolta dei requisiti funzionali da sviluppare e la progettazione del sistema senza entrare in dettagli implementativi.

  - Capitolo 4 - Implementazione, descrittivo della fase implementativa del progetto.

In ultima analisi viene presentato il lavoro di test fatto sul funzionamento del sistema e sui dispositivi beacon per il loro adattamento al progetto.
Seguono le conclusioni sul lavoro svolto e possibili sviluppi futuri.

Parte I

Il caso di studio

Capitolo 1

Tecnologie utilizzate

In questo capitolo vengono descritte in modo completo gli strumenti messi a disposizione da Lapsy per la costruzione del progetto.
Partendo da un caso iniziale, successivamente abbandonato, si ripercorrono le fasi di studio e scelta delle tecnologie adottate per il progetto.

1.1 Situazione iniziale

Il progetto destinato all'Hotel di Roma è stato pensato per individuare il personale della struttura, per monitorarne i movimenti nell'edificio. Il personale, in particolare gli operatori delle pulizie, deve essere dotato di braccialetti che possano inviare segnali bluetooth per farsi rilevare da antenne, programmate da Lapsy ed installate nei luoghi di interesse all'interno dell'edificio.
Inizialmente il progetto prevedeva l'uso di una board Arduino combinata con l'utilizzo di una antenna ad-hoc per ricevere determinati segnali di advertising.
L'antenna era completamente dipendente dalle caratteristiche della Arduino, data la mancanza di memoria disponibile e di un'interfacciamento a internet per collegarsi con i server. Operava in modalità master/slave, quindi ogni qualvolta riceveva un segnale di advertising, avvisava la board dell'avvenuto.
Arduino, un microcontrollore programmabile in C e/o C++, era visto come una possibile soluzione per il tipo di progetto, ma durante il periodo di tirocinio si sono analizzate ulteriori possibilità, che più si avvicinavano al concetto di mini-computer. Questa analisi, seguita successivamente da fasi di test per validarne il corretto funzionamento delle board, è dovuta principalmente alla mancanza nell'Arduino di una CPU e delle varie porte USB, Ethernet ed HDMI, queste fondamentali.

La programmazione di un Arduino si focalizza sull'interazione con segnali di input/output, quindi più di carattere elettronico, mentre altre alternative che andremo ad analizzare nel capitolo 1.3 implementano l'uso del sistema operativo della casa produttrice, o di una distribuzione Linux derivata da Debian, per la loro programmazione di I/O e gestione dei dati.

1.2 Introduzione iBeacon

Nella seguente sezione si vanno a spiegare gli studi avvenuti sul funzionamento del componente principale del braccialetto indossabile, l'iBeacon, e i vantaggi che offre la tecnologia da esso utilizzato per le comunicazioni con altri device, rispetto ad altre possibili scelte. Inoltre si da particolare attenzione alla struttura del protocollo adottata per il tipo di utilizzo.

Un beacon – che significa faro – è un piccolo trasmettitore che sfrutta il Bluetooth Low Energy, garantendo un basso consumo energetico mantenendo comunque un raggio d'azione variabile da pochi centimetri a 60-70 metri. Ci siamo affidati a una logica Master/Slave, lasciando che il segnale che emette continuamente e che viene spedito in Broadcast, chiamato segnale di advertising, potesse esser captato da qualsiasi dispositivo che supporti la tecnologia BLE, che sia uno smartphone o tablet o nel nostro caso un'antenna. Inoltre l' iBeacon, introdotta da Apple in concomitanza al rilascio di iOS 7, è alimentato da una batteria a bottone di una durata media di 2 anni.

Quando si parla di comunicazione wireless a bassa potenza, BLE non è l'unica tecnologia sul campo, infatti ha il suo stesso livello di capacità ZigBee, sua principale rivale, ma ciò che le contraddistingue maggiormente sono alcune delle loro capacità tecniche, che sono state fondamentali per il nostro tipo di progetto (si veda *[BLE]*). La prima caratteristica individuata è il data rate over-the-air,

dove ZigBee fornisce un rate di 125Kb/s contro i 1Mb/s per la tecnologia BLE. L'altra, come è già stata accennata, è il ridotto consumo di batteria, questa fondamentale per rendere il più longevo possibile la durata dei dispositivi, rimandando la sostituzione delle batterie a bottone ad anni di utilizzo. Questo provoca una diminuzione del range di azione/comunicazione rispetto alla tecnologia ZigBee, ma nel contesto in cui ci troviamo un range di 60-70 metri è più che sufficiente per coprire un intero edificio di piccole/medie dimensioni e quindi soddisfare i prerequisiti anche di progetti futuri. Si è pensato inoltre alla possibilità di far interagire, sempre per progetti futuri, i beacon con smartphone e tablet e la tecnologia BLE è perfetta per questo, e lo conferma la sua influenza su smartwatch, healthcare & fitness applications e mobile gaming (giochi in realtà virtuale aumentata – VR).

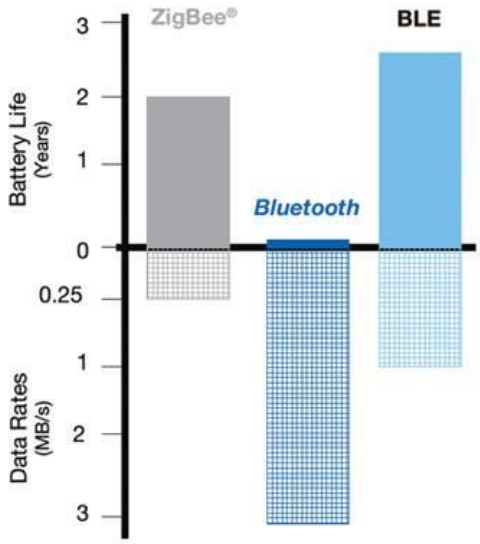

Figura 1.1: Confronto di durata della batteria e velocità dati tra ZigBee e BLE.

Lo spettro di 2.4GHz del Bluetooth prevede per la tecnologia LE (Low Energy) 40 canali da 1MHz, ma solo 3 verranno utilizzati per il trasferimento dei pacchetti di advertising. Questi sono i canali 37, 38, e 39 il resto viene inceve utilizzato per data exchange durante la connessione.

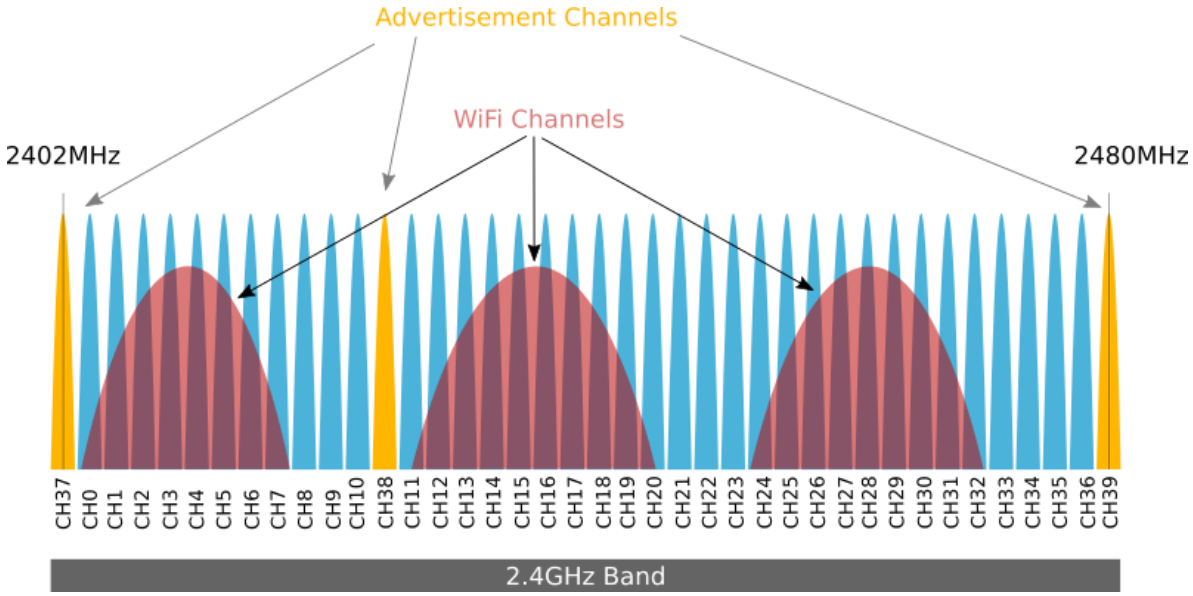

Figura 1.2: Spettro 2.4GHz Bluetooth tra 2402Mhz - 2480Mhz

Quando un device comunicante tramite BLE è in advertising mode, i pacchetti vengono spediti periodicamente su ogni canale di advertising. L'intervallo di tempo che intercorre tra un pacchetto e il successivo può essere scelto e fissato manualmente, con l'aggiunta di un random delay. Il primo tipo di intervallo può esser settato da 20ms a 10.24ms, in steps di 0.625ms. Il random delay è pseudo-casuale e va da 0ms a 10ms e viene automaticamente aggiunto. Questo serve per ridurre la possibilità di collisioni tra pacchetti di advertising di device differenti, il quale è fondamentale evitare per il progetto Lapsy. Questo va quindi ad aumentare la robustezza del Bluetooth Smart (o BLE).

Il protocollo adottato dagli iBeacon, nella quale Apple ha definito il suo Advertising Data, verrà spiegato nel capitolo successivo ma in generale, quando non è in uno stato di connessione con un altro dispositivo, prevede l'invio continuo di pacchetti di advertising composti da quattro principali informazioni: *UUID*, *Major*, *Minor* e *TXPower*. Queste informazioni, come tutte le altre che andremo ad analizzare successivamente, sono settabili solo tramite l'utilizzo del protocollo GATT, che permette di scoprire i servizi del dispositivo e di poterli leggere e modificare. Questo è possibile da terminale Debian o più semplicemente tramite app *nRF Connect* di cui è stato fatto utilizzo.

1.2.1 Protocollo iBeacon

Analizziamo il pacchetto che viene spedito dal iBeacon. Questo è composto da 4 principali sezioni: il *preambolo*, *l'access address,* la *PDU* (*Packet Data Unit*), e la *CRC.*

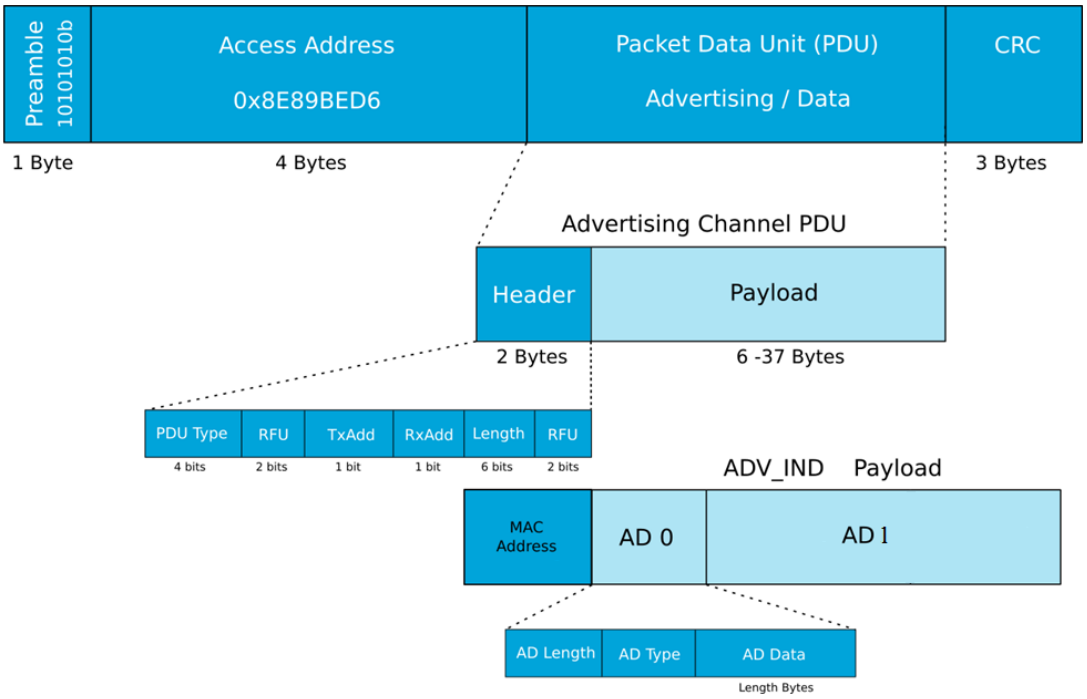

Figura 1.3: Protocollo iBeacon

Il pacchetto utilizzato per la trasmissione dati/advertising nella tecnologia Bluetooth Low Energy generalmente è costituito dal preambolo di un byte per la gestione interna del pacchetto, da un Access Address che ha sempre lo stesso valore fisso 0x8E89BED6 (4 bytes), dalla CRC che altro non è che la checksum, valore di 3 bytes calcolato sulla PDU per verificare la presenza di errori nel pacchetto durante il suo trasferimento, e la PDU. La sezione di maggior interesse è quest'ultima, che può essere di due tipi: o per la trasmissione dati tramite connessione con un'altro dispositivo o per segnali di advertising. Noi ci occuperemo di analizzare solo il secondo caso, la PDU advertising, che a seconda dei dati contenuti nella payload può raggiungere anche i 39 bytes di lunghezza.

