



Università degli Studi di Messina

Dipartimento di Ingegneria

**Corso di Laurea Triennale in Ingegneria Elettronica e
Informatica**

PROGETTAZIONE E IMPLEMENTAZIONE DI UN FRAMEWORK PER SMART ENVIRONMENT BASATO SU TECNOLOGIA BLE

Tesi di Laurea di:
Emanuele **MUNAFÒ**

Relatore:
Chiar.mo Prof. Francesco **LONGO**

Anno Accademico 2017-2018

Un rigraziamento speciale va alla mia famiglia per il loro continuo ed inestimabile affetto e
supporto.

Indice

Elenco delle figure	5
Glossario	7
1 Introduzione	1
2 Background	6
2.1 Bluetooth	6
2.2 Tecnologia BLE	7
2.3 Livello fisico e di collegamento in Bluetooth	8
2.4 Advertisement Bluetooth	11
2.5 Beacon	14
2.6 HCI UART	15
2.7 Raspberry	20
2.8 Crittografia	23
3 Smart Environment	26
3.1 Descrizione	26
3.2 Descrizione dei requisiti	28
3.3 Progettazione e implementazione	30
3.3.1 Basi di dati	31
3.4 Funzionamento	33
4 Il protocollo CtrlBeacon	36
4.1 Introduzione	36
4.2 Descrizione dei pacchetti	36
4.2.1 HelloBroadcast	38
4.2.2 Wifi Password	39
4.2.3 Command	40

4.2.4 ACK	42
5 Interazione tra utenti e smart core	43
5.1 Configurazione	44
5.1.1 Analisi delle dipendenze e dei requisiti	44
5.1.2 Scelte per la configurazione	45
5.2 Descrizione del software	46
5.2.1 Modulo beacon	47
5.2.2 Modulo smartObject	47
5.2.3 Modulo smartCore	48
5.2.4 Moduli ausiliari	49
5.3 Sequenza delle operazioni	50
5.3.1 Attesa e risposta	50
5.3.2 Autenticazione	55
5.3.3 Aggiornamento	55
5.3.4 Termine della connessione	56
6 Interazione tra utenti e smart object	57
6.1 Configurazione	57
6.1.1 Analisi delle dipendenze e dei requisiti	57
6.1.2 Scelte per la configurazione	58
6.2 Descrizione del software	58
6.2.1 Modulo smart object	58
6.3 Sequenza delle operazioni	61
6.3.1 Attesa ed esecuzione	62
7 Conclusione	66
7.1 Analisi dei requisiti e degli obiettivi raggiunti	66
7.2 Possibili miglioramenti ed estensioni future	68
7.3 Vantaggi	68
7.4 Limiti	69
7.5 Confronto con le tecnologie esistenti	69
Bibliografia	70
Appendice A File di configurazione	71
Appendice B Generazione e installazione del file di immagine	74

Elenco delle figure

1.1	Rappresentazione schematica dello smart environment	2
1.2	Rappresentazione di un ambiente intelligente	4
2.1	Rappresentazione di una struttura a livelli semplificata di BLE	8
2.2	Mappatura delle frequenze utilizzate dalla tecnologia Bluetooth	9
2.3	Formato di un pacchetto BLE (uncoded PHY)	10
2.4	Formato del PDU di Advertising	11
2.5	Formato dell'header del PDU di Advertising	11
2.6	Pacchetto BLE di legacy advertising	12
2.7	Payload PDU di advertising	12
2.8	Indirizzi BLE	13
2.9	Pacchetto AltBeacon	14
2.10	Disposizione dei campi di un AltBeacon	15
2.11	Livello di trasporto HCI UART	15
2.12	Livello di trasporto HCI UART (2)	16
2.13	Tipi di pacchetti HCI	16
2.14	HCI: LE Set Advertising Parameters	17
2.15	HCI: LE Set Advertising Data	18
2.16	HCI: Set Advertising Data	19
2.17	HCI: Set Scan Parameters	19
2.18	Raspberry Pi 3B	21
2.19	Raspberry Pi 0	22
2.20	Algoritmo di cifratura AES	24
2.21	Funzionamento AES CBC	25
3.1	Esempio di uno smart environment	27
3.2	Esempio di interazione con oggetti intelligenti	30
3.3	Schema ER base di dati	32

3.4	Diagramma dei casi d'uso	34
4.1	Rappresentazione di un AltBeacon	37
4.2	Rappresentazione del pacchetto HelloBroadcast	38
4.3	Rappresentazione del pacchetto WifiPassword	39
4.4	Rappresentazione del pacchetto generico cifrato usato nel protocollo Ctrl-Beacon	40
4.5	Rappresentazione del pacchetto generico da cifrare usato nel protocollo CtrlBeacon	42
5.1	Diagramma dell'interazione tra utente e smart core	43
5.3	Composizione del modulo smartObject	48
5.4	Rappresentazione del modulo smartCore	49
5.5	Moduli ausiliari	49
5.6	Sequence diagram relativo allo smart core	51
6.1	Composizione del modulo smartObject	59
6.2	Sequence diagram relativo allo smart object	62
6.3	Macchina a stati finiti di un ricevitore	64
6.4	Macchina a stati finiti di un trasmettitore	65

Glossario

Acronimi / Abbreviazioni

- ACK* Acknowledgement: nell'ambito delle telecomunicazioni rappresenta il meccanismo di conferma ricezione
- AES* Advanced Encryption Standard
- PDU* Automatic Gain Control
- AP* Access Point
- BLE* Bluetooth Low Energy
- BR* Basic Rate
- CBC* Cipher Block Chaining
- CSI* Camera Serial Interface
- DHCP* Dynamic Host Configuration Protocol
- DSI* Display Serial Interface
- EDR* Enhanced Data Rate
- ER* Entità Relazione
- FSM* Finite State Machine
- GAP* Generic Access Protocol
- GUI* Graphic Unit Interface
- HCI* Host Controller Interface
- HDMI* High Definition Multimedia Interface

HTTP Hyper Text Transfer Protocol

ISM Industrial Scientific Medical

LE Low Energy

MAC Message Authentication Code

OGF Opcode Command Field

OGF Opcode Group Field

PAN Personal Area Network

WPAN Wireless Personal Area Network

PDU Protocol Data Unit

PKCS Public-Key Cryptography Standards

RF Radiofrequenza

SQL Structured Query Language

UART Universal Asynchronous Receiver and Transmitter

Capitolo 1

Introduzione

L'idea alla base di tale elaborato si fonda sull'interesse, oggi sempre crescente, nei confronti di tecnologie che consentano una maggiore interazione con gli oggetti e gli spazi che ci circondano. In tal senso i luoghi contenenti degli oggetti che acquisiscono la capacità di interazione verranno definiti *smart environment*. Oggi esistono varie alternative che si propongono come soluzione a tale problema delle quali le più note sono Amazon Echo e Google Home, sistemi Cloud-based che consentono di rendere gli ambienti interattivi. Tuttavia tali soluzioni presentano vari limiti e difetti. Per questi motivi si è ritenuto interessante proporre un'alternativa, sviluppando un sistema che abbia le stesse funzioni ma che miri a ottenere dei vantaggi rispetto alle alternative presenti. Questo elaborato vuole descrivere in modo esaustivo sia il lavoro svolto, sia le caratteristiche, il funzionamento e l'analisi del prodotto ottenuto. Sono anche stati definiti alcuni vincoli chiave (necessità) da rispettare e su cui porre una particolare attenzione.

L'intero sistema deve:

1. Essere realizzato utilizzando un protocollo che non necessiti dell'accesso online alla rete.
2. Poter essere considerato, in tutte le sue parti, sufficientemente sicuro.
3. Garantire delle prestazioni comparabili o superiori alle alternative già esistenti.
4. Avere costi di realizzazione ridotti
5. Essere sufficientemente flessibile per poterne permettere l'espansione e l'ottimizzazione.
6. Consentire compatibilità con quanti più dispositivi, possibilmente utilizzando tecnologie a basso consumo.

Tali requisiti hanno determinato la necessità di progettare un nuovo protocollo che facesse uso della tecnologia nota come beacon Bluetooth. Il nuovo protocollo sarà definito sopra uno già esistente chiamato AltBeacon, standard assai diffuso che garantisce un alto livello di compatibilità. Sono state a tal punto definite alcune funzionalità considerate utili ai fini dell'applicazione di interesse.