1.2.2 PDU advertising

La PDU advertising è composta da un Header di 2 bytes suddivisa a sua volta in 6 campi da pochi bit ciascuno e dove il campo PDU Type, l'unico di nostro interesse, è quello che definisce come il beacon si comporta con gli altri dispositivi. Ci sono diversi tipi di PDU, quelli su cui ci focalizzeremo saranno **ADV_IND** e **ADV_NONCONN_IND**.

Il primo, che assume valore 0000, definisce un tipo di advertisment generico, è di fatti il più comune. Questo non è diretto verso uno specifico dispositivo ma lavora in modalità broadcast e inoltre permette la connessione, ciò significa che un "dispositivo centrale", come uno smartphone, può richiedere la connessione al beacon che trasmette, che si comporta quindi da "peripheral" o semplicemente host.

Per quanto riguarda **ADV_NONCONN_IND**, con valore 0001, si comporta in modo analogo al precedente per quanto riguarda il tipo invio segnali di advertising ma non permette la connessione.

Gli altri campi più piccoli e di ancor meno interesse sono i campi RFU che vengono mantenuti per scopi futuri, Txadd e Rxadd per personalizzare

maggiormente il tipo di advertise che si vuole spedire ed infine Lenght che mantiene la lunghezza del payload.

Analizziamo ora il cuore del messaggio: la PDU di advertising, spoglio del suo Header, contiene il MAC address Bluetooth e il payload. Il payload è composto di due strutture AD (Advertising Data structure).

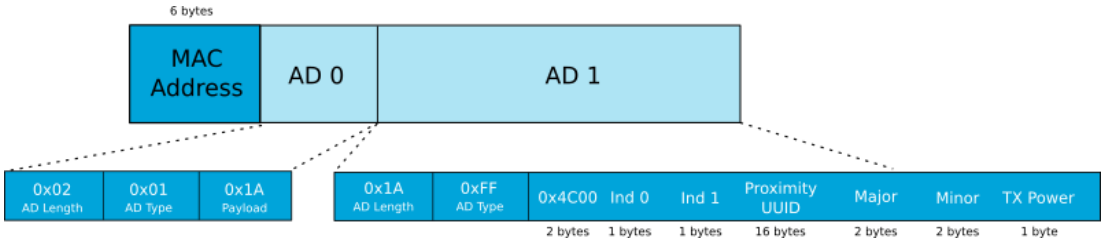

Figura 1.4: PDU pacchetto advertisement

La prima AD contiene informazioni generali e in particolare ha data type 0x01 per indicare il tipo di flags che lo seguono. Infatti nella parte Payload di AD 0 vengono utilizzati flags per definire le capacità del iBeacon e quali tipologie di LE supporta.

Con un esempio pratico, se nel campo AD Type si ha 0x01 e Payload 0x1A, dunque abbiamo che il valore dei flags è 0x1A = 000011010 che si traduce nella maniera seguente:

bit 0 (OFF) : indica *LE Limited Discoverable Mode*;

bit 1 (ON) : indica *LE General Discoverable Mode*;

bit 2 (OFF) : indica se *BR/EDR* è supportato. Utilizzato se l'iBeacon opera in Dual Mode;

bit 3 (ON) : indica se *LE* e *BR/EDR Controller* operano simultaneamente;

bit 4 (ON) : indica se *LE* e *BR/EDR Host* operano simultaneamente.

LE Limited Discoverable Mode : Il dispositivo è *Discoverable* (dunque invia continuamente segnali di advertising) solo per 30s circa.

LE General Discoverable Mode : Il dispositivo è *Discoverable* (dunque invia continuamente segnali di advertising) senza interruzioni, quindi fintanto che è alimentato e non connesso con altri device.

BR/EDR : Tecnologia bluetooth introdotta per la prima volta nella versione Bluetooth 2.0 che permette sia un Basic Rate di 1Mbit/s che un Enhanced Data Rate (EDR) per una maggiore velocità di trasferimento dati, che arriva fino a 2.1 Mbit/s.

LE e *BR/EDR Controller/Host* : Se BR/EDR è attivo, il dispositivo ne permette il suo utilizzo con uso simultaneo della tecnologia Low Energy in dual mode, comportandosi quindi o da Controller o da Peripheral. Un device che supporta il ruolo centrale può stabilire una connessione LE con altri dispositivi e qualunque altro device che la accetta ha un ruolo di Host o Peripheral.

Il più importante tipo di advertising data è la seconda AD (AD 1), contenente le informazioni specificate e standardizzate dall'azienda Apple sull'iBeacon.

I dati che risiedono nella seconda parte del payload, a differenza della prima AD 0, sono state pensate per poter identificare i dispositivi in maniera univoca. Questo metodo di identificazione è sicuramente necessario per capire quale beacon sta inviando pacchetti all'antenna ricevente ma queste sono informazioni di individuazione di alto livello, pensate per creare una vera e propria gerarchia per la visibilità di un beacon specialmente nei settori aziendali e di marketing.
La gerarchia è adattata a seconda delle esigenze, direttamente dal proprietario dei beacon, e le proprietà di questi ultimi possono essere modificate in qualsiasi momento.
Durante la spiegazione dei campi contenuti nel AD1 si andrà tramite esempio a spiegare come poter strutturare una gerarchia.

Il primo byte indica tramite un numero di bytes la lunghezza del Advertising Data 1. Con il valore 0x1A si indica la presenza di 26 bytes, 25 per il payload e 1 per il tipo di AD. Il tipo di AD viene indicato con 0xFF che altro non è che la *Manufacturer Specific*, ovvero il valore con la quale Apple identifica il suo Advertisement Data.
Seguono poi due bytes che indicano l'identificativo della società, dove 0x4C00 è il corrispettivo per la società Apple. Successivamente sono presenti altri 2 bytes che operano come indicatori di advertisement (advertising indicators) che hanno sempre valore 0x02 e 0x15.
Questi valori descritti sono valori che per lo più vengono ignorati a differenza dei successivi bytes.

I campi rimanenti sono quelli di maggior rilievo e vengono identificati, come già accennato precedentemente, come *Proximity UUID*, *Major*, *Minor* e *TXPower*.

UUID: Questa è una stringa di 16 byte utilizzata per differenziare grandi gruppi di beacon tra di loro. Diversi UUID possono essere adottati dallo stesso proprietario dei beacon, principalemente per identificare l'utilizzo, gruppi di

lavoro o settori. Ad esempio se il centro commerciale IperCoop gestisce una network di beacon, allora i segnali di advertising avviseranno gli smartphone, o altri strumenti di interesse, che nel reparto alimentari o vicino ai negozi o nei magazzini(zona di lavoro) si sono avvicinati e capirà, individuando il suo UUID, se è necessario avviare una determinata app, attivare una notifica sullo smartphone o qualche altra azione automatica.

Major: Questa è una stringa di 2 byte per distinguere un gruppo più piccolo di beacon.Questo può servire, ricollegandoci all'esempio precedente, per individuare in quale catena di negozi, o quale settore di negozi, la persona sta facendo acquisti. Nel campo della sicurezza, si può pensare al caso in cui una persona si stia avvicinando troppo a una zona soggetta a lavori in corso e una notifica o un allarme lo avvisa di mantenere le dovute distanze.

Minor: Questa è una stringa di 2 byte utilizzata per individuare l'esatto beacon. Quindi seguendo l'esempio del centro commerciale, ogni singolo negozio ha il suo minor, per indentificare con precisione la posizione.

Tx Power: La Broadcasting Power (o Transmit Power da cui prende il nome) e' una potenza del segnale usata per determinare la prossimità (distanza) dal beacon. Più precisamente essa indica la potenza del segnale che è stata impostata manualmente nella fase di configurazione mentre la potenza ricevuta, per capirne il significato in ordine di distanza, viene chiamata RSSI (Received Signal Strength Indicator), come fosse una unità di misura. La Tx Power può essere calibrata a piacimento dal proprietario dei beacon con un valore mantenuto in un range tra -40dBm e +4dBm. La potenza impatta direttamente il range su cui il segnale di advertising può agire: più la potenza è alta più il range aumenta e il segnale è forte. La RSSI è proporzionale alla distanza del device e al valore che è stato impostato inizialmente nella configurazione e anch'essa ha un range di valori compreso tra -20 e -100: per valori di *Transmit Power* massimi assumerà valori

vicini a -100 (potenza del segnale debole) a distanza di 55/60 metri, mentre per valori di Tx Power bassi, a soli pochi metri.
Naturalmente settare valori di potenza massimi ha effetti negativi sulla durata della batteria, e questo sarà uno dei principali problemi del progetto seguito in fase di tirocinio.

1.3 L'antenna

In questa sezione si spiega la funzionalità dell'antenna e si descrivono i due tipi di board utilizzate e testate per costituire l'antenna del progetto Lapsy. Nel capitolo successivo verranno spiegate quindi le differenze e i problemi che si hanno avuto durante la fase di testing dei dispositivi e che hanno portato a una scelta della board.

Nel progetto di tirocinio l'antenna è quel dispositivo dedito a riceve e riconoscere i segnali di advertising provenienti dai wearable device (iBeacon), individuandone la distanza da esso ed eventuali spostamenti. Queste informazioni devono essere quindi spedite al server messo a disposizione dall'azienda e in particolare deve esser capace di intuire quando un dispositivo entra nella zona ed esce in tempi ridotti, inviando al server segnali di log in e logout dalla zona.

L'antenna deve poter essere configurata alla prima accensione da un operatore, settandone quindi i propri attributi di identificazione al server e quelli necessari per filtrare o meno segnali di advertisement provenienti dall'esterno. La board sarà infatti programmata per considerare solamente segnali aventi UUID corrispondente a quello impostato su di esso, filtrando tutti gli altri messaggi di disturbo. Le caratteristiche che identificano l'antenna nel progetto sono invece il codice zona, che identifica una specifica stanza, aula o zona, e un tenant id, che specifica in quale edificio l'antenna è stata installata. L'antenna non necessita quindi di prestazioni hardware elevatissime, ma solo di per poter operare continuamente e di non dipendere da batterie esterne.

Le board su cui sono state effettuati test sono state la A10-OlinuXino-LIME e RaspBerry pi3 Model B, questo perchè mettevano a dispisizione porte USB, HDMI, ETHERNET e l'uso di una CPU a differenza della Arduino. Inoltre entrambe supportavano l'utilizzo di un sistema operativo Debian Linux e quindi la programmazione dell'applicazione in python, a differenza del classico Arduino, programmabile solamente in C++ tramite l'IDE Arduino Software, meno flessibile nel codice per ciò che si stava per andare a creare.

1.3.1 A10-OlinuXino-LIME

A10-OlinuXino-LIME è una single-board computer ovvero un calcolatore implementato su una sola scheda elettronica, pensata per creare applicazioni *IoT* , di hobbystica ma anche per progetti professionali (si veda *[A10]*).

Il progetto Olimex ruota attorno a un System-on-a-chip (SoC), che incorpora un processore A10 Cortex-A8 CPU 1GHz, una GPU Mali 400, e 512MB DDR3 di memoria RAM. La Olimex A10 non prevede né hard disk né una unità a stato solido, come quasi tutte le single-board computer, affidandosi invece a una scheda SD per il boot e per la memoria non volatile. Inoltre ha solamente 2 porte USB.
Vi sono due varianti della A10, quella che abbiamo appena analizzato e la A10-OLinuXino-LIME-4GB, la quale differisce dalla prima per una memoria NAND build-in da 4GB che permette l'uso di un sistema operativo senza il bisogno di una SD card. Vi sono inoltre le board A20-OLinuXino-LIME e A20-OLinuXino-LIME-4GB praticamente identiche alle, rispettivamente, A10-OLinuXino-LIME e A10-OLinuXino-LIME-4GB con la sola differenza del processore che è montato su di esse. Naturalmente questo porta vantaggi in termini di velocità ed efficienza: come già detto la A10 monta un Cortex A8 core mentre la A20 una dual-core Cortex A7, di gran lunga migliore. Questo porta la A20 ad essere più idonea e

potente per task computazionali più pesanti, e naturalemente richiedenti maggior quantità di energia.
Nel nostro caso Lapsy ha messo a disposizione due tipi di board: la A10 e RaspBerry pi3. La scelta della prima board è dovuta principalmente ai costi, molto più contenuti rispetto alla scelta di una A20.

1.3.2 RaspBerry Pi 3 Model B

Anche il Raspberry è una single-board computer (si veda *[Rbpi3]*) ma presenta caratteristiche, dal punto di vista delle prestazioni, migliori.
Essa presenta infatti una CPU da 1.2 GHz 64-bit quad-core ARM 7100 , memoria RAM da 1GB LPDDR2 e come la A10 non prevede né hard disk né una unità a stato solido e si affida a una SD card per il boot e memoria volatile.
Queste caratteristiche, che rendono il dispositivo già di un altro livello rispetto a quello precedente, presenta inoltre 3 caratteristiche fondamentali che hanno aiutato a prendere una decisione finale: la presenza sul Raspberry di 4 porte USB 2.0, di un collegamento di rete Ethernet 10/100 Mbit/s (come la OlinuXino) e di una Wireless LAN 802.11n integrata, così come è integrato anche il Blutooth, 4.1 eLE.