Il protocollo deve:

1. Prevedere l'interazione di più utenti con un solo oggetto.
2. Contenere dei meccanismi di autenticazione e gestione dei permessi.
3. Avere prestazioni accettabili e garantire la ricezione pur considerando il canale inaffidabile ovvero che non garantisce l'integrità e la ricezione dei dati.
4. Poter essere implementato facilmente su diversi dispositivi.
5. Poter essere esteso da terze parti allo scopo di implementare nuove funzionalità.
6. Offrire meccanismi di sicurezza contro vari vettori di attacco.

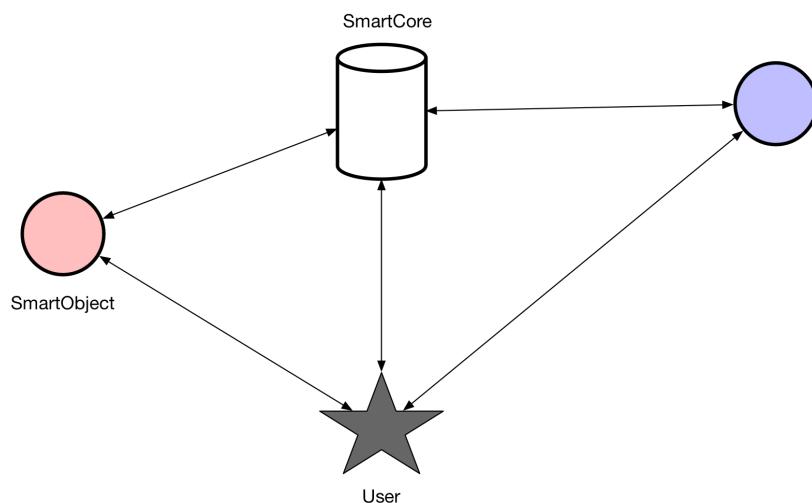


Figura 1.1 Rappresentazione delle entità coinvolte in uno smart environment

In particolare è possibile identificare all'interno dello *smart environment* differenti tipologie di entità.

Smart Core Lo *smart core*, rappresentato dal cilindro nelle figure, è il dispositivo che viene utilizzato tanto per la gestione dei dati e della base di dati quanto per l'implementazione dei meccanismi di sicurezza e autenticazione.

Smart Object Lo *smart object*, rappresentato dal cerchio nelle figure, rappresenta il dispositivo che, collegato ad attuatori e sensori, consente di ricevere comandi ed inviare messaggi all’utente.

Utente Utente viene considerata una persona dotata di smartphone o dispositivo compatibile che, trovandosi all’interno dello *smart environment* e in possesso dell’autorizzazione o della fiducia necessaria, vuole interagire con esso. Il funzionamento schematizzato in figura può essere pensato come suddiviso in varie procedure che, nel loro complesso e soddisfacendo tutte le proprietà e i requisiti analizzati finora, rendono efficace il sistema. Tali procedure sono:

- La registrazione di un utente e l’ottenimento di una chiave privata generata casualmente e che dovrà essere custodita dall’utente.
- Lo scambio di messaggi tra utente e *smart object*, cifrato e con conferma di ricezione, che consenta l’esecuzione di azioni tramite attuatori o l’ottenimento di informazioni di vario tipo.

Oltre la progettazione si è realizzata una possibile implementazione del protocollo in linguaggio Python che consiste di più livelli: un modulo utilizzato per l’esecuzione di comandi e il parsing degli eventi a basso livello e degli altri definiti per il funzionamento degli oggetti appartenenti allo *smart environment* (*smart core* e *smart object*). Il software è stato programmato ed ottimizzato per una soluzione hardware, a basso costo e assai diffusa, che prende il nome di Raspberry. In particolare si è pensato di utilizzare differenti modelli per una ulteriore riduzione dei costi, delle dimensioni e dei consumi, pur mantenendo ottime prestazioni. La progettazione è stata inoltre accompagnata da una fase di test e verifica del prodotto finale che ha consentito di analizzarne i vantaggi e le possibili limitazioni.

In tal senso, il prodotto finale viene definito framework di cui viene fornita la definizione che si trova nel Cambridge Dictionary.

Framework: a supporting structure around which something can be built[8].

La strategia seguita che ha portato alla realizzazione di ambienti intelligenti può essere immaginata come costituita da varie fasi[1]:

- L’analisi dei requisiti
- La progettazione
- L’implementazione

- La validazione e il collaudo

La soluzione proposta prevede infatti la realizzazione di Smart Object ovvero di oggetti che consentano di creare un'interazione tra gli oggetti, l'ambiente circostante e l'utente. Le possibili applicazioni spaziano su vari ambiti e contesti differenti, dagli ambienti interni a quelli esterni e ancora dalla macchina del caffè al frigorifero.



Figura 1.2 Rappresentazione di un ambiente intelligente

Infine si vuole illustrare brevemente il contenuto dei capitoli seguenti e la relazione che intercorre tra di essi. Nel Capitolo 2 verranno richiamati alcuni concetti necessari alla comprensione dell'elaborato. Nel Capitolo 3 si avrà una descrizione degli ambienti

intelligenti e verrà illustrato sia il funzionamento sia le fasi necessarie per la progettazione e l'implementazione del prodotto finale. Nel Capitolo 4 verrà descritto il protocollo necessario al suo funzionamento. Nei Capitoli 5 e 6 saranno analizzate più nel dettaglio le due principali interazioni che possono presentarsi nel sistema. Infine il settimo ed ultimo capitolo conterrà le conclusioni ovvero l'analisi dei risultati ottenuti dai vari punti di vista.

Capitolo 2

Background

2.1 Bluetooth

Bluetooth wireless technology is a short-range communications system intended to replace the cable(s) connecting portable and/or fixed electronic devices. The key features of Bluetooth wireless technology are robustness, low power consumption, and low cost. Many features of the core specification are optional, allowing product differentiation.[10]

La tecnologia Bluetooth è un sistema di comunicazione appartenente alle reti WPAN e quindi a portata ridotta e senza fili. I principali vantaggi e obiettivi di tali tecnologie, definite nello standard IEEE 802.15, sono il basso consumo e il basso costo.

Il protocollo Bluetooth prevede due differenti modalità di funzionamento: Basic Rate (BR) e Low Energy (LE). La modalità BR offre connessione sincrone e asincrone con bitrate di 721.2 kbps, 2.1Mbps se in modalità EDR e fino a 54 Mbps in applicazioni che fanno uso dello standard 802.11 AMP. La modalità LE offre una bitrate e un duty cycle decisamente inferiore ma a fronte di consumi e costi notevolmente ridotti. Entrambe le modalità possiedono meccanismi di scansione e connessione ed inoltre fanno uso della banda senza licenza dei 2.4 GHz (ISM).

Lo stack Bluetooth è costituito da un host e uno o più controller. L'host viene definito come un'entità logica che comprende tutti i livelli sopra l'HCI e sotto i profili non-core. Un controller viene definito come l'entità logica costituita da tutti i livelli sottostanti l'HCI.[10] In altri termini il controller contiene l'interfaccia radio, solitamente un dispositivo integrato a basso costo contenente un microprocessore e il modulo radio Bluetooth. L'host invece si occupa di gestire i dati e gli eventi ad un più alto livello ed in tal senso viene spesso implementato come parte di un sistema operativo o come un software da eseguire su di esso.

In alcuni casi host e controller possono essere presenti entrambi come implementazione diretta sul microcontrollore; in tal caso il sistema viene definito hostless.

2.2 Tecnologia BLE

Il principale vantaggio e obiettivo delle tecnologie BLE è rappresentato da un consumo notevolmente ridotto che ha permesso la diffusione di numerosi dispositivi indossabili quali, ad esempio, smartwatch e fitness-tracker. Grazie ad un consumo ridotto, un dispositivo BLE può essere alimentato da una singola batteria per mesi o addirittura anni. Il consumo tanto basso è dovuto alla modalità di funzionamento di BLE che mantiene il modulo radio spento per la maggior parte del tempo. In aggiunta BLE ascolta solo su tre canali e la radio si riattiva solo per inviare o ricevere brevi sequenze di dati, con pacchetti di piccolo formato. Inoltre BLE configura le connessioni molto rapidamente, fatto che minimizza ulteriormente il periodo di accensione del modulo radio. L'introduzione della tecnologia del Bluetooth Low Energy ha inoltre abbassato la latenza rispetto a Bluetooth Classic. BLE presenta anche altre differenze, infatti tale tecnologia utilizza una topologia di rete a stella e un indirizzo di accesso a 32 bit che, teoricamente, consente a miliardi di dispositivi di essere connessi contemporaneamente. La topologia di rete utilizzata da Bluetooth Classic (piconet) era limitata a otto dispositivi connessi simultaneamente o fino a 255 dispositivi in modalità parcheggio.