Capitolo 2

Progetto Lapsy

In questo capitolo vengono elencate le decisioni prese riguardo l'antenna ed il comportamento che essa dovrà adottare durante il suo funzionamento.
In primo luogo vengono descritti i tipi di test e i risultati che hanno portato alla scelta di una board per la costruzione dell'antenna, successivamente viene descritta la logica e le librerie adoperate per il suo funzionamento.

2.1 Studio e testing antenne

La scelta dell'antenna da utilizzare per il progetto di tirocinio, e in particolare per quelli futuri, è stata presa dopo un periodo di prova che consisteva nel controllare come i dispositivi OlinuXino A10 e RaspBerry Pi3 si comportavano in certe situazioni, sia dal punto di vista delle funzionalità di cui erano in possesso che delle prestazioni.

2.1.1 Tipi di test applicati

I test applicati alle board sono stati i seguenti:

- Controllo comportamento della board in presenza di segnali di advertising ad alta frequenza, anche provenienti da fonti diverse.
- Controllo temperatura raggiunta dalla board quando sottoposto alla presenza di segnali e per lungo periodo.
- Calcolo reazione/velocità di avviso ai server per presenza dispositivo.

Il primo tipo di test si basa sul controllo della lettura dei messaggi che gli iBeacon inviano continuamente. Questo implica che non solo deve riuscire a supportare la quantità di segnali in ingresso senza presentare rallentamenti(questi visibili attraverso la propria interfaccia grafica Debian su SDcard), ma anche riuscire a

vederli senza nessuna perdita di pacchetto(o comunque entro un margine di errore/perdita ristretto).
Il secondo tipo di test è stato necessario per evitare che un'antenna dopo poche ore sia in uno stato di surriscaldamento ed inutilizzo.
Infine il terzo test, molto connesso con i primi due, permette, durante la fase di implementazione, di comprendere quanto un'antenna sia reattiva nel percepire e contemporaneamente avvisare il server di movimenti e/o ingressi nell'area.

2.1.2 Costi antenne

Le due antenne a disposizione presentano un prezzo molto simile, con piccolo vantaggio economico per la OlinuXino A10. Quest'ultima board data l'assenza dell'integrazione dei moduli Bluetooth e Wi-Fi su di essa, ci ha spinto all'acquisto di ulteriore materiale, questo comprendente di USB bluetooth adapter e Wi-Fi adapter, materiale da dover fornire per ogni singola antenna di cui se ne farà uso.

2.1.3 Problematiche riscontrate e scelte prese

Una particolarità, che a primo impatto può sembrare banale ma non lo è affatto, è la presenza di sole due porte USB nella board Olimex. L'assenza di ulteriori porte utili per poter collegare mouse e tastiera, oltre alle due USB adapter, mette il programmatore in seria difficoltà. Nonostante tutto durante la fase di testing si è continuato a testare la board per verificarne le prestazioni alla presenza di segnali di advertising.

L'uso di USB adapter da parte della OlinuXino A10 permette al device di poter "vedere" i beacon vicino ad esso ma solo dopo una fase di loro configurazione. Quindi si è testata la reazione della board alla presenza di segnali di advertising. Si è rilevato che la board non è assolutamente adatta al tipo di lavoro: l'accensione di un beacon provoca un crash istantaneo per il dispositivo.

Dopo una serie di tentativi andati a vuoto, si è impostato sui beacon la frequenza della trasmissione dei segnali di advertising a un segnale ogni 2 secondi: il crash viene ritardato a qualche secondo, ma non evitato. Questa fase di testing è stata quella che ha inciso maggiormente sulla scelta della board da utilizzare. Probabilmente il crash è dovuto alle limitate prestazioni del processore Cortex-A8 a disposizione della board, per questo motivo si è successivamente testato il RaspBerry.
Il RaspBerry risolve gran parte, se non tutti, i problemi che affliggono la A10 OlinuXino.
Il numero di porte USB a disposizione è più che sufficiente per il loro utilizzo. Si ricorda infatti che i moduli Bluetooth e Wifi sono incorporati nella board, lasciando liberi altri 2 slot per usi alternativi.
Inoltre il raspberry con le sue prestazioni più elevate, sia in RAM che nella CPU, non soffre di nessun rallentamento o crash dovuto alla ricezione.

Il RaspBerry Pi 3 quindi permette l'uso di un unico elemento (la board) privo di componenti aggiuntive, concedendo performance migliori e spese inferiori a quelle della A10-OlinuXino-Lime, se si tiene conto degli adapter. La board inoltre non presenta casi di surriscaldamento dopo lunghi periodi di lavoro (ricezione segnali bluetooth e in secondo momento invio al server). Pertanto il RaspBerry viene individuato come antenna per il progetto Lapsy.

2.2 Proximity & scene analysis

Il compito principale delle antenne è quello di individuare i wearable device quando entrano ed escono dal loro raggio di azione e quando all'interno della zona rilevata si muovono.
Si punta quindi a ridurre al minimo le connessioni e i calcoli: occorre un approccio semplice ed economico.

Le connessioni vengono adoperate in particolare nei passi iniziali per la configurazione del beacon, per il settaggio dei valori di frequenza del segnale, potenza del segnale e per acquisire il valore di batteria attuale del dispositivo. Fuori da questa fase di configurazione, sfruttano a pieno potenziale i segnali di advertising per farsi individuare e garantiscono la continuità nel loro funzionamento senza interruzioni, oltre che fornire un sostanziale aumento nel loro risparmio energetico.

Si usa quindi una tecnica a metà tra *proximity* e *scene analysis*.

La *scene analysis* è una tecnica di location sensing che sfrutta oggetti e caratteristiche di un'ambiente visti da una certa angolazione per calcolare la posizione dell'osservatore o degli oggetti stessi dell'ambiente. In paricolare si parla di *differential scene analysis* quando si cercano le differenze tra una scena e quella successiva e quindi si individua la posizione stimata degli oggetti.

La *proximity analysis* invece è una tecnica che consiste nel determinare quando un oggetto è "vicino" a una posizione prestabilita, ovvero quella dell'antenna, che è nota (si veda *[PA]*). L'oggetto è individuato tramite fenomeni aventi un determinato range, nel nostro caso i segnali di advertising. Esistono tre approcci principali per determinarne la presenza o meno: tramite contatto fisico, wireless mode oppure facendo uso di automatic system ID, presenti principalmente su carte di credito o nelle codici a barre.

Nel nostro progetto si fa uso di *proximity analysis* per rilevare nella zona un eventuale dispositivo entrante (ed uscente) appena viene rilevato un segnale esterno e di *scene analysis* per comprendere se questo si sta muovendo o è in una posizione fissa, prendendo in considerazione la potenza del segnale che stiamo ricevendo: se rileviamo sbalzi di potenza di segnale rilevanti, questo indica sicuramente uno spostamento del wearable device.

Una terza tecnica che non è stata accennata è la *angulation analysis*, basata sulle potenze "sentite" da ogni antenna in riferimento ad un'unico beacon: naturalmente richiederebbe più antenne installate in una singola stanza, che non è il caso del progetto in esame. Quindi non ci interessa una esatta posizione calcolabile solo

con *angulation analysis*, ma ci limitiamo a inviare al server segnali di login, logout ed “presente” quando in movimento.

2.3 Librerie adoperate

Data la limitata complessità del software nell’antenna non sono state adoperate librerie particolari, fatta eccezione per JSON, request, wifi e subprocess.

subprocess : Il modulo subprocess permette di creare nuovi processi e di connettersi ai loro “pipe” di input/output/error e quindi di ricavarne il codice di ritorno. Nel progetto seguito subprocess è stato ciò che ci ha permesso di far comunicare l’applicazione dell’antenna con i beacon: infatti nessuna libreria o modulo esterno è stato pensato nel linguaggio python per la ricezione dei segnali di advertising.
Dopo lunghe ricerche su possibili progetti passati di altre aziende o della stessa casa produttrice del segnali di advertising Apple, nessuna soluzione adottabile è stata trovata. Si è quindi ricercato tra i comandi *builtin* Bash e sono stati individuati due comandi per “sentire” i segnali di advertising:
hcitool [-i <hciX>] [command [command parameters]]
ed
hcidump [option [option...]] [filter]
Subprocess si occupa di combinare le funzionalità delle sopracitate espressioni sfruttando un vero e proprio parallelismo di esecuzione tra di esse.
Il primo comando generalmente viene utilizzato per configurare le connessioni Bluetooth o per inviare determinati messaggi a Bluetooth device. In particolare con l’opzione ‘`-i`’ si può specificare con quale interfaccia Bluetooth (dell’antenna) comunicare ed eserguire il comando datogli. Nel nostro caso, il comando di interesse era `lescan`, il quale individuava ogni dispositivo Bluetooth

con tecnologia Low Energy (da qui il nome LE-scan). Quindi una lista di MacAddress dei dispositivi veniva stampato su terminale.

Questo non è il risultato finale che si voleva ottenere ma combinato con il comando `hcidump --raw` dunque venivano visualizzati non i tipi di pacchetti e loro MacAddress ma solo i dati in essi contenuti. Nella tesi, durante il capitolo di Implementazione del progetto di tirocinio, verranno mostrati gli output dei comandi sopracitati e le relative modifiche che si sono dovute applicare per un corretto funzionamento.

Subprocess per garantire il parallelismo dei due comandi fa uso della classe `Popen` che crea i processi e li gestisce. Si propone un esempio per semplificarne la comprensione:

```
command = "timeout -s SIGINT 1s hcitool -i hci0 lescan"
args = command.split()
p1 = subprocess.Popen(args,
                      stdout=subprocess.PIPE,
                      stderr=subprocess.PIPE
                     )
# equivalente a scrivere:
#    subprocess.Popen(['timeout','-s','SIGINT','1s','hcitool','-
i','hci0','lescan'],..,..)
testo, errore=p1.communicate()
# communicate() permette l'avvio del processo e ritorna i valori di
uscita
```

Esempio utilizzo libreria subprocess

JSON: (JavaScript Object Notation) è un semplice formato per lo scambio di dati. E' un formato human readable, perchè di facile lettura e scrittura, mentre per le macchine risulta facile da generare e analizzarne la sintassi. Ha una sintassi minimale ed è portabile poiché si basa su 2 strutture fondamentali:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un dizionario, una tabella hash, un elenco di

chiavi o un array associativo. Nel nostro caso, in python, viene considerato un dizionario, con coppia key:value

• Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza. Queste caratteristiche fanno di JSON un linguaggio ideale per lo scambio di dati.

Request: Per l'invio e ricezione dei dati tra server ed antenna, è stato scelto l'uso di *request*, una libreria HTTP con licenza Apache2, scritta in Python. Request si fa carico di tutto il lavoro per implementare HTTP/1.1 su Python rendendo immediata l'integrazione delle applicazioni con i web services. Non c'è bisogno di aggiungere manualmente query string agli URL, o di fare form-encoding dei dati di POST. Inoltre il Keep-alive e il pooling delle connessioni HTTP sono 100% automatici. Un'alternativa era il modulo urllib2 della libreria standard Python che mette a disposizione quasi tutte le principali funzionalità HTTP ma la sua interfaccia è molto frastagliata, serve più lavoro ed addirittura anche l'overriding di metodi per realizzare il più semplice dei task. Si propone di seguito un esempio di comune utilizzo:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import urllib2

gh_url = 'https://api.github.com'

req = urllib2.Request(gh_url)

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')

auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)
```

```
urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.getcode()
print handler.headers.getheader('content-type')


# ------
# 200
# 'application/json'
```

Esempio utilizzo libreria urllib2

In un classico utilizzo delle funzionalità della libreria urllib2, vengono spese molte righe di codice solo per settare la password e username per l'`opener`, da utilizzare nell'handler, che permette l'apertura della url. Da esso si reperisce lo status del server (200) e il tipo di contenuto (di tipo json) .

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import requests

r = requests.get('https://api.github.com', auth=('user', 'pass'))

print r.status_code
print r.headers['content-type']

# ------
# 200
# 'application/json'
```

Esempio utilizzo libreria request

L'utilizzo della libreria request non solo limita notevolmente l'uso di codice ma fa uso di funzioni chiare e semplici per il programmatore.

Wifi: python WiFi è un modulo Python che fornisce l'accesso in lettura e scrittura alle capacità di una scheda di rete wireless utilizzando le estensioni wireless di Linux (si veda *[PW]*). Si è trovata la necessita del suo utilizzo per permettere all'antenna una connessione di tipo wireless con internet in maniera automatica con i dati pre-configurati dall'operatore alla prima accensione. Di questa libreria sono stati utilizzati principalmente i moduli Cell e Scheme: il primo individua tutte le connessioni disponibili sulla wlan prescelta, Scheme invece è lo schema con la quale viene rappresentata la connessione scelta.

Le Cell possono essere individuate anche da shell con `iwlist wlan scan` :

```
# iwlist wlan0 scan
wlan0     Scan completed :
          Cell 01 - Address: 02:0F:B5:4F:74:ED
                    ESSID:"mynetwork"
                    Mode:Ad-Hoc
                    ...
          Cell 02 - Address: 02:0F:B7:5D:24:EF
                    ESSID:"lapsynetwork"
                    Mode:Ad-Hoc
                    ...
```

Output 'iwlist wlan0 scan' da linea di comando

Uno schema è strutturato nella seguente forma:

```
scheme = Scheme.for_cell('wlan0',ssid,cell,passkey)
scheme.save()
scheme.activate()
```

Esempio utilizzo del modulo wifi.Scheme

Dove `scheme.save()` salva in memoria lo schema utilizzato e `scheme.activate()` permette di connettersi alla linea scelta.

2.4 Throttling

Il server messo a disposizione dalla Lapsy ha lo scopo di ricevere qualsiasi segnale venga emesso da tutte le antenne e di poter tener traccia quindi di tutti i beacon presenti. Deve inoltre tener aggiornati i RaspBerry, inviando quindi ogni 5 minuti circa un segnale in output.

Il server, senza nessun tipo di controllo, viene sottoposto a un rate elevato di messaggi di input/output al minuto, proporzionale al numero di beacon ed antenne, per questo è stato necessario nella parte client (quindi delle antenne) implementare un Throttling process. Questo processo consiste nel limitare il numero di messaggi per ogni antenna da inviare al server per evitare un surriscaldamento di quest'ultimo e/o un inaspettata diminuzione delle prestazioni.

Per prima cosa si è diminuito il rate di ricerca dei segnali di advertising a 1 secondo, dato che ci troviamo in un progetto che non richiede tempi di avviso immediato, come nel campo della sicurezza. Inoltre, solo per i segnali di "presenza" in una stanza, la segnalazione al server avveniva solo con spostamenti rilevanti del device, evitando quindi la cattura di variazioni della potenza di segnale dovuti a gesticolazioni. Questo è stato implementato calcolando la media della potenza percepita per gli ultimi 5 segnali di presenza ricevuti, quindi se un margine di errore si presenta tra la potenza attuale e quella media calcolata, il segnale di "presenza" viene spedito al server. Il numero di segnalazioni quindi è stata ridotta notevolmente, mantenendo comunque una buona velocità di ricerca dei device.

Parte II

L’applicazione

-

Geolocalizzazione indoor

basata su tecnologia Bluetooth Low Energy

Capitolo 3

Progettazione

La progettazione del software prevede raccolta dei requisiti funzionali da sviluppare e una descrizione completa e non ambigua delle funzionalità principali del software che è stato realizzato e le rispettive caratteristiche. Essa illustra le scelte fatte riguardo ai compiti che il software svolge, senza entrare in dettagli implementativi.

3.1 Requisiti funzionali

Di seguito verranno descritti i requisiti funzionali, quindi il comportamento del sistema in risposta a determinati input e le principali operazioni che vengono svolte all'interno del software dell'antenna.

3.1.1 Inserimento dati di configurazione

L'inserimento dei dati di configurazione è l'unica azione a diretto contatto tra un'operatore e le antenne. Questo avviene alla prima accensione del dispositivo e mediante interfaccia grafica per semplificarne l'utilizzo. L'azione di configurazione consiste nell'impostare i codici delle zone e degli edifici in cui le antenne verranno collocate durante la fase di installazione e del filtro UUID, decisivo per comprendere quali segnali di advertising scartare e quali elaborare. Vengono inoltre considerati dati per stabilire connessioni con rete WiFi, nel caso

di assenza del cavo Ethernet. I dati di configurazione quindi vengono memorizzati in un file a parte, in formato *.ini* , riutilizzabile dalla stessa antenna nel caso di eventuali reboot o blackout.

RF01	Antenna	Configurazione
Input	Codice zona (cod_zona), codice edificio (ten_id), filtro UUID, dati connessione wifi.	
Processo	Scrittura e salvataggio dei dati di configurazione su file *.ini* .	
Output	File *.ini* contenente i dati, rappresentativo per l'antenna.	

Tabella 1.1 : Requisito funzionale 1

3.1.2 Riconoscimento dei segnali di advertising

Il seguente requisito funzionale consiste nel comprendere quali dati del pacchetto di advertising che viene ricevuto rappresentano le caratteristiche e gli identificativi dei beacon. Queste sono necessarie per capire quale device, e quindi quale persona, è dentro alla zona di visibilità dell'antenna.

RF02	Segnale advertising	Riconoscimento
Input	Segnali di advertising	
Processo	Operazione di riconoscimento dei segnali	
Output	Insieme dei valori che caratterizzano i beacon trovati: UUID, major, minor e TxPower	

Tabella 1.2 : Requisito funzionale 2

3.1.3 Memorizzazione ed aggiornamento beacon presenti

Dopo il riconoscimento, i valori dei beacon vengono memorizzati con orario(tempo) in cui è stato trovato e con le caratteristiche dell'antenna. Quest'ultime vengono anch'esse salvate per far si che al momento dell'invio dati al server, nel caso di entrata 'E' uscita 'U' o presente (movimento rilevante) 'P', si possa capire in quale zona e in quale edificio il beacon è stato trovato/uscito.
Queste caratteristiche dell'antenna rappresentano quindi la zona , che nel caso del nostro progetto può esser vista come una camera o un salotto, e l'edificio nel quale si trova, unico nel nosto lavoro.

RF03	Caratteristiche beacon	Inserimento
Input	UUID, major, minor, TxPower e tempo corrente.	
Processo	Aggiornamento valori dei beacon già presenti ed inserimento per quelli nuovi, appena entrati nella zona.	
Output	Una lista rappresentante tutti i beacon presenti e con valori di TxPower e tempo corrente aggiornati agli ultimi valori trovati.	

Tabella 1.3 : Requisito funzionale 3

3.1.4 Filtro beacon fuori zona

Effettuato l'inserimento e l'aggiornamento di tutti i dati dei beacon presenti nella zona coperta dall'antenna, vengono filtrati i device che non sono stati rilevati per più di 10 secondi. Questa informazione viene vista infatti come una loro assenza nella zona, quindi viene informato il server di una uscita 'U' del device.

RF04	Beacon	Eliminazione
Input	Lista beacon trovati ed aggiornati	
Processo	Individuazione beacon che non sono più visibili dall'antenna da più di 10 secondi	
Output	Lista beacon aggiornata e filtrata dei dispositivi fuori dalla zona dell'antenna	

Tabella 1.4 : Requisito funzionale 4

3.2 Requisiti non funzionali

I requisiti non funzionali analizzati per il progetto riguardano le proprietà del sistema e le caratteristiche dei servizi che offre. Essi definiscono i vincoli di sviluppo del sistema.

Il numero di antenne richieste dall'hotel che fanno uso dello stesso software sono state 25, anche se il numero massimo di antenne può estendersi fino a massima richiesta. I device sono infatti indipendenti l'uno dall'altro e possono inviare dati al server in qualsiasi momento, per questo si è dovuto implementare un throttling process su ogni dispositivo per evitare perdita di prestazioni da parte server. Inoltre ogni antenna può supportare la manipolazione e la ricerca istantanea, quindi senza perdite di tempo e segnali, di un alto numero di beacon, che si aggira sui 15/20 elementi.

Il sistema si basa sulla memorizzazione di dati in locale, quindi non utilizza DBMS ed è modulare e diviso in base al tipo di operazione, in modo da rendere più facili la modifica e il controllo delle singole parti.

3.3 Scenario: gestione segnali di advertising

Il beacon dentro la zona coperta da antenna invia segnali di advertising, contenenti anche valori indesiderati. In primo luogo viene verificato che il beacon sia individuabile dall'antenna (cioè appartenente allo stesso progetto) con un controllo sulla corrispondenza del valore UUID. Successivamente per i beacon "visibili" vengono filtrati i valori di interesse (UUID, major, minor, TxPower) e memorizzati in una classe, rappresentante i beacon trovati. Si crea così una lista dei beacon trovati su cui verranno applicati i controlli di *movimento*, *uscita dalla zona* ed *entrata nella zona.*

Ogni controllo emette un segnale di avviso al server, indicandone la causa ('P' *presente*, 'E' *entrata*, 'U' *uscita*) e inviando i dati caratterizzanti del device e dell'antenna.

Quindi la lista dei beacon presenti nella zona occupata dall'antenna, aggiornata dei nuovi valori di tempo e potenza media calcolata, è pronta per il riutilizzo.

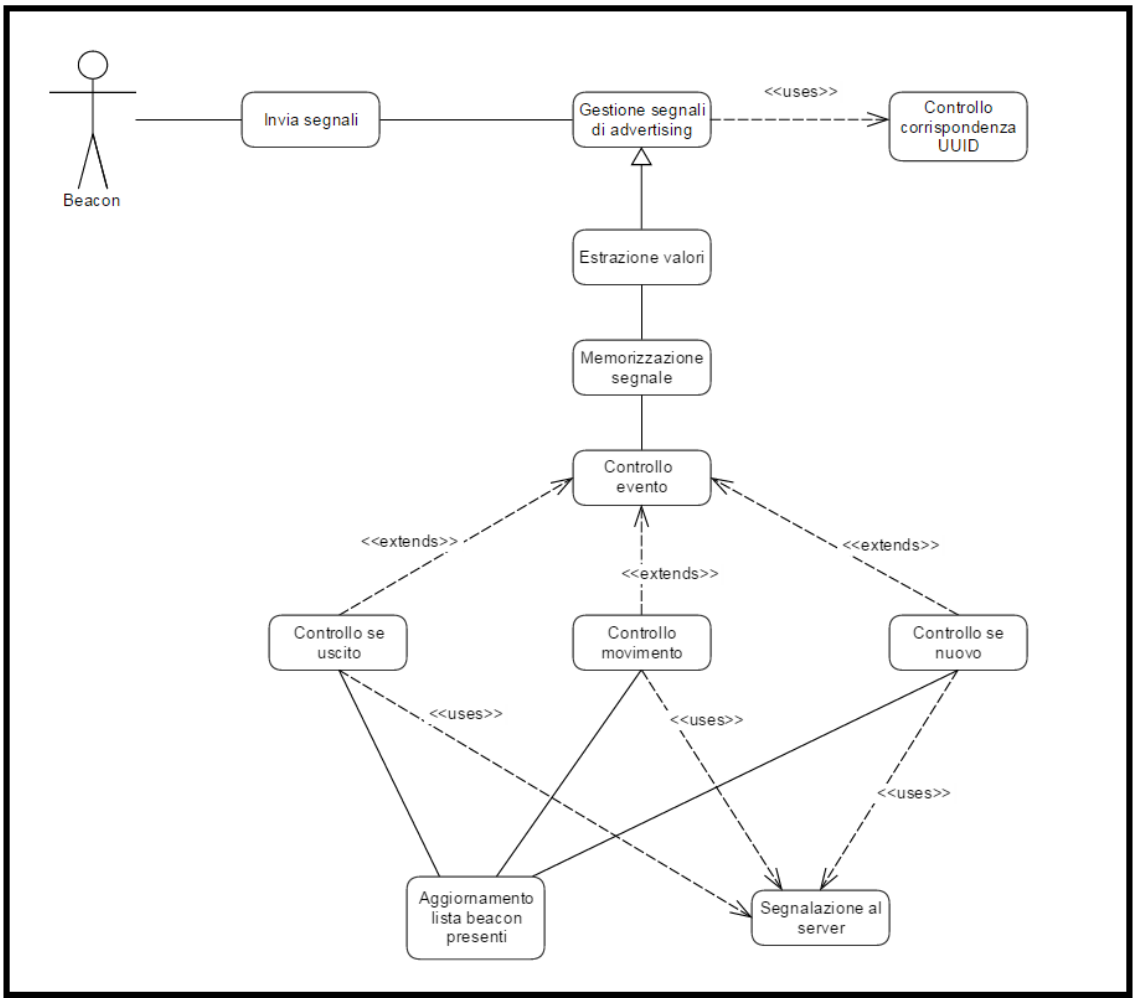

Figura 3.1: Scenario - Gestione segnali di advertising

3.4 Activity Diagram

Alla prima accensione la fase di configurazione richiede l'inserimento delle caratteristiche dell'antenna, salvate successivamente su file. Vengono quindi settati i valori dell'antenna e creata una connessione a una rete wifi se disponibile. Ogni 30minuti viene fatta una ricerca per gli aggiornamenti di sistema, con eventuale loro applicazione e reboot dell'antenna.
Nel caso nessun aggiornamento sia necessario, si passa alla ricezione dei segnali di advertising, riconoscendone le caratteristiche principali (UUID, major, minor e TxPower). Una prima fase di controlli viene eseguita per rilevare movimenti bruschi dei dispositivi o nuovo ingresso nella zona coperta da antenna. Quindi viene avvisato il server con un segnale json. Un ulteriore controllo viene eseguito per ripulire la lista dei beacon presenti da quelli che non inviano segnali da più di 10 secondi (device usciti dalla zona), con corrispondente avviso al server. Quindi la lista dei beacon presenti è pronta per il riutilizzo.