Il protocollo BLE fa uso della trasmissione di advertisement, brevi messaggi che possono essere inviati senza necessità di instaurare prima una connessione. Un dispositivo BLE può operare in quattro differenti modalità di funzionamento da cui dipende il comportamento degli stessi dispositivi. Due modalità vengono utilizzate ai fini di instaurare una connessione bidirezionale e sono rappresentati dai seguenti casi:

- Un dispositivo, definito periferico, assume il ruolo di trasmettitore (*advertiser*) in modo tale da poter essere rilevato da un altro dispositivo e instaurare una connessione. Due esempi potrebbero essere un sensore di temperatura e uno speaker wireless.
- Un dispositivo, definito centrale, scansiona per rilevare pacchetti di *advertising* e inizializza una connessione, svolgendo quindi il ruolo di master in una o più connessioni. Esempi di tali dispositivi potrebbero essere gli smartphone e i computer.

Le altre due modalità sono utilizzate per instaurare un canale di comunicazione monodirezionale e possono essere descritte dai seguenti casi:

- Un dispositivo, definito emittente o più comunemente *broadcaster*, assume il ruolo di trasmettitore al fine di inviare informazioni nei dintorni che si estendono sino alla sua

portata. Un esempio potrebbe essere un tag elettronico che segnala costantemente la sua presenza.

- Un dispositivo, definito osservatore, scansiona per rilevare pacchetti di *advertising* non al fine di stabilire una connessione, quanto piuttosto per ricevere le informazioni contenute all'interno. Un esempio potrebbe essere un dispositivo che monitora la presenza di un tag elettronico.

Allo scopo di analizzare vantaggi e svantaggi della tecnologia Bluetooth, giunta oggi alla versione 5.0, si vuole richiamare qualche concetto relativo al suo funzionamento sui vari livelli.

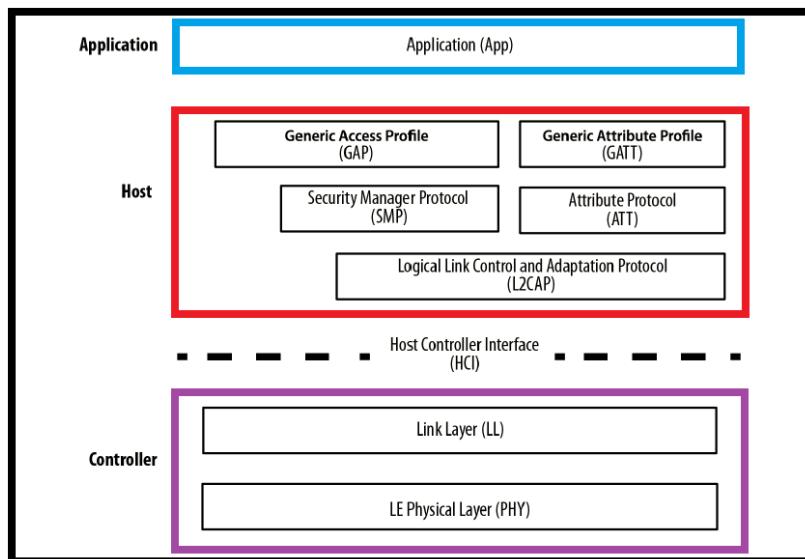


Figura 2.1 Rappresentazione di una struttura a livelli semplificata di BLE

2.3 Livello fisico e di collegamento in Bluetooth

Il canale fisico di comunicazione è costituito da 40 Canali RF nell'intorno della banda ISM a 2.4 Ghz. In particolare tali frequenze sono condivise per diversi scopi, ad ognuno del quale può essere associato un canale, come l'invio di pacchetti dati, di pacchetti di *advertising* e infine per l'invio di pacchetti periodici (2.2). In particolare il canale di *advertising* usa tutti i 40 canali RF, di cui 3 sono definiti canali di *advertising* primari e i restanti 37 sono definiti come secondari. I primi sono utilizzati per tutte le operazioni di *advertising* retrocompatibili (BT 4.2 e versioni precedenti) e per i messaggi di *advertising* iniziali, i secondi sono utilizzati dal protocollo durante la comunicazione. Due dispositivi che vogliono comunicare dovranno

utilizzare un canale fisico condiviso, in altri termini dovranno sintonizzarsi sullo stesso canale RF. Per evitare che più dispositivi, inclusi in uno spazio limitato, possano casualmente utilizzare gli stessi canali nello stesso istante, si fa uso di un indirizzo di accesso che contiene al suo interno informazioni circa il canale utilizzato. L'indirizzo di accesso (Access Address) è una proprietà del canale fisico e viene trasmesso all'inizio di ogni pacchetto inviato. Bisogna inoltre notare che il livello di collegamento fa uso di un solo canale per istante.

RF Channel	RF Center Frequency	Channel Index	Channel Type		
			Data	Primary Advertising	Secondary Advertising
0	2402 MHz	37		•	
1	2404 MHz	0	•		•
2	2406 MHz	1	•		•
...
11	2424 MHz	10	•		•
12	2426 MHz	38		•	
13	2428 MHz	11	•		•
14	2430 MHz	12	•		•
...
38	2478 MHz	36	•		•
39	2480 MHz	39		•	

Figura 2.2 Associazione delle frequenze utilizzate da Bluetooth rispetto al tipo di canale

Nel momento in cui il livello di collegamento è sincronizzato in tempo, frequenza e indirizzo di accesso, allora il canale viene definito connesso (o semplicemente sincronizzato nel caso di un canale fisico periodico). Inoltre le specifiche del protocollo precisano che verrà utilizzato un ordinamento di tipo Little Endian per la definizione di campi nei pacchetti e per i PDU. Avremo inoltre che il bit meno significativo sarà il primo ad essere inviato sul canale.

Con l'introduzione di due nuovi livelli fisici per il BLE, in Bluetooth 5.0 sono presenti 3 differenti livelli fisici:

- LE 1M Uncoded con 1M/s di bitrate rappresenta il livello già presente in BT 4.0
- LE Coded PHY che supporta fino a 1M/s facendo uso di una codifica con correzione di errore, consente di estendere la portata di trasmissione quadruplicata rispetto a BT 4.2
- LE 2M PHY raddoppia la velocità di trasmissione e supporta sino a 2M/s

A causa della ancora bassa diffusione di BT 5.0 e allo scopo di garantire il funzionamento con la maggior parte dei dispositivi attuali, è stato scelto di rendere compatibile il protocollo con la versione di BT 4.0 e quindi si farà uso del canale LE 1M Uncoded.

Passando ad un'analisi su un più alto livello è possibile definire il formato di tale pacchetto utilizzato per entrambi i canali di *advertising* e dati. Il pacchetto contiene quattro campi obbligatori (preamble, indirizzo di accesso, PDU e CRC).

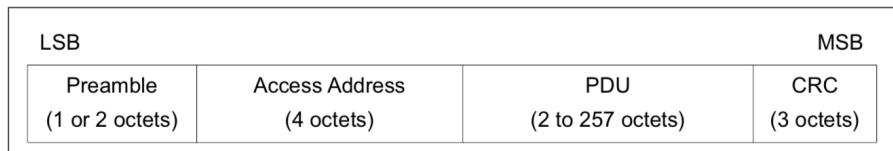


Figura 2.3 Formato pacchetti BLE (uncoded PHY)

- Il preamble viene utilizzato dal ricevitore al fine di effettuare la sincronizzazione in frequenza, la stima dei tempi di ritardo e l'allenamento dell'*Automatic Gain Control*. Il preamble per il canale LE 1M PHY è costituito da una sequenza lunga 8 bit costituita dall'alternanza di 0 e 1.
- L'indirizzo di accesso è un indirizzo costituito da 32 bit che deve soddisfare numerosi vincoli. Poiché BLE utilizza un numero limitato di canali, risulta necessario avere un indirizzo che consenta di identificare univocamente la connessione per evitare che due connessioni si sovrappongano sullo stesso canale. I vincoli che deve soddisfare hanno invece lo scopo di continuare le operazioni del preamble e consentono anche di poter eseguire un filtraggio hardware per risvegliare il ricevitore solo se particolari condizioni vengono soddisfatte, risparmiando così energia in ricezione. Va notato che l'indirizzo di accesso per i pacchetti in broadcast è fissato a 0x8E89BED6.
- Il PDU può assumere due forme in base al canale di trasmissione utilizzato. Per un pacchetto inviato su un canale di *advertising* si ha un *Advertising Channel PDU*, altrimenti nel caso di una trasmissione sul canale fisico dati si utilizza un *Data Channel PDU*.
- Ogni pacchetto inviato avrà un CRC di 24 bit calcolato sull'intero PDU che garantisce l'integrità dei dati.