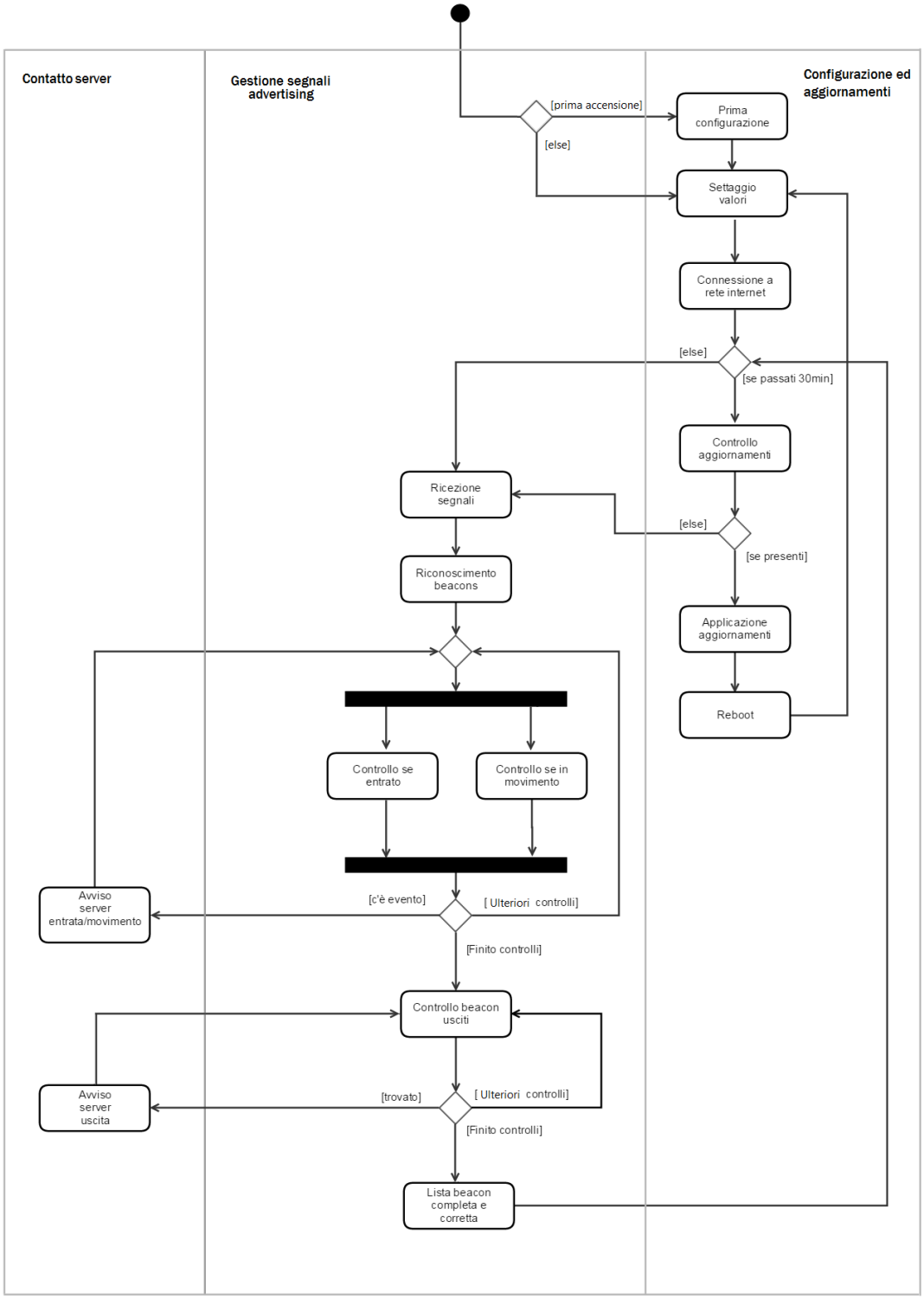

Figura 3.2: Activity Diagram caso generale

3.5 Le strutture Antenna e Beacon

Per la realizzazione dei dispositivi le informazioni di base si sono concentrate nella classe Dispositivo. Queste rappresentano per l'antenna i dati caratteristici, adottati invece dai Beacon come informazioni di "zona" in cui si trovano. Sono state pensate per la classe Antenna e Beacon i metodi riportati in *Figura 3.3* discusse nel capitolo *4. Implementazione*, tutti modularizzati per rendere la comprensione, manutenibilità e gestione da parte del programmatore facile ed intuitiva.

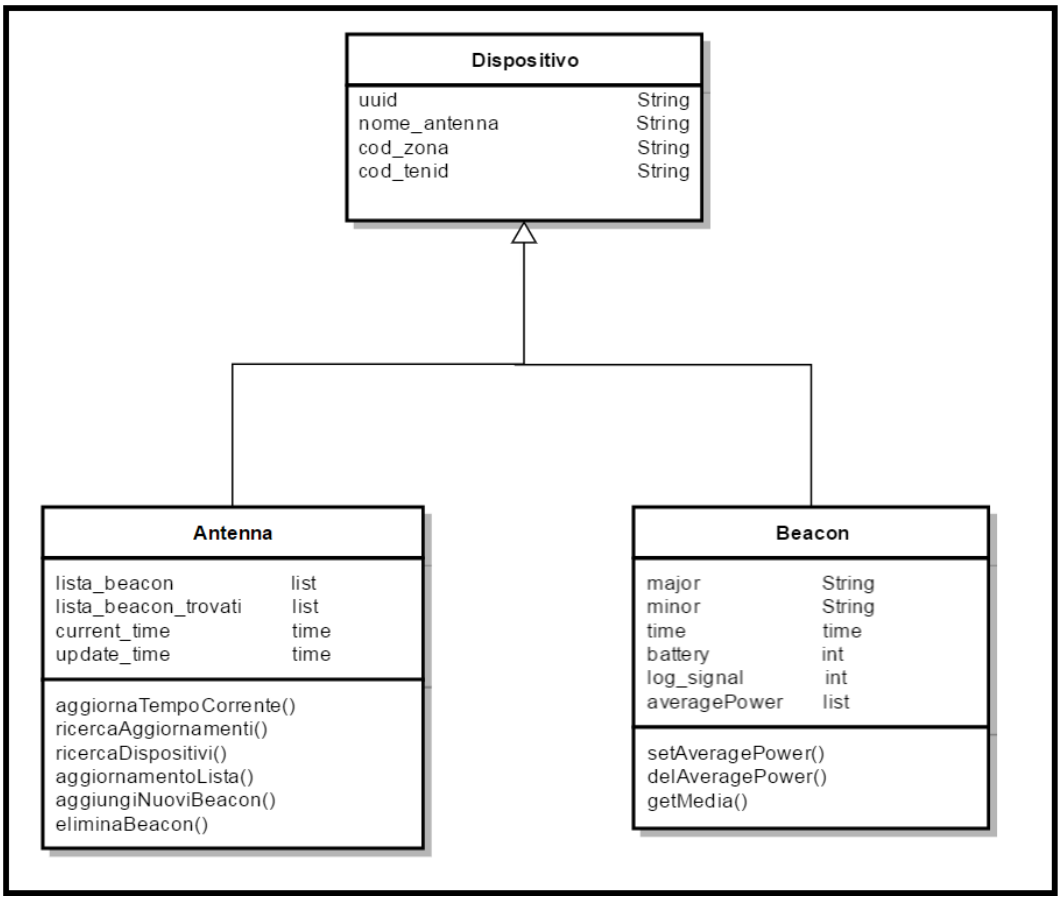

Figura 3.3: Diagramma delle classi - Struttura dispositivi

Nel campo averagePower vengono memorizzate e gestite le potenze dei segnali rilevati.

I segnali possono trasmettere valori di TxPower "oscillanti", con un errore che non può essere trascurato. Ogni potenza rilevata viene interpretata dal sistema come una certa distanza del wearable device dall'antenna quindi per comprendere quando un dispositivo si sta muovendo nella zona, viene calcolata la differenza tra la potenza attuale del beacon e una media delle ultime 5 potenze trovate, anzichè il semplice *"ultimo valore rilevato"*, per prevenire informazioni errate ed inutili notifiche al server, dovute ai valori oscillanti.

La lista quindi viene aggiornata continuamente degli ultimi valori ottenuti tramite le funzioni *setAveragePower* e *delAveragePower*. getMedia invece viene adoperata per il calcolo della media degli ultimi 5 valori. Uno sguardo più approfondito ai metodi viene dato in *4.1.7 Classe Beacon*.

Capitolo 4

Implementazione

In questo capitolo viene analizzato il progetto dal punto di vista implementativo. Viene descritta la configurazione dei dati dell'antenna e il sistema di ricerca aggiornamenti, per passare successivamente alle funzionalità principali dell'antenna e avviso server.

4.1 Struttura del progetto

L'avvio del sistema parte dal *mainLapsy.py* , il quale gestisce, solo per il primo avvio del RaspBerry, il modulo *ConfigurazioneLapsy.py* per l'inserimento da input dei valori dell'antenna, salvati in *tmp/configuration.ini.* Quindi si passa alla fase di settaggio dei valori in *src/settings/SettaggioAntenna.py* e se necessario anche alla connessione a internet tramite wifi. L'esecuzione principale del sistema si dirama su tutti i moduli implementati nella cartella *src/* . In particolare *StrutturaAntenna.py* e *StrutturaBeacon.py* definiscono le classi rappresentanti i device nei loro attributi e funzionalità.

AggiornamentoLista.py ed *EliminaBeacon.py* sono i moduli che con l'utilizzo di *InvioServer.py* permettono di tenere aggiornato il server dei movimenti da parte dei beacon tra le antenne installate.
Infine Update.py consente il trasferimento di messaggi in formato json tra antenne e server.

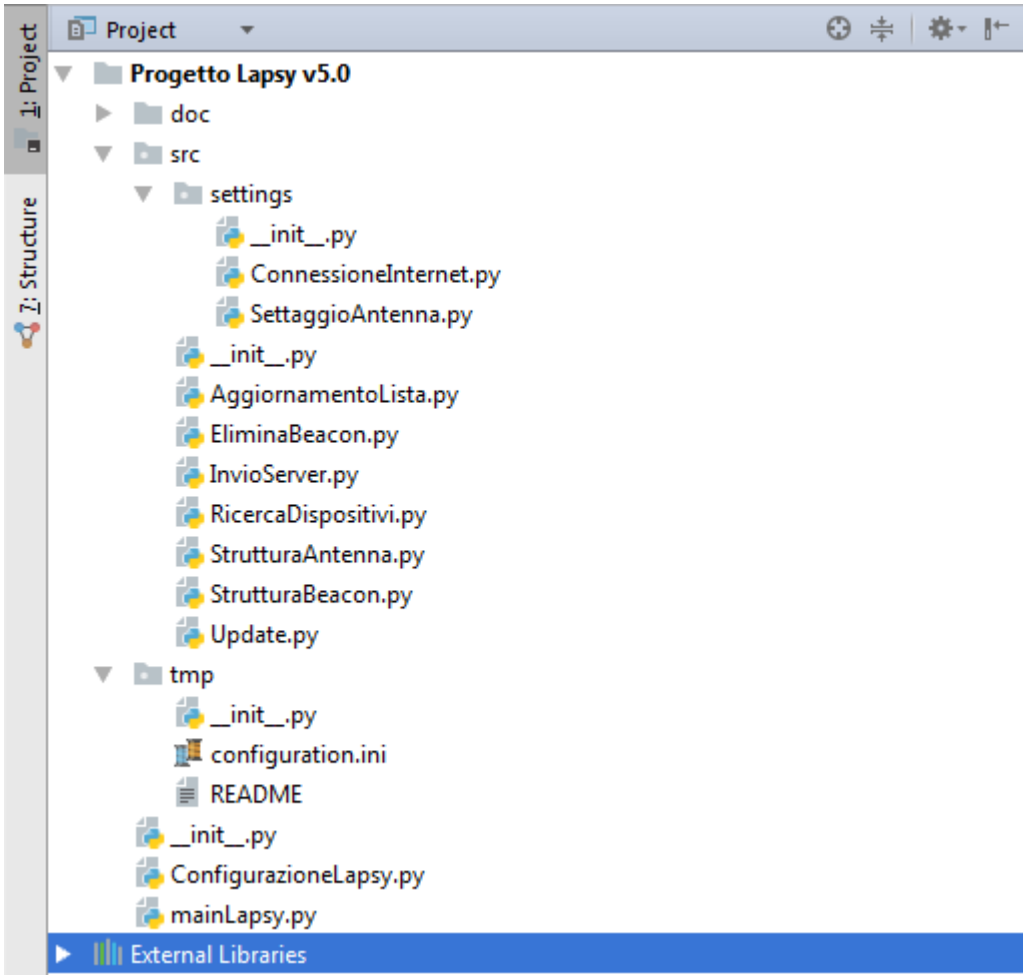

Figura 4.1: Struttura del progetto

4.1.1 mainLapsy.py

Al *main* viene lasciato il solo compito della chiamata delle funzioni nei moduli: la prima per la gestione della configurazione dei dati e relativo settaggio nell'antenna, dopodichè inizia la fase di update, ricerca beacon e controllo dei loro movimenti, il tutto all'interno di un ciclo *while true*.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
from src.StrutturaAntenna import Antenna
from src.settings.SettaggioAntenna import setAntenna

if __name__ == "__main__":

          # Si controlla se il file di configurazione (configuration.ini) esiste già.
          # Questa azione rappresenta la prima accensione del dispositivo.

          if os.path.exists("/home/pi/ProgettoLapsy/tmp/configuration.ini"):
             print "Already configured"
          else:
             # Importo la parte grafica per la configurazione manuale
             import ConfigurazioneLapsy
             ConfigurazioneLapsy.inizio()

          # Ricevo un dizionario dei valori/attributi della antenna in questione .
          # Questi valori vengono prelevati dal file di configurazione
configuration.ini .
          dict_attributi_antenna = setAntenna()

          # Creo un'istanza della classe Antenna
          antenna = Antenna(dict_attributi_antenna)

          while True:

          # Ricerchiamo eventuali aggiornamenti.
                    antenna.aggiornaTempoCorrente()
                    antenna.ricercaAggiornamenti()

                    antenna.lista_beacon_trovati = []

          # Sezione di lettura dei segnali di advertising.
                    antenna.ricercaDispositivi()

          # Aggiorno la lista di beacon trovati con relativi valori.
                    if len(antenna.lista_beacon) > 0:
                      antenna.aggiornamentoLista()

          # Se ho trovato beacons nuovi, li inseriamo nella nostra lista dispositivi e
avvisiamo il server.
                    antenna.aggiungiNuoviBeacon()

          # Eliminiamo eventuali beacons di cui abbiamo perso traccia da oltre 10s.
                    if len(antenna.lista_beacon) > 0:
                      antenna.eliminaBeacon()


          #fine while true:
```

Codice 4.1: Fase di chiamata alle funzioni dei metodi

4.1.2 Modulo ConfigurazioneLapsy

Per la configurazione dell'antenna è stata implementata una semplice interfaccia per un più facile interfacciamento del sistema con il programmatore. Questa è stata ideata costruendo la classe *simpleapp_tk(Tkinter.Tk)* con una fase di inizializzazione per la parte grafica, facendo uso della libreria Tkinter, una libreria molto comune per python.

```
class simpleapp_tk(Tkinter.Tk):
  def __init__(self,parent):
     Tkinter.Tk.__init__(self,parent)
     self.parent = parent
     self.configure(background='grey')
     self.initialize()

  def initialize(self):
     self.grid()

     self.label = Tkinter.Label(self,text="UUID", bg="blue", fg="black",relief=RAISED)
     self.label.grid(column=0,row=0,sticky='EW')

     self.uuid = Tkinter.Entry(self, bd=5)
     self.uuid.grid(column=1,row=0,sticky='EW',pady=20, padx = 20)

     […]

     button = Tkinter.Button(self,
                    text="Configura",
                    command=self.controlla,
                    activebackground='red',
                    activeforeground='white')
                    button.grid(column=2, row=4, pady=10, padx=10)
     […]

     self.grid_columnconfigure(0,weight=1)
```

Codice 4.2: metodo initialize di simpleapp_tk

Ogni bottone viene gestito chiamando il metodo della classe simpleapp_tk che gli è stato associato durante la sua creazione.

Di seguito viene proposto il funzionamento del metodo *scrivi()* .

```
def scrivi(self,scelta,ssidc,pswc):
        " Funzione per la scrittura delle configurazioni"
        import ConfigParser
        Config = ConfigParser.ConfigParser()

        # Creiamo un file di configurazione per il prossimo riavvio...
        cfgfile = open("tmp/configuration.ini",'w')
        section1='Antenna'
        Config.add_section(section1)
        Config.set(section1,'UUID',self.uuid.get())
        Config.set(section1,'Nome',self.antenna.get())
        Config.set(section1,'Codice Zona', self.codzona.get())
        Config.set(section1,'Tenant id', self.tenid.get())

        section2='Connessione'
        Config.add_section(section2)
        Config.set(section2,'Scelta',scelta)
        Config.set(section2,'SSID',ssidc)
        Config.set(section2,'Password', pswc)

        Config.write(cfgfile)
        cfgfile.close()
        tkMessageBox.showinfo("Avviso","Salvataggio avvenuto")
```

Codice 4.3: metodo scrivi di simpleapp_tk

Il metodo scrivi() si occupa della scrittura del file configuration.ini per la configurazione dell'antenna. Il file è stato strutturato in 2 sezioni:

- **Antenna** racchiude il filtro uuid dell'antenna, con relativo nome, codice zona e tenant id. Questi verranno adottati per costruire successivamente la classe Antenna all'avvio del programma.
- **Connessione** con il campo *Scelta* descrive il tipo di connessione effettuato per l'accesso alla rete internet. Quindi se presenti, vengono memorizzati anche i campi *SSID* e *Password*; In loro assenza si presume sia presente una connessione lan.

Figura 4.2: contenuto e struttura di configuration.ini

4.1.3 Modulo ConnessioneInternet

Per la funzione accessoInternet è stata utilizzata la libreria wifi che permette la ricerca di reti disponibili in zona. A livello di comando da shell la libreria richiede la scansione con il comando *iwlist* delle reti e ne analizza l'uscita.

```
from wifi import Cell,Scheme

def accessoInternet(ssid, passw):
          " Funzione per l'accesso a rete internet tramite wifi/lan"

          cell = Cell.all('wlan0')
          for cella in cell:
            if cella.ssid == ssid:
              if Scheme.find('wlan0',ssid)  ==  None:
                    scheme = Scheme.for_cell('wlan0',
                                             ssid  ,
                                             cella ,
                                             passkey=passw
                                           )
                    scheme.save()
                    scheme.activate()
              else:
                    scheme = Scheme.find('wlan0',ssid)
                    scheme.activate()
```

Codice 4.4: accessoInternet del modulo ConnessioneInternet

Per ottenere un elenco delle diverse celle della zona, dove per celle si intende ogni connessione wifi disponibile, è stato dato il comando Cell.all(). Ciò restituisce un elenco di oggetti Cell, ognuno con diversi attributi di cui è di nostro interesse 'ssid'. Quindi viene filtrata la connessione specificata da utente e creato uno schema, salvato ed attivato. In un eventuale riavvio del dispositivo, la funzione ricerca tra gli schemi già esistenti se è presente quello con ssid corrispondente, evitando un incrementale creazione di schemi.

4.1.4 Modulo SettaggioAntenna

Per settare i valori dell'antenna, questi vengono estratti dal file di configurazione quindi i valori caratteristici del dispositivo vengono salvati e mandati alla classe Antenna sottoforma di dizionario. Per la connessione invece si passano i valori come parametri di ingresso alla funzione accessoInternet descritta in *4.1.3 Modulo ConnessioneInternet.*

```
def setAntenna():
        "Funzione per estrazione configurazioni beacon da file .ini"

        # Lettura configurazione dell'antenna
        Config = ConfigParser.ConfigParser()
        Config.read("tmp/configuration.ini")

        #creazione dictionary info antenna
        dict_one = dict(Config.items('Antenna'))

        myFilter = dict_one['UUID']
        nomeAntenna = dict_one['Nome']
        cod_zona = dict_one['Codice Zona']
        cod_tenid = dict_one['Tenant id']

        #creazione dictionary info connessione
        dict_two = dict(Config.items('Connessione'))

        tipo_connessione = dict_two['Scelta']

        if tipo_connessione == 'wifi':
                ssid = dict_two['SSID']
                psw = dict_two['Password']
                print 'Connettendo alla rete wifi ' + ssid + '...'
                ConnessioneInternet.accessoInternet(ssid,psw)
                print 'Connessione riuscita.'

        return dict_one
```

Codice 4.5: funzione setAntenna in modulo SettaggioAntenna

4.1.5 Classe Dispositivo

Dispositivo rappresenta la classe padre per le classi Antenna e Beacon. Da qui vengono settati i valori rappresentanti l'antenna. La scelta di rendere anche Beacon una classe figlia di Dispositivo è stata fatta perchè, all'invio dei segnali al server, questo potesse contenere tutte le informazioni necessarie per l'invio, facendo quindi uso di un singolo parametro, l'oggetto rappresentante il beacon.

```
class Dispositivo(object):
          def __init__(self,*args):
                    self.UUID = args[0]
                    self.nomeAntenna = args[1]
                    self.cod_zona = args[2]
                    self.cod_tenid = args[3]
```

Codice 4.6: Classe Dispositivo in StrutturaBeacon

4.1.6 Classe Antenna

La classe Antenna richiama il costruttore *(__init__)* di Dispositivo per settare i valori ricevuti da *setAntenna* descritta in *4.1.4 Modulo SettaggioAntenna*, il tempo corrente e le liste di beacon nella zona coperta da essa.

```
class Antenna(Dispositivo):
      " Classe che rappresenta una antenna nelle sue caratteristiche e proprietà"

      def __init__(self,dict_attributi):
          super(Antenna,self).__init__(dict_attributi['UUID'],dict_attributi['Nome'],d
ict_attributi['Codice Zona'],dict_attributi['Tenant id'])
            self.lista_beacon = []
            self.lista_beacon_trovati = []
            self.currentTime = int(round(time.time() * 1000))
            self.timeUpdate = int(round(time.time() * 1000))

      […]
```

Codice 4.7: Costruttore di classe Antenna in StrutturaAntenna

I metodi implementati per la classe sono quelli richiamati da main, durante il ciclo *while True*. Molti di essi fanno riferimento a funzioni costruite in moduli differenti per mantenere un alto livello di manutenibilità. Queste sono *ricercaDispositivi*, *aggiornamentoLista*, *aggiungiNuoviBeacon* ed *eliminaBeacon*.

Come si può notare in *Codice 4.8*, vi sono due ulteriori funzioni: la prima per l'aggiornamento del tempo corrente, la seconda per la ricerca degli aggiornamenti. Quest'ultima avviene nel momento in cui il *currentTime* dell'antenna differisce di un certo valore da *TimeUpdate*, corrispondente nel nostro caso a 30 minuti (1800 secondi).

Se il controllo è vero, dunque viene invocata la funzione *ricercaAggiornamenti* nel modulo *Update*. Se nessun aggiornamento è stato necessario, si aggiorna il timeUpdate per riavviare il "contatore" del tempo trascorso da una ricerca all'altra.

```
[…]

def aggiornaTempoCorrente(self):
      self.currentTime = int(round(time.time() * 1000))

def ricercaAggiornamenti(self):
      " Funzione per ricerca aggiornamenti e reboot se necessario "

      if abs(int(round(self.currentTime)) - int(round(self.timeUpdate))) > 1800:
                print "Checking for new version..."
                evento = Update.ricercaAggiornamenti()
                if evento == "current version":
                  print "No updates found."
                  self.timeUpdate = int(round(time.time() * 1000))
                else: # necessita di un reboot per applicare le modifiche
                  print "Updated to version ",str(evento)
                  os.system('/sbin/shutdown -r now')

[…]
```

Codice 4.8: funzioni aggiornaTempoCorrente ed ricercaAggiornamenti della classe Antenna

4.1.7 Classe Beacon

La classe Beacon richiama il costruttore *(__init__)* di Dispositivo per settare i valori ricevuti da *addNewBeacons* descritta in *4.1.9 Modulo AggiornamentoLista*, con relativo *major*, *minor*, *time* per tener traccia di quando il dispositivo è stato trovato, *battery* al momento non utilizzato nel progetto, *log_signal* ovvero il livello medio di potenza del segnale e la lista averagePower contenente le ultime 5 potenze di segnale rilevate per il corrispettivo beacon, utile per il calcolo della media e per rilevare spostamenti bruschi.

```
class Beacon(Dispositivo):
      " Classe che rappresenta un beacon nelle sue caratteristiche e proprietà"
      def __init__(self,major,minor,time,uuid,nome,codzona,tenid,lsig):
            super(Beacon,self).__init__(uuid,nome,codzona,tenid)
            self.major = major
            self.minor = minor
            self.time = time
            self.battery = 0
            self.log_signal = lsig
            self.averagePower = []
      […]
```

Codice 4.9: Costruttore di classe Beacon in StrutturaBeacon

Tra le funzioni implementate abbiamo:

- setAveragePower per aggiungere nella lista delle potenze rilevate quella dell'ultimo segnale sentito dall'antenna.

```
      […]

      def setAveragePower(self,power):
          " Aggiunge una potenza rilevata "
          self.averagePower.append(abs(power))
```

Codice 4.10: funzione setAveragePower di classe Beacon in StrutturaBeacon

- delAveragePower per eliminare la potenza meno recente rilevata.

```
      def delAveragePower(self):
          " Elimina la potenza rilevata meno recente "
          del self.averagePower[0]
```

Codice 4.11: funzione delAveragePower di classe Beacon in StrutturaBeacon

- get_media calcola e restituisce la media delle potenze memorizzate.

```
def get_media(self):
    " Calcola e restituisce la media delle potenze "
    somma = 0
    for val in self.averagePower:
            somma = somma + val
    return somma/len(self.averagePower)
```

Codice 4.12: funzione get_media di classe Beacon in StrutturaBeacon

Tutte queste funzioni verranno adoperate in *updatePropBeacons* di *AggiornamentoLista*.

4.1.8 Modulo RicercaDispositivi

La ricerca dei segnali di advertising è composta principalmente da due fasi: la prima, quella più stretta al termine "ricerca", scannerizza tutti i segnali BLE in ingresso all'antenna, la seconda reperisce solo le caratteristiche utili dai segnali.

```
def ricercaDispositivi(self,myFilter):
    "Funzione per il reperimento e filtraggio delle caratteristiche utili dei beacon "

    dump,err = scan()

    if err != '' or dump == 'Set scan parameters failed: Input/output error.':
        gestioneErrori()
        dump, err = scan()

    p = ''
    myFilter = formattaUUID(p.join(myFilter.split("-")))
    myFilter = myFilter.upper()
    indice_appoggio = 0

    # si cerca dentro al testo dump la presenza di myFilter (l' UUID)
    for indice in range(len(dump)):

          # per ogni presenza di myFilter, si salvano i dati annessi al dispositivo
        if myFilter in dump[indice_appoggio:indice]:

            major = p.join(dump[indice:(indice+8)].split("\n"))
            major = int(p.join(major.split()),16)

            minor = dump[(indice+9):(indice+14)]
            minor = int(p.join(minor.split()),16)

            txpowerTMP = dump[(indice+17):(indice+20)]
            txpowerTMP = p.join(txpowerTMP.split())
```

```
            binary_string = bin(int(txpowerTMP, 16))[2:].fill(8)
            txpower = twos_comp(int(binary_string,2), len(binary_string))

            indice_appoggio = indice + 17
            self.lista_beacon_trovati.append({'uuid':myFilter,
                                              'major':major,
                                              'minor':minor,
                                              'txpower':txpower,
                                              'mFound':False})
```

Codice 4.13: funzione ricercaDispositivi in modulo RicercaDispositivi

Come si può notare da *Codice 4.13*, viene adoperata la funzione *scan()*. Questa permette di far uso della libreria subprocess per dare comandi da shell, in particolare per effettuare uno scan dei segnali presenti nella zona.