2.4 Advertisement Bluetooth

I dispositivi bluetooth inviano i pacchetti di *advertising* per trasmettere dati in broadcast. Il PDU di *advertising* ha un header lungo 16 bit e un payload di lunghezza variabile fino a 31 bytes per Bluetooth Legacy e fino a 255 bytes per BT 5.0.

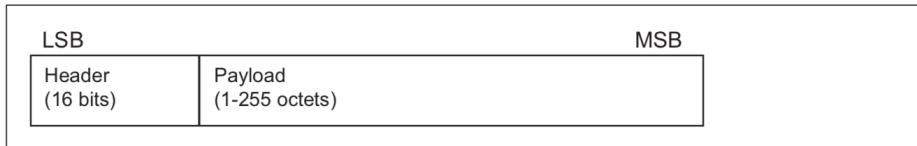


Figura 2.4 Formato del PDU di Advertising

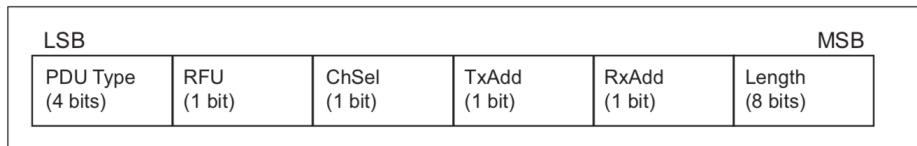


Figura 2.5 Formato dell'header del PDU di Advertising

I dispositivi bluetooth inviano i pacchetti di *advertising* per trasmettere dati in broadcast. Tali pacchetti possono trasportare un payload contenente fino a 31 bytes di informazioni disponibili allo sviluppatore (255 bytes per BT 5.0). In Bluetooth 5 si possono distinguere otto pacchetti di *Advertising* identificati da differenti valori nel campo header. Il campo payload dei pacchetti di *advertising* è strettamente vincolato al tipo di pacchetto.

- ADV_IND
- ADV_DIRECT_IND
- ADV_NONCONN_IND
- ADV_SCAN_IND
- ADV_EXT_IND
- AUX_ADV_IND
- AUX_SYNC_IND
- AUX_CHAIN_IND

Questi pacchetti sono inviati dal dispositivo in stato di *advertising* e ricevuti da un dispositivo in stato di scansione o inizializzazione. In particolare i primi quattro sono definiti pacchetti di *legacy advertising* e sono supportati già in bluetooth 4.0, gli ultimi 4 sono invece chiamati pacchetti di *extended advertising* e sono stati introdotti nell'ultima versione.

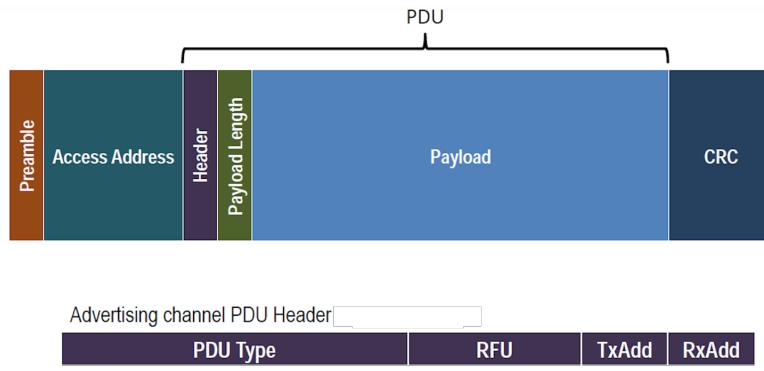


Figura 2.6 Formato generico di un pacchetto BLE di *legacy advertising*

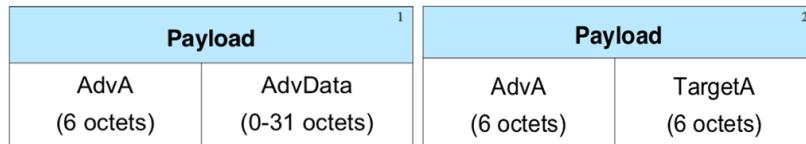


Figura 2.7 Formato 1 e 2 (da sinistra a destra) del Payload relativo al PDU di Advertising

Nelle figure 2.6 e 2.7 è possibile notare sia la struttura complessiva del pacchetto BLE fino al livello del PDU, sia i formati del payload utilizzati dai pacchetti di *legacy advertising*. Il PDU di tipo ADV_IND utilizza il formato 1 del payload mostrato in Fig.2.7. Questo tipo di pacchetto dovrebbe essere utilizzato per eventi di *advertising* con richiesta di connessione (*connectable*) ed eventi scansionabili e indiretti (*scannable undirected*). Il valore di TxAdd specifica se l'indirizzo di *advertising* (AdvA) è pubblico o random rispettivamente con i valori di TxAdd uguali a zero e uno. Il campo AdvA deve contenere l'indirizzo di *advertising* mentre AdvData contiene i dati inviati dall'host del trasmettitore (*advertiser*).

Il pacchetto di tipo ADV_DIRECT_IND è caratterizzato dal payload del secondo formato. Viene utilizzato per eventi di *advertising* diretti e collegabili (*connectable*). Il valore specificato in TxAdd specifica la tipologia dell'indirizzo dell'*advertiser*, mentre RxAdd indica il tipo di indirizzo del destinatario. Il payload è costituito da due campi, AdvA contiene l'indirizzo (pubblico o casuale) dell'*advertiser*, TargetA contiene l'indirizzo (pubblico o casuale) del destinatario.

I pacchetti che dovrebbero essere utilizzati per eventi indiretti non collegabili e non scansionabili sono quelli di tipo ADV_NONCONN_IND. Il valore di TxAdd e il formato del payload è analogo a quello dei pacchetti ADV_IND.

Il pacchetto ADV_SCAN_IND utilizza il Formato 1 del Payload relativo al PDU di *advertising* mostrato in Fig.2.7 e dovrebbe essere utilizzato per eventi indiretti e scansionabili.

Per comprendere il significato degli eventi si possono dare delle definizioni:

- Collegabile (*connectable*) quando uno scanner può iniziare una connessione dopo la notifica di tale evento.
- Scansionabile (*scannable*) quando uno scanner può iniziare una richiesta di scansione dopo la ricezione di questo evento.
- Indiretto (*undirected*) nel caso di una trasmissione in broadcast senza specificazione di un indirizzo.

Per completezza viene anche mostrata la struttura e la relazione che intercorre tra i vari tipi di indirizzi BLE specificati nel protocollo GAP.

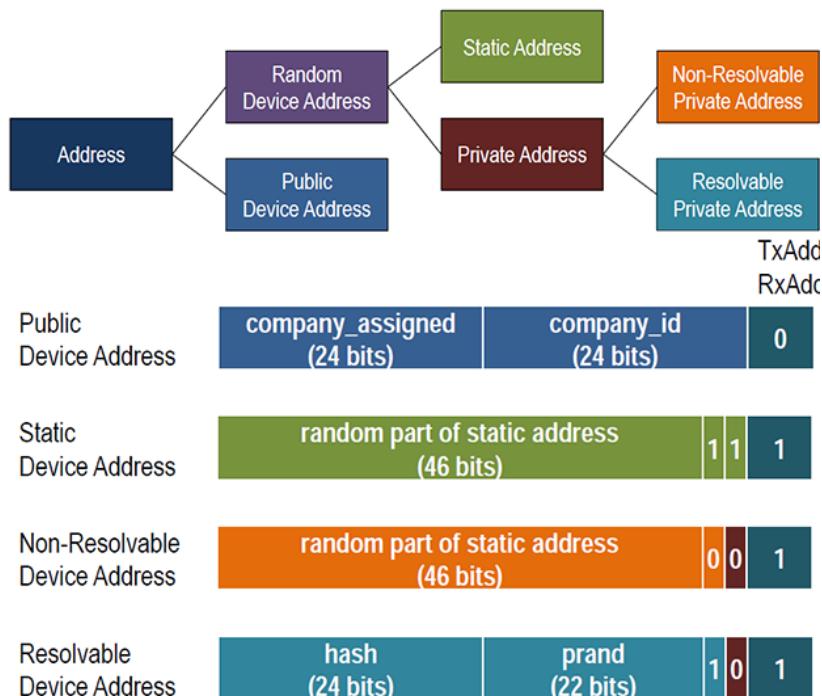


Figura 2.8 Rappresentazione dei tipi di indirizzi usati da BLE (come previsto dal livello GAP)

Gli indirizzi posseggono differenti proprietà in base alla loro tipologia. Un indirizzo statico non può essere cambiato fino allo spegnimento del dispositivo, un indirizzo privato può

cambiare nel tempo e un indirizzo risolvibile può essere usato per derivare il vero indirizzo. L'indirizzo casuale rappresenta un meccanismo utilizzato per garantire una migliore privacy evitando il tracciamento e l'intercettazione di uno specifico dispositivo.

2.5 Beacon

Un beacon, nell'ambito delle comunicazioni wireless, rappresenta un frammento di informazione trasmesso in aria (broadcast). Il contenuto del messaggio può essere qualunque: una temperatura, un comando, una stringa, una direzione, una posizione e così via. Possiamo distinguere due differenti tipi di trasmettitori, alcuni definiti statici in quanto inviano sempre la stessa informazione, altri definiti dinamici in quanto caratterizzati dall'invio di dati mutevoli nel tempo. La tecnologia BLE sembra quindi un ottimo supporto per l'invio di beacon a corto raggio, garantendo bassi consumi, bassi costi e un'elevata compatibilità. Ad oggi, la maggioranza degli smartphone è ormai dotata di Controller Bluetooth compatibili con lo standard BLE e i sistemi operativi più diffusi hanno già implementato le API per l'utilizzo di tale tecnologia. Bisogna precisare che in tale elaborato, d'ora in avanti, ci si riferirà al termine Beacon facendo implicitamente accezione ai Beacon BLE, ovvero all'utilizzo degli Advertisement forniti dalla tecnologia Bluetooth Low Energy per l'invio di frammenti di dati. In Fig. 2.9 e Fig. 2.10 si può notare la struttura e la composizione di un pacchetto AltBeacon.

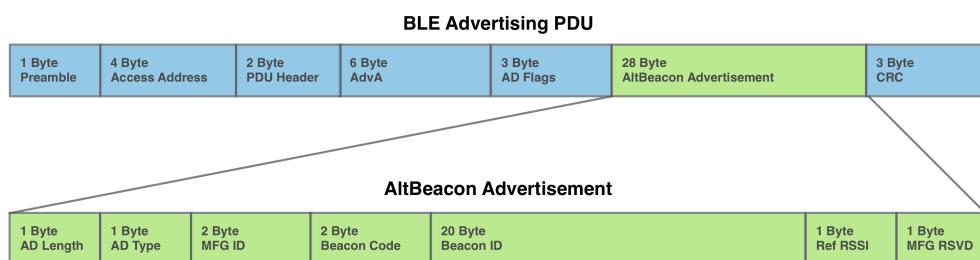


Figura 2.9 Struttura esplosa del pacchetto AltBeacon

Offsets	Octet	0	1	2	3
Octet	Bit	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31		
0	0	AD LENGTH	AD TYPE	MFG ID	
4	32	BEACON CODE		BEACON ID	
8	64		BEACON ID (CONTINUED)		
12	96		BEACON ID (CONTINUED)		
16	128		BEACON ID (CONTINUED)		
20	160		BEACON ID (CONTINUED)		
24	192	BEACON ID (CONTINUED)		REFERENCE RSSI	MFG RESERVED

Figura 2.10 Campi del pacchetto AltBeacon

Bisogna notare che ogni dispositivo con Android 4.3+ e un chipset BLE può rilevare beacon. In particolare nel 2017, in accordo con il Google Play Store, il 92% dei dispositivi soddisfaceva tali requisiti.

2.6 HCI UART

Per ottenere una buona flessibilità e garantire un'elevata compatibilità fra dispositivi eterogenei si è pensato di realizzare una libreria in linguaggio python che possa consentire un semplice quanto immediato utilizzo del protocollo. Per rendere la libreria indipendente dalla piattaforma ed evitare la dipendenza da moduli o applicativi di alto livello, si è pensato di utilizzare la libreria BlueZ che fornisce, tra le altre cose, un'implementazione a basso livello dell'HCI UART. L'obiettivo del livello di trasporto HCI UART è quello di rendere possibile l'utilizzo di Bluetooth HCI tra due interfacce UART sullo stesso PCB.

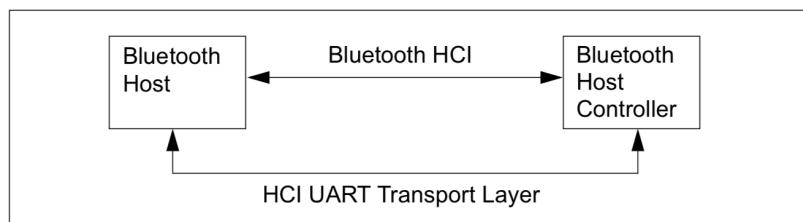


Figura 2.11 Schema semplificativo del livello di trasporto HCI UART

Per avere una visione complessiva si può osservare la Figura 2.12 che mostra il percorso seguito dai dati trasferiti da un dispositivo ad un altro. In particolare si ha che il driver HCI sull'host consente lo scambio di dati e comandi con il firmware HCI sull'hardware Bluetooth.

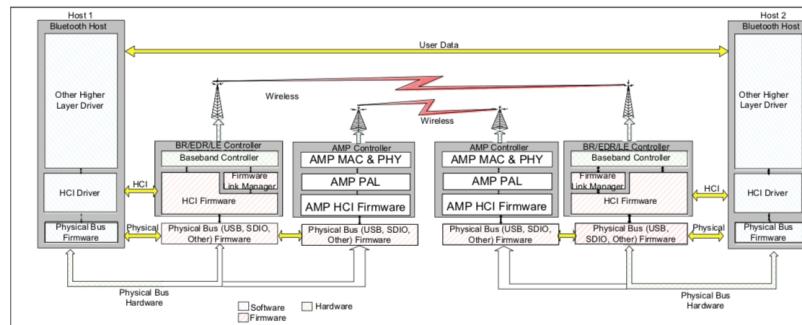


Figura 2.12 Coinvolgimento del livello di trasporto HCI durante una comunicazione

Ci sono quattro tipi di pacchetti HCI che è possibile inviare sul livello di trasporto UART a cui corrispondono quattro codici identificativi che vengono anteposti ad ogni pacchetto e seguiti dalla lunghezza del pacchetto.

HCI packet type	HCI packet indicator
HCI Command Packet	0x01
HCI ACL Data Packet	0x02
HCI Synchronous Data Packet	0x03
HCI Event Packet	0x04

Figura 2.13 Tabella dei tipi di pacchetti HCI e relativi codici

Tra questi possiamo dire che i comandi e gli eventi sono inviati tra host e controller. Infatti il software implementato si basa sull'invio di comandi e sull'utilizzo di eventi asincroni che vengono notificati all'host quando accade qualcosa. Quando l'host rileva tale evento dovrà scansionarlo e filtrarlo per determinare quale particolare evento si è verificato. BLE offre un numero ridotto di comandi ed eventi rispetto allo standard completo, ma essendo comunque assai numerosi si propone di seguito un accenno per l'utilizzo di alcuni dei comandi e degli eventi utilizzati nell'implementazione. Nel caso di un comando risulta necessario distinguere due campi che identificano univocamente l'azione che si intende eseguire. L'OGF (Opcode Group Field) specifica la categoria di appartenenza del comando e l'OCF (Opcode Command Field) specifica invece il comando. In aggiunta sono spesso necessari dei parametri che indichino gli argomenti da passare a tale comando. In BlueZ è possibile inviare dei comandi tramite l'apertura di un socket HCI e noti i valori suddetti tramite l'istruzione `hci_send_cmd()` di cui si mostra il prototipo:

```
int hci_send_cmd(int sock, uint16_t ogf, uint16_t ocf, uint8_t plen, void *param);
```

Troviamo la presenza di un ulteriore parametro (*plen*) che specifica la lunghezza dei parametri del comando. Chiamare l'operazione di lettura su un socket HCI significa attendere fino alla ricezione del prossimo evento comunicato dal controller. Un evento è costituito da un header, che ne specifica il tipo, seguito dai relativi parametri. Per comprendere il funzionamento basti pensare al caso in cui un dispositivo volesse eseguire una scansione e ottenere le informazioni così ricevute. Sarà sufficiente inviare il comando con OCF settato a OCF_INQUIRY e poi attendere filtrando eventi del tipo EVT_INQUIRY_RESULT e EVT_INQUIRY_COMPLETE. Per una lista esaustiva dei codici è possibile consultare il codice sorgente della libreria BlueZ o le specifiche del Core Bluetooth [10].