```
def scan():
    " Ricerca i segnali di advertising "
    command = "timeout -s SIGINT 1s hcitool -i hci0 lescan"
    command1 = "timeout -s SIGINT 1s hcidump --raw"
    args = shlex.split(command)
    p1 = subprocess.Popen(args, stdout=subprocess.PIPE,stderr=subprocess.PIPE)
    args1 = shlex.split(command1)
    p2 = subprocess.Popen(args1, stdout=subprocess.PIPE,stderr=subprocess.PIPE)
    testo, errore=p2.communicate()
    return testo,errore
```

Codice 4.14: funzione scan in modulo RicercaDispositivi

Un esempio di output del comando *hcitool –i hci0 lescan* è il seguente:

Per un beacon con UUID E2C56DB5-DFFB-48D2-B060-D0F5A71096E0, major 0, minor 0, e potenza Tx Power di -59 RSSI, otterremo un pacchetto di segnale di advertising di questo tipo:

```
d6 be 89 8e 40 24 05 a2 17 6e 3d 71 02 01 1a 1a ff 4c 00 02 15 e2 c5 6d
b5 df fb 48 d2 b0 60 d0 f5 a7 10 96 e0 00 00 00 00 c5 52 ab 8d 38 a5
```

hcitool lescan restituirà il messaggio:

```
02 01 1a 1a ff 4c 00 02 15 e2 c5 6d b5 df fb 48 d2 b0 60 d0 f5 a7 10 96
e0 00 00 00 00 c5
```

analizzabile come dimostrato in sezione *1.2.2 PDU advertising* .

Quindi la seconda parte di *Codice 4.13* per *RicercaDispositivi* si occupa principalmente di estrarre i valori e convertirli in una forma *human-readable*, da esadecimale a decimale.
Altre funzioni meno importanti sono state adoperate da RicercaDispositivi come gestioneErrori, formattaUUID e twos_comp.

4.1.9 Modulo AggiornamentoLista

Per l'aggiornamento dei valori dei beacon già presenti nella zona coperta da antenna e per l'inserimento di quelli nuovi il modulo *AggiornamentoLista* ha a disposizione due funzioni distinte.

- *addNewBeacons*

Facendo uso della lista dei beacon trovati, creata in *ricercaDispositivi* (vedi *Codice 4.13*), si individuano i dispositivi "nuovi" ovvero quelli che mantengono il campo *mFound False* , e automaticamente viene aggiunta un'istanza della classe Beacon, con relativi campi compilati, alla lista dei beacon presenti.

```
def addNewBeacons(self):
  " Aggiunge i nuovi beacon dentro la lista, quindi la restituisce "
  for elem in self.lista_beacon_trovati:
     if elem['mFound'] == False:
          self.lista_beacon.append(Beacon(elem['major'],
                                          elem['minor'],
                                          self.currentTime,
                                          self.uuid,
                                          self.nomeAntenna,
                                          self.cod_zona,
                                          self.cod_tenid,
                                          elem['txpower'])
                                        )
          # Notifichiamo il server
          inviaServer(self.lista_beacon[-1],'E')
```

Codice 4.15: funzione addNewBeacons in modulo AggiornamentoLista

Come ultima azione viene infomato il server dell'avvenuto con *inviaServer*.

- *updatePropBeacons*

Facendo uso della lista dei beacon trovati, creata in *ricercaDispositivi* (vedi *Codice 4.13*), si guarda quali sono già presenti nella zona, verificandone major e minor. Quindi se ne modificano i campi time e la lista delle sue ultime 5 potenze rilevate. Sfruttando i metodi implementati nella classe Beacon, viene aggiornata sia la lista delle potenze in averagePower che la media di esse. Inoltre se la potenza rilevata ha valore molto distaccato rispetto alla media, viene avvisato il server per il movimento del beacon.

```
def updatePropBeacons(self):
  " Funzione per l'aggiornamento dei valori nella lista dei beacon trovati"

  media = 0

          # Controllo per ogni beacon trovato major e minor.
          # Dopodichè se presente anche nella lista dei beacon "già presenti/trovati",
          # ne aggiorno currentTime e media potenza tx.
  for elem in self.lista_beacon_trovati:
     for C in range(len(self.lista_beacon)):
        if (elem['major'] == self.lista_beacon[C].major) and (elem['minor'] ==
self.lista_beacon[C].minor):

          elem['mFound'] = True
          self.lista_beacon[C].time = self.currentTime

            # Elimino l'ultimo valore trovato(se necessario) e aggiungo quello nuovo
          if not len(self.lista_beacon[C].averagePower) < 5:
              self.lista_beacon[C].delAveragePower()

          # chiamo la funzione di StrutturaBeacon
          self.lista_beacon[C].setAveragePower(elem['txpower'])

          # Reperisco la media attuale
          media = self.lista_beacon[C].get_media()

          # la salvo nella classe Beacon
          self.lista_beacon[C].log_signal = media

          # Se il nuovo valore e' molto diverso della media lo notifico al server
          # Viene infatti visto come un brusco o rilevante movimento
          if abs( self.lista_beacon[C].log_signal - abs(elem['txpower']) ) >= 20 :
            inviaServer(self.lista_beacon[C],'P')
```

Codice 4.16: funzione updatePropBeacons in modulo AggiornamentoLista

4.1.10 Modulo EliminaBeacon

Dalla lista di beacon presenti, vengono filtrati quelli che da più di 10 secondi non inviano nessun segnale, considerati fuori zona. Viene poi avvisato il server dell'uscita.

```
TTL = 10

def eliminaBeacons(self):
          "Funzione per controllo beacon da eliminare"

          lung = len(self.lista_beacon)
          count = 0
          for element in range(lung):
                    valore = element - count
                    # Se il tempo corrente - tempo da vivo > 10 elimina
                    if (int(round(self.currentTime)) -
int(round(self.lista_beacon[valore].time))) > TTL:

                       inviaServer(self.lista_beacon[valore], 'U')

                       # Elimino il beacon dalla lista di beacon da memorizzare
                       del self.lista_beacon[valore]
                       count += 1
                    # Quando ho analizzato tutti i beacon, stop.
                    if valore == lung:
                      break
```

Codice 4.17: funzione eliminaBeacons in modulo EliminaBeacon

4.1.11 Modulo InvioServer

Il metodo *sendServer* è stato implementato per inviare notifiche al server in caso di ingresso/uscita di un beacon o suo spostamento. Viene fatto uso del formato json per l'invio dei messaggi. Oltre ai campi dei beacon, viene spedita anche la *cViolation*, ovvero il tipo di evento che è avvenuto: "E" se il device è entrato nella zona, "P" se era già presente ed "U" se è uscito (timeout o soglia minima raggiunta).

```
def sendServer(beacon, cViolation):
   "Funzione per invio spostamenti beacon al server"

   url = 'http://lapserv.herokuapp.com/ac/eventi'


   payload = json.dumps(
            {
               "beac_uuid": str(beacon.UUID),
               "beac_major": beacon.major,
               "beac_minor": beacon.minor,
               "ten_id": beacon.cod_tenid,
               "zona_code": str(beacon.cod_zona),
               "viol_mon_type": cViolation,
               "log_signal": float(beacon.log_signal),
               "battery": float(beacon.battery)
            }
         )
   headers = {'mime-type': 'application/json', 'Accept-Charset': 'UTF-8'}
   r = requests.post(url, data=payload, headers=headers)
```

Codice 4.18: funzione sendServer in modulo InvioServer

4.1.12 Modulo Update

Per aggiornare il dispositivo, viene scaricato da server un file di pochi byte contenente il numero di versione e una lista dei moduli da aggiornare. Quindi viene confrontata la versione attuale, contenuta in un file *readme.txt*, con quella fornita dal server: se sono uguali, il file scaricato viene eliminato e l'antenna prosegue la ricerca dei segnali. In caso contrario viene utilizzato il metodo *applicazioneAggiornamenti*() che effettua una sostituzione dei moduli con quelli più recenti ed esegue un reboot del sistema.

```
def ricercaAggiornamenti():
    " Funzione che controlla se sono necessari aggiornamenti di versione "

    # Controllo la versione del file
    nomeFile = "tmp/README"
    versione_vecchia = riceviVersione(nomeFile)

    url = 'http://www.lapsy.me/img/beacons/AggRaspberry/README'
    locale = url.split('/')[-1]

    r = requests.get(url, stream = True)
    with open(locale, 'wb') as f:
        for chunk in r.iter_content(chunk_size=512):
            if chunk:
                f.write(chunk)

    versione_nuova = riceviVersione(locale)

    # se le versioni sono diverse, applico aggiornamenti
    if versione_vecchia == versione_nuova:
        print "Software già aggiornato alla versione più recente: " + versione_nuova
        os.remove("README")
        return "current version"
    else:
        print "Da aggiornare"
        os.remove("tmp/README")
        os.rename("README", "tmp/README")
        # applico gli aggiornamenti
        applicazioneAggiornamenti('README')
        return versione_nuova
```

Codice 4.19: funzione ricercaAggiornamenti in modulo Update

Per conoscere la versione corrente, viene fatto uso della funzione *riceviVersione* con parametro di ingresso il percorso del file in cui poterla trovare.

```
def riceviVersione(nomeFile):
    " Funzione che estrapola la nuova/vecchia versione da applicare"
    f = open(nomeFile,"r")
    testo = f.read()
    f.close()
    # prendo la versione contenuta nel README
    version = []
    for indice in range(len(testo)):
        if 'version' in testo[:indice]:
            version = re.split("[, \s = \t \n]+",testo[indice:])
            print version[1]
            break
    return version[1]
```

Codice 4.20: funzione riceviVersione in modulo Update

Infine *applicazioneAggiornamenti* prende in una lista i nomi dei moduli da aggiornare, quindi li scarica da server e li sostituisce con quelli presenti nel progetto. Per la ricerca dei file nel progetto è stato fatto uso di *os.walk()* che passatogli il percorso in cui ricercare i moduli restituisce una tupla di 3 elementi, il primo rappresentante il percorso in cui ci si trova (*path*), il secondo una lista di directory all'interno del path ed il terzo una lista dei file presenti nel path. Quando nella lista dei file è presente un modulo con nome uguale a quello aggiornato, viene quindi sostituito.

```
def applicazioneAggiornamenti(nomeF):
    " Funzione che sostituisce moduli con quelli di ultima versione "

    f = open(nomeF, "r")
    righe = f.readlines()
    f.close

    if len(righe) <= 1:
        del righe[0]

    # cerco i file da aggiornare i cui nomi sono contenuti dentro al file README
    # una volta trovati li scarico

    urls = 'http://www.lapsy.me/img/beacons/AggRaspberry/'
    for fileM in righe:
        scarica_url = urls + fileM
        r = requests.get(scarica_url, stream = True)
        with open(fileM, 'wb') as f:
            for chunk in r.iter_content(chunk_size=512):
                if chunk:
                    f.write(chunk)

    # Per ogni cartella presente, cerca se c'è un file nominato
    # uguale a quello da sostituire, quindi lo sostituisco
        for tripla in os.walk(os.getcwd()):
            lista_file = tripla [2]
            for elem in lista_file:
                if elem == fileM:
                    tmppercorso = tripla [0] + '/' + elem
                    nuovopercorso = tmppercorso[1:]
                    os.remove(nuovopercorso)
                    os.rename(fileM,nuovopercorso)
```

Codice 4.21: funzione applicazioneAggiornamenti in modulo Update

4.2 myscript.service

Per rendere indipendente il sistema e poter risolvere casi di blackout, è stato implementato il riavvio automatico del sistema. In particolare è stato fatto uso di un servizio che, al termine del caricamento del kernel Linux e del desktop, potesse avviare l'applicazione (si veda *[SysU]*).

E' stato quindi creato uno script di servizio in */lib/systemd/system/*:

```
root@raspberrypi:/home/pi/ProgettoLapsy# sudo nano /lib/systemd/system/myscript
service
root@raspberrypi:/home/pi/ProgettoLapsy# chmod 644 /lib/systemd/system/myscript
service
root@raspberrypi:/home/pi/ProgettoLapsy#
```

Figura 4.3: creazione file .service

Abbiamo configurato la sezione *Unit* con una parte di descrizione e la sezione obbligatoria *Service* assegnando al campo *Type* il valore idle. Idle permette di eseguire il servizio quando tutti compiti di inizializzazione della macchina debian sono conclusi, anche per quanto riguarda l'interfaccia grafica. Quest'ultima è necessaria per poter visualizzare, alla prima accensione del dispositivo, l'interfaccia da cui settare i valori dell'antenna.
ExecStart esegue all'avvio del servizio il comando che gli viene impostato, nel nostro caso il percorso completo del comando python e del modulo main. Inoltre tutti gli eventuali errori vengono salvati su un file di log e reindirizzati sul stdout tramite il comando 2>&1 (stderr reindirizzato sul file descriptor 1 ovvero stdout).