Viene ora mostrata la sintassi da utilizzare per i comandi maggiormente utilizzati per l'implementazione del protocollo. Si noti che l'OGF che deve essere utilizzato per i comandi da inviare al Controller sarà pari a 0x08.

LE Set Advertising Parameters Tale istruzione viene utilizzata dall'host per configurare i parametri di advertising. La sintassi costituita da OCF e parametri è mostrata nella tabella seguente.

Command	OCF	Command parameters	Return Parameters
HCI_LE_Set_Advertising_Parameters	0x0006	Advertising_Interval_Min, Advertising_Interval_Max, Advertising_Type, Own_Address_Type, Peer_Address_Type, Peer_Address, Advertising_Channel_Map, Advertising_Filter_Policy	Status

Figura 2.14 Formato del comando LE Set Advertising Parameters

A seguire viene mostrata la struttura dei parametri:

- Advertising_Interval_Min and Advertising_Interval_Max rappresentano il massimo e il minimo intervallo di *advertising*. In tale campo è possibile specificare un valore nell'intervallo 0x0020-0x4000 che specifica un tempo appartenente all'intervallo (20ms - 10.24s).

- Advertising_Type può assumere i valori rappresentanti il tipo di advertisement (ADV_IND, ADV_NONCONN_IND, etc).
- Own_Address_Type e Peer_Address_Type specificano il tipo di indirizzo (pubblico, casuale, ...) di entrambi i dispositivi coinvolti nella comunicazione.
- Advertising_Channel_Map rappresenta una bitmap da impostare per trasmettere su solo alcuni dei canali possibili (37, 38, 39). Specificando un valore di xxxxxxxx1b si utilizzerà il canale 37 mentre con un valore di xxxx111b si utilizzeranno tutti e tre i canali.
- Advertising_Filter_Policy consente di eseguire un filtro basato su una lista di dispositivi autorizzati.

Se il comando viene eseguito con successo verrà restituito uno stato rappresentante il valore 0x00.

LE Set Advertising Data Tale comando viene utilizzato per definire i dati che dovranno essere inviati nella fase di advertising. Il suo uso è quindi limitato ai pacchetti di *advertising* che posseggono un campo AdvData.

Command	OCF	Command parameters	Return Parameters
HCI_LE_Set_Advertising_Data	0x0008	Advertising_Data_Length, Advertising_Data	Status

Figura 2.15 Formato del comando HCI LE Set Advertising Data

Dove:

- Advertising_Data_Length indica la dimensione di Advertising_Data espressa in numero di byte (0x00-0x1F).
- Advertising_Data può contenere fino a 31 bytes di dati.

In caso di corretta esecuzione del comando verrà restituito un valore pari a 0x00.

LE Set Advertising Enable Questo comando viene utilizzato al fine di richiedere al controller di iniziare o fermare l'invio di advertisement. Il controller utilizzerà il periodo di trasmissione specificato tramite LE Set Advertising Parameters.

Command	OCF	Command parameters	Return Parameters
HCI_LE_Set_Advertising_Enable	0x000A	Advertising_Enable	Status

Figura 2.16 Formato del comando HCI LE Set Advertising Data

La sintassi è molto semplice e presenta un solo parametro definito Advertising_Enable che può assumere due soli valori: 0x00 per indicare lo stato di spegnimento e 0x01 per abilitare l'invio di advertisement.

LE Set Scan Parameters Questo comando va utilizzato per la specifica dei parametri di scansione in base alle esigenze dell'applicazione.

Command	OCF	Command parameters	Return Parameters
HCI_LE_Set_Scan_Parameters	0x000B	LE_Scan_Type, LE_Scan_Interval, LE_Scan_Window, Own_Address_Type, Scanning_Filter_Policy	Status

Figura 2.17 Formato del comando HCI LE Set Scan Parameters

Risulta possibile utilizzare 5 parametri:

- LE_Scan_Type indica se la scansione deve essere di tipo passivo (0x00) o attivo (0x01). Una scansione attiva è caratterizzata dall'invio di PDU di scansione.
- LE_Scan_Interval consente di definire l'intervallo temporale che deve trascorrere tra due scansioni LE successive. Tale campo può assumere un valore compreso tra 0x0004 e 0x4000 (che indica un range temporale da 2.5 ms a 10.24 s).
- LE_Scan_Window consente di definire la durata della scansione LE e dovrebbe essere minore o uguale dell'intervallo definito. Tale campo può assumere gli stessi valori già visti in LE_Scan_Interval.
- Own_Address_Type rappresenta il tipo di indirizzo da utilizzare nel caso di invio di PDU di scansione.
- Scanning_Filter_Policy consente di specificare le politiche di filtraggio. Di default vengono accettati tutti i pacchetti di *advertising* eccetto quelli diretti e inviati ad altri dispositivi.

2.7 Raspberry

Raspberry Pi è una famiglia di computer low-cost dalle dimensioni molto ridotte e con la capacità di interagire con il mondo esterno tramite l'utilizzo dei pin di I/O di cui è dotata. Sono presenti anche le componenti usuali come una scheda di rete, una scheda grafica e dispositivi di memorizzazione. Ad oggi sono stati prodotti diversi modelli e in tale contesto verranno descritti i due modelli utilizzati durante la fase di implementazione. La Raspberry Pi 3 rappresenta uno degli ultimi modelli e possiede le seguenti specifiche tecniche:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN con Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 40-pin extended GPIO
- 4 porte USB 2
- Uscita stereo a 4 contatti e una porta video composito
- HDMI
- Porta CSI per la connessione del modulo fotocamera
- Porta per display DSI per la connessione di un display touchscreen
- Porta Micro SD per il caricamento del sistema operativo e la memorizzazione di dati
- Porta Micro USB con corrente di alimentazione fino 2.5A

I bassi consumi, le specifiche avanzate e le dimensioni ridotte rendono la Raspberry una soluzione hardware ideale per la fase di prototipazione e realizzazione di oggetti smart.

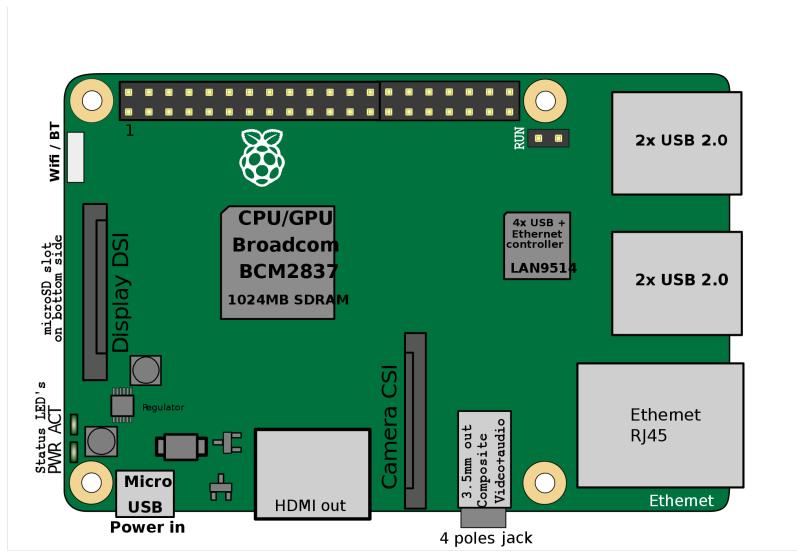


Figura 2.18 Rappresentazione della Raspberry Pi 3B con rispettive porte

La seconda versione prende il nome di Raspberry Pi 0W e possiede una totale compatibilità software con la Raspberry Pi 3 pur possedendo costi e dimensioni particolarmente ridotti. Le specifiche sono elencate di seguito:

- 1GHz, single-core CPU
- 512MB RAM
- Mini HDMI e porte USB On-The-Go
- Alimentazione tramite Micro USB
- 40-pin header
- Composite video e reset headers
- Porta CSI
- 802.11 b/g/n wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

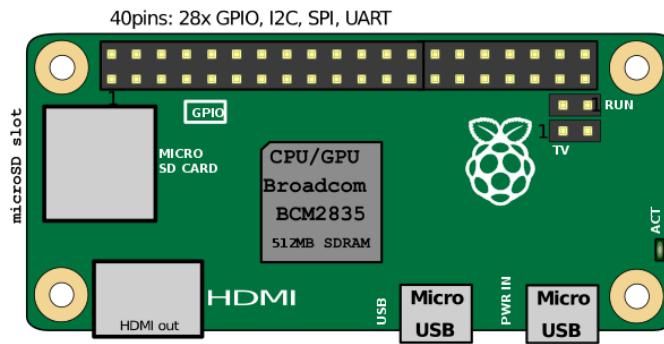


Figura 2.19 Rappresentazione della Raspberry Pi 0 con rispettive porte disponibili

Le Raspberry possono eseguire differenti sistemi operativi dei quali i più comuni sono sicuramente i sistemi basati su kernel Linux tra i quali spicca in particolare (per notorietà) Raspbian. Il nome, non casuale, indica da un lato l'elevata compatibilità con la Raspberry e al contempo la parentela con Debian, famosa distribuzione su cui Raspbian è basata. Per il progetto si è utilizzata l'ultima versione, Raspbian Stretch, con i seguenti software applicativi installati e configurati:

- BlueZ libreria utilizzata per la comunicazione con il controller bluetooth tramite socket HCI.
- Apache server HTTP per la gestione delle richieste web.
- Hostapd (Host access point daemon) utilizzato per la creazione di un AP Wi-fi.
- Dnsmasq utilizzato come server DHCP.
- Python > 3.4 (presente di default sulla quasi totalità delle recenti distribuzioni)

Tale lista rappresenta i requisiti software necessari per l'implementazione del sistema su ambienti Linux. A causa della possibile complessità di configurazione per utenti poco esperti, l'installazione di tali software e la loro configurazione può essere evitata clonando il file di immagine fornito che contiene l'intero sistema pronto all'uso. Questo approccio è possibile solo facendo uso di una Raspberry che, come è possibile ricordare, effettua il caricamento del sistema operativo usando come sorgente la micro SD.

2.8 Crittografia

Allo scopo di offrire una buona sicurezza è stato necessario scegliere ed implementare algoritmi crittografici oltre che vari meccanismi ad-hoc. In particolare, come spiegato in maggior dettaglio successivamente, tutti i pacchetti inviati dall'utente saranno cifrati. Si è fatto uso di uno dei migliori algoritmi simmetrici oggi disponibili: AES con chiave a 256bit.

L'AES è un cifrario a blocchi, il payload in chiaro viene suddiviso in blocchi di lunghezza fissa (128bit) che vengono cifrati e successivamente riuniti per comporre il messaggio cifrato. La dimensione della chiave può avere tre valori: 128, 192, 256 bit. L'input e l'output dell'algoritmo sono sequenze di bit che nell'implementazione sono trattati come array (matrici) di byte. Quindi avremo una matrice 4×4 di dimensione totale 16 byte (essendo ogni coefficiente della matrice rappresentato da 1 byte). La chiave invece sarà rappresentata, nelle sue tre versioni (128/192/256 bit), rispettivamente da matrici 4×4 , 4×6 , 4×8 . Si illustra adesso il funzionamento dell'algoritmo di cifratura AES definendo prima alcuni parametri:

- *word* per indicare una singola colonna di un blocco.
- $N_B = 4$ per indicare il numero di word in ogni blocco
- $N_K = [4, 6, 8]$ per indicare il numero di word della chiave (128/192/256bit)
- $N_R = [10, 12, 14]$ per indicare il numero dei round

L'algoritmo è costituito da due funzioni principali: *KeyExpansion* e *Cipher*. *Cipher* è la funzione che si occupa della codifica del messaggio, *KeyExpansion* è un generatore di chiavi: infatti ad ogni round viene utilizzata una chiave diversa per la cifratura e tali chiavi sono generate proprio dalla funzione *KeyExpansion* e passate come argomento alla funzione *Cipher*. Il numero di chiavi necessarie è dato dal numero di round più una ($N_R + 1$) poiché proprio $N_R + 1$ volte viene eseguita *AddRoundKey* (unica funzione a far uso della chiave). Inoltre le operazioni non vengono effettuate direttamente sull'input ma su una matrice ausiliaria che prende il nome di State (o matrice di stato, rappresentando effettivamente lo stato delle operazioni). Inizialmente l'input viene copiato in tale matrice, alla terminazione la matrice State verrà usata per prelevare l'output. La copiatura avviene per colonne, quindi i primi 4 byte in ingresso formeranno la prima word (colonna). Di seguito viene mostrato il diagramma per l'operazione di cifratura del blocco in AES.

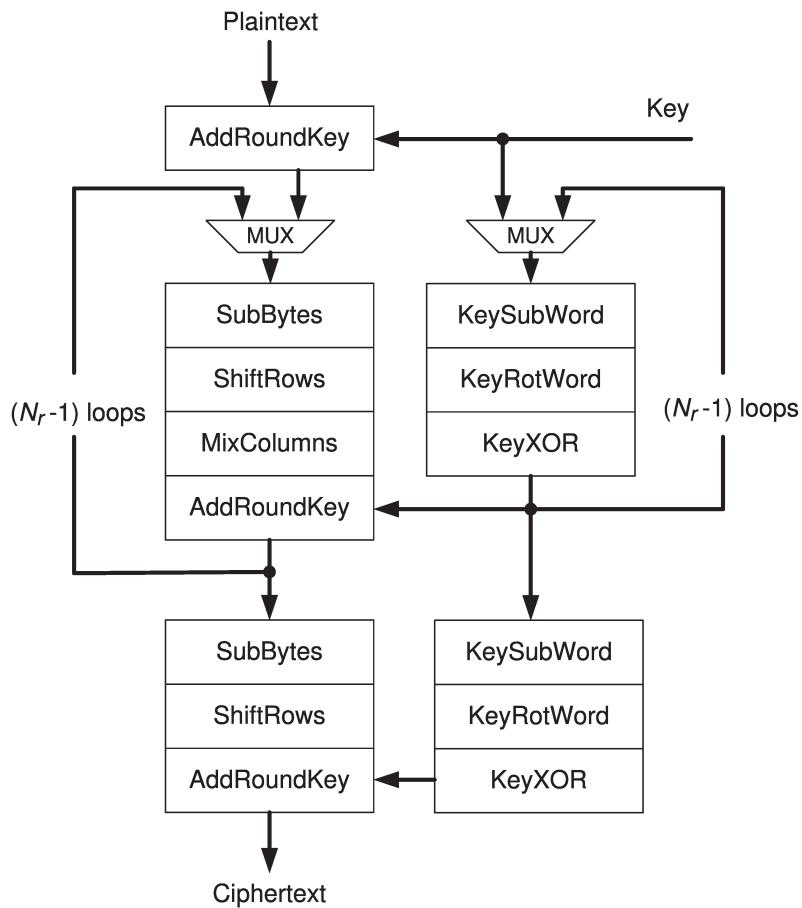


Figura 2.20 Diagramma del funzionamento dell'algoritmo di cifratura di AES

Nel caso specifico si è utilizzata AES in modalità CBC (Cipher Block Chaining). In tale modalità ogni blocco del payload in chiaro viene XORato con l'ultimo testo cifrato e, nel caso del primo blocco, con quello che prende il nome di IV (Vettore di Inizializzazione). L'IV cambiando nel tempo, garantisce che due pacchetti in chiaro uguali vengano cifrati in modo differente rendendo più complessi attacchi di analisi o a forza bruta. AES CBC viene spesso utilizzata con il padding PKCS#7 che necessita dell'utilizzo di un byte per blocco per indicare il numero degli ottetti che sono stati paddati. Tuttavia si è preferito mantenere una struttura dei pacchetti fissa consentendo di ottenere un byte aggiuntivo rispetto a tale soluzione.