```
pi@raspberrypi: ~/ProgettoLapsy
File  Edit  Tabs  Help
  GNU nano 2.2.6              File: /lib/systemd/system/myscript.service

[Unit]
Description= Script Lapsy
After = multi-user.target

[Service]
Type = idle
ExecStart = /usr/bin/python/ /home/pi/ProgettoLapsy/mainLapsy.py > /home/pi/lapsy.log 2>&1

[Install]
WantedBy = multi-user.target

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Figura 4.4: creazione sezioni in file myscript.service

Come ultimo passaggio è stato quindi abilitato il servizio ed effetuato un primo reboot del sistema.

```
root@raspberrypi:/home/pi/ProgettoLapsy# sudo systemctl daemon-reload
root@raspberrypi:/home/pi/ProgettoLapsy# sudo systemctl enable myscript.service
Created symlink from /etc/systemd/system/multi-user.target.wants/myscript.service to /lib/systemd/system/mys
cript.service.
root@raspberrypi:/home/pi/ProgettoLapsy# reboot
```

Figura 4.5: attivazione servizio /lib/systemd/system/myscript.service

Capitolo 5

Test del sistema

In questo capitolo vengono messi in luce i test avvenuti per verificare il corretto funzionamento dell'antenna. Inoltre sono state studiate le potenze dei segnali emessi dai beacon per comprendere il miglior rapporto potenza/frequenza da adottare per i dispositivi.

Lo scopo principale di questa fase è stata trovare un buon compromesso tra frequenza dei segnali emessi e loro potenza. E' intuibile pensare a un comportamento ottimo da parte dei beacon nel caso queste due caratteristiche siano massime: una elevata potenza permette un grande range di azione da parte dell'antenna per l'individuazione dei dispositivi, una alta frequenza invece maggior quantità di segnali al minuto ricevibili, ovvero maggior velocità di aggiornamento delle posizioni dei beacon da parte dell'antenna.
La potenza massima impostabile per un beacon è di +4dbm, limitata a solo 0dBm nella fase di test, quella minima è di -40dbm; La frequenza assunta nel test è stata impostata come valore massimo per invio segnale ad 0.4s e minimo per un segnale ogni 5s.

Per una più facile comprensione viene proposta una tabella di conversione dbm/W:

Potenza (dBm)	Potenza (Watt)
-40 dBm	0.0000001 W
-30 dBm	0.000001 W
-20 dBm	0.00001 W
-10 dBm	0.0001 W
0 dBm	0.001 W
1 dBm	0.0012589 W
2 dBm	0.0015849 W
3 dBm	0.0019953 W
4 dBm	0.0025119 W

Conversione Potenza dBm → Watt

Come si può notare, ciò che ci ha portato ad assumere come valore massimo di potenza impostabile per un beacon a 0dBm, nel caso di questo progetto, è stata la forte diminuzione della potenza in Watt necessaria al dispositivo, che si tramuta in una forte diminuzione dell'uso della sua batteria, rispetto alla sua potenza massima di +4dBm, quasi 2 volte e mezzo superiore. Questa scelta è stata adottata una volta effettuati i test, che si sono rivelati più che sufficienti per la potenza impostata a 0dBm, la quale permette all'antenna di poter vedere il dispositivo fino a una distanza di 12 metri.

La potenza impostata nella fase di test assume i valori di 0dBm, -5dBm, -10dBm, -20dBm e -30dBm.
La frequenza impostata nella fase di test assume i valori di 0.4s, 1s, 2.5s e 5s .

		Potenza Massima (0 dBm)			
	Frequenza	0.4s	1s	2.5s	5s
Distanza	14m	-	-	-85/95	-
	12m	-85/91 (max 99)	-87/95	-85/95	-
	10m	-74/88	-77/88/90	-80/90	-73/77
	8m	-75/80	-75/90	-70/80-83	-75/85
	6m	-70/80	-68/78	-70/80	-68/80
	4m	-62/75	-58/77	-60/74	-70/82
	3m	-60/80	-57/75	-60/75	-55/70
	2m	-61/74	-55/75	-59/70	-55/65
	1m	-55/72 (max 77)	-60/75	-54/70	-50/65
	50cm	-56	-53/56	-50	-48/50
	30cm	-37/39	-37/39	-35/37	-34/37

Tabella 5.1: potenze rilevate da antenna data potenza massima (0dBm)

Dalla *Tabella 5.1* si può notare come l'antenna a una distanza di 12 metri dai dispositivi riesca, con valori limite (si ricorda che -100 RSSI rappresenta la distanza massima "visibile"), a riceverne i segnali. Questa potenza si rivela quindi più che sufficiente per il nostro progetto, limitato ad un uso in camere e stanze. Vengono quindi effettuati altri test con impostate potenze inferiori sui beacon.

		Potenza Alta (-5 dBm)			
	Frequenza	0.4s	1s	2.5s	5s
Distanza	14m	-	-	-	-
	12m	-	-	-	-
	10m	-85/95	-80/90	-87/96	-
	8m	-75/85	-78/90	-80/92	-76/85
	6m	-80/90	-77/90	-75/86	-70/85
	4m	-71/81	-75/88	-78/87	-73/83
	3m	-69/79	-70/80	-70/80	-68/78
	2m	-62/72	-60/73	-60/75	-58/70
	1m	-56/70 (max 80)	-66/79	-55/70	-60/70
	50cm	-56/58	-53/56	-52/53	-48/50
	30cm	-38/40	-38/39	-36/39	-35/38

Tabella 5.2: potenze rilevate da antenna data potenza alta (-5 dBm)

Da *Tabella 5.2* i valori di RSSI sono aumentati su tutti i fronti, per tutte le distanze e tutte le frequenze impostate sui beacon. In particolare il range di "visibilità" dell'antenna comincia a farsi più ristretto, supportando una distanza massima di 10 metri.

		Potenza Media (-10 dBm)			
	Frequenza	0.4s	1s	2.5s	5s
Distanza	14m	-	-	-	-
	12m	-	-	-	-
	10m	-	-	-	-
	8m	-82/92	-	-	-
	6m	-74/85	-85/99	-80/90	-76/84
	4m	-77/87	-75/95	-77/80	-74/85
	3m	-72/83	-80/94	-75/84	-73/83
	2m	-67/79	-75/85	-60/62	-68/78
	1m	-55/75	-52/54	-62/75	-55/70
	50cm	-56/60	-57/61	-56/60	-50/53
	30cm	-40/43	-39/42	-37/39	-37/38

Tabella 5.3: potenze rilevate da antenna data potenza media (-10 dBm)

		Potenza Bassa (-20dBm)			
	Frequenza	0.4s	1s	2.5s	5s
Distanza	14m	-	-	-	-
	12m	-	-	-	-
	10m	-	-	-	-
	8m	-	-	-	-
	6m	-80/90	>-95	-	-
	4m	-81/93	-80/90	-	-
	3m	-77/85 (max 90)	-80/95	-	-
	2m	-77/90	-77/90	-73/83	-78/85
	1m	-77/90	-70/80	-70/80	-67/80
	50cm	-59/70	-60/75	-40/75	-57/70
	30cm	-56/70	-57/70	-50/60	-42/44

Tabella 5.4: potenze rilevate da antenna data potenza bassa (-20 dBm)

Per *Tabella 5.3* e *Tabella 5.4* il comportamento è analogo a quello di *Tabella 5.2*. Valori molto alti per massime distanze, quest'ultime comprese tra 6 - 4 metri.

		Potenza Minima (-30 dBm)			
	Frequenza	0.4	1	2.5	5
Distanza	14m	-	-	-	-
	12m	-	-	-	-
	10m	-	-	-	-
	8m	-	-	-	-
	6m	-	-	-	-
	4m	-	-	-	-
	3m	-	-	-	-
	2m	-	-	-82/92	-
	1m	-90/95	-81/97	-75/87	-88/90
	50cm	-84/95	-73/85	-62/74	-67/70
	30cm	-59/73	-61/73	-62	-63/70

Tabella 5.5: potenze rilevate da antenna data potenza minima (-30 dBm)

Infine l'ultima *Tabella 5.5* identifica un possibile utilizzo dei dispositivi e dell'antenna solo per usi a breve distanza, non necessario nel caso del nostro sistema di geolocalizzazione.

In primo luogo dai test apportati è possibile identificare una alta oscillazione, come era stata già accennata in *3.5 Le struttura Antenna e Beacon - pag 37*, i quali confermano come sia necessario l'uso di una media dei valori ottenuti dai dispositivi per comprendere la loro distanza effettiva.

Viene individuata una buona soluzione potenza/frequenza/distanza dei dispositivi iBeacon per la *Tabella 5.2*, e in certi casi anche *Tabella 5.3*.
E' stato quindi scelto per la configurazione dei beacon il valore di -5 dBm (potenza alta), con una frequenza di 4 secondi, mantenendo un uso della batteria limitato.

La frequenza bassa è stata decisa perchè lo scopo del progetto è lontano da quello che può essere definito un sistema di sicurezza, il quale implica alta reattività dell'antenna.

Un risultato del comportamento dell'antenna è stato salvato in *lapsy.log*, il quale viene proposto in forma completa nella sezione *Appendici*, dove viene mantenuta una distanza di 5/6 metri (più che sufficiente per simulare una camera di Hotel) per poi effettuare, dopo circa un minuto (nel minuto *09:10:58*), uno spostamento verso l'antenna, il quale viene notificato al server.

```
[...]
Nov 17 09:09:53 raspberrypi: Sent 56697267-1122-3344-0000-001000000011+256+8, E, -86
Nov 17 09:09:57 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -77
Nov 17 09:10:01 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -83
[...]
Nov 17 09:10:53 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -78
Nov 17 09:10:58 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -51
Nov 17 09:10:58 raspberrypi: Sent 56697267-1122-3344-0000-001000000011+256+8, P, 82
Nov 17 09:11:02 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -65
[...]
Nov 17 09:11:18 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -95
Nov 17 09:11:28 raspberrypi: Sent 56697267-
1122-3344-0000-001000000011+256+8, U, 0
```

Codice 5.1: Comportamento antenna salvato in file lapsy.log

Conclusione e sviluppi futuri

Nell'ambito di questa tesi è stato raggiunto l'obiettivo prefissato.

E' stata completata la realizzazione di un dispositivo che tramite geolocalizzazione bluetooth lavora da antenna per controllare eventi di wearable device intorno ad essa.
Il dispositivo è inoltre capace di lavorare in maniera indipendente, senza l'ausilio di una persona che le specifichi i comandi da eseguire per il suo avvio o manutenzione.

Lo studio di diverse tecnologie da poter applicare ha permesso scelte progettuali ed implementative semplici e nel miglior modo efficienti.

Alcuni sviluppi futuri per il progetto possono comprendere:

- l'uso effettivo del campo batteria nei beacon, con eventuale avviso al server del tempo di vita rimanente del dispositivo;
- gestione di segnali d'allarme per ingressi indesiderati nell'area;
- possibilità di connessione con i wearable device per invio comandi di allarme (vibrazione) o notifiche.

Appendici

File lapsy.log

Nov 17 09:08:50 raspberrypi: Already up-to-date.

Nov 17 09:09:21 raspberrypi: Checking for new version...

Nov 17 09:09:21 raspberrypi: Sent ping

Nov 17 09:09:25 raspberrypi: No updates found.

Nov 17 09:09:53 raspberrypi: Sent 56697267-1122-3344-0000-001000000011+256+8, E, -86

Nov 17 09:09:57 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -77

Nov 17 09:10:01 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -83

Nov 17 09:10:05 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -81

Nov 17 09:10:09 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -78

Nov 17 09:10:14 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -78

Nov 17 09:10:20 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -81

Nov 17 09:10:24 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -87

Nov 17 09:10:29 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -79

Nov 17 09:10:33 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -90

Nov 17 09:10:37 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -80

Nov 17 09:10:41 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -86

Nov 17 09:10:45 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -79

Nov 17 09:10:49 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -87

Nov 17 09:10:53 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -78

Nov 17 09:10:58 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -51

Nov 17 09:10:58 raspberrypi: Sent 56697267-1122-3344-0000-001000000011+256+8, P, 82

Nov 17 09:11:02 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -65

Nov 17 09:11:06 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -75

Nov 17 09:11:10 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -88

Nov 17 09:11:14 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -93

Nov 17 09:11:18 raspberrypi: beacon 56697267-1122-3344-0000-001000000011+256+8 => -95

Nov 17 09:11:28 raspberrypi: Sent 56697267-1122-3344-0000-001000000011+256+8, U, 0

Sitografia

- [A10] A10-OlinuxIno : https://www.olimex.com/wiki/A10-OLinuXino-LIME

- [BLE] Bluetooth Low Energy : http://www.argenox.com/a-ble-advertising-primer/

- [PA] Proximity Analysis : https://pdfs.semanticscholar.org/3a7c/b58dec6c39d31db6c36aec6091e3149baaf6.pdf

- [PW] Python wifi : https://media.readthedocs.org/pdf/wifi/latest/wifi.pdf

- [Rbpi3] RaspBerry pi3 model b : https://www.raspberrypi.org/documentation/

- [SysU] Systemd unit file : https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/sect-Managing_Services_with_systemd-Unit_Files.html