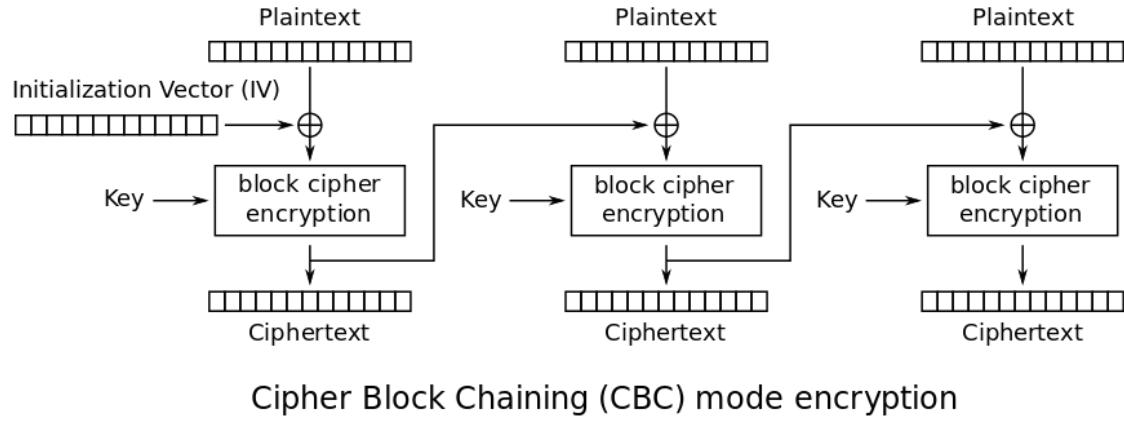


Figura 2.21 Diagramma del funzionamento dell'algoritmo AES in modalità CBC

Allo scopo di garantire l'autenticità si è invece utilizzato un MAC e più in particolare un approccio del tipo *encrypt then MAC* utilizzando come controllo di integrità il CRC offerto dallo stack BLE.

Capitolo 3

Smart Environment

3.1 Descrizione

Uno *smart environment* può essere definito come un luogo in cui sono presenti oggetti intelligenti e interconnessi che posseggono la capacità di interagire in modo naturale con l'uomo. Una interessante definizione è quella proposta da Mark Weiser che descrive così uno *smart environment*:

A physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network[14].

Bisogna notare che i dispositivi intelligenti, interconnessi tra loro, devono consentire lo scambio di dati al fine di poter consentire l'interazione. In particolare ogni dispositivo deve avere dei sensori ed attuatori oltre che la capacità di trasmettere informazioni all'interno di uno spazio che può, in tal senso, essere definito come uno spazio intelligente (o Smart Space). Affinché si abbia una interazione con l'uomo è necessario che gli utenti abbiano con sé altri dispositivi intelligenti (ad es. degli smartphone) che consentano la partecipazione allo smart space. In tal senso l'insieme costituito da oggetti intelligenti (o smart object) e spazi fisici intelligenti viene definito ambiente intelligente (o *smart environment*). Più precisamente è possibile identificare diverse applicazioni per gli ambienti intelligenti e durante la realizzazione si è cercato di mantenere un approccio che consenta di garantire tale generalità al fine di ottenere un sistema flessibile che possa essere applicato in più contesti.

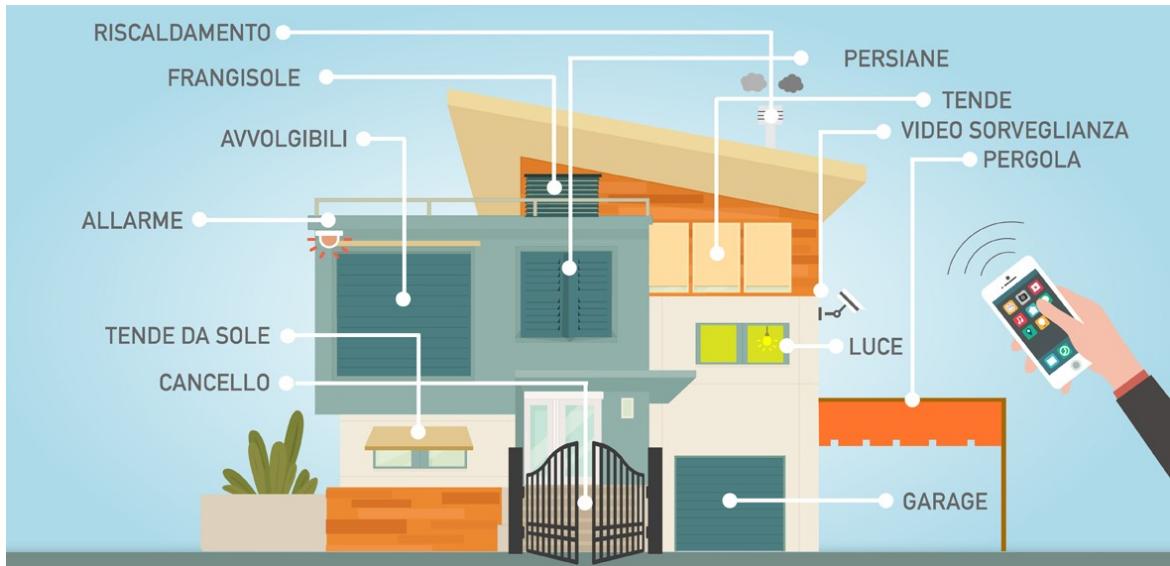


Figura 3.1 Rappresentazione di una possibile applicazione di uno smart environment

Senza perdere di generalità durante la fase di progettazione si è pensato di realizzare uno *smart environment* che consenta l’interazione tra un utente dotato di smartphone e uno spazio intelligente. Oggi esistono varie alternative che si propongono come soluzione a tale problema. Fra le più note troviamo Alexa e Google Home, sistemi cloud-based che consentono di rendere gli ambienti interattivi tramite comandi vocali. Tuttavia tali soluzioni presentano limiti e difetti ed in tal senso si è ritenuto interessante proporre un’alternativa, sviluppando un sistema che abbia le stesse funzioni ma che miri a ottenere dei vantaggi rispetto alle alternative presenti. Volendo descrivere lo smart-environment nella sua totalità si può pensare al seguente scenario:

Un utente desidera partecipare ad uno *smart environment* e chiede di essere registrato. Alla registrazione verrà associato un gruppo ed un codice identificativo univoco. Questa fase non dovrà più essere ripetuta. Una volta registrato, l’utente (o più esattamente il dispositivo dell’utente) sarà in grado di rilevare la presenza di uno *smart environment* autonomamente. Tuttavia, prima di interagire con esso, l’utente dovrà seguire delle procedure di autorizzazione e aggiornamento che gli consentiranno di identificarsi e di acquisire maggiore conoscenza sull’ambiente circostante ovvero informazioni su quali oggetti è possibile interagire e quali azioni è possibile eseguire. Dopo aver eseguito tali procedure, che dovranno essere ripetute ad ogni nuova visita dell’ambiente intelligente (con un timeout definito dal custode dell’ambiente), sia l’utente sia l’ambiente sono pronti per l’interazione. L’utente potrà identificare l’oggetto con il quale vuole interagire puntando su di esso lo smartphone e potrà specificare l’azione che intende eseguire mediante un comando vocale (ad es. "Accenditi"). Le ultime due azioni verranno eseguite da una rete neurale il cui compito è quello di ricevere in ingresso

un’immagine ed un comando vocale e restituire in uscita il codice identificativo univoco dell’oggetto con il quale si vuole interagire e il comando da eseguire in forma codificata. A questo punto il comando verrà trasmesso al dispositivo puntato che dovrà provvedere all’autenticazione del mittente e solo eventualmente all’elaborazione della richiesta espressa dall’utente. Le interazioni tra lo smartphone dell’utente e gli oggetti avvengono utilizzando un protocollo basato su tecnologia BLE.

3.2 Descrizione dei requisiti

Per una descrizione più dettagliata è possibile considerare i requisiti che sono stati imposti precedentemente alla fase di progettazione. Il primo requisito è stato quello di progettare l’intero *smart environment* facendo sì che non fosse necessario l’accesso alla rete. Questa condizione, se a prima vista appare una scelta controcorrente in un mondo orientato al cloud, possiede, ad un’analisi più attenta, vari lati positivi:

1. La non transazione dei dati online.
2. Una minore latenza.
3. Una maggiore capacità di tolleranza ai guasti.
4. Una maggiore privacy e sicurezza.

Un ulteriore requisito imposto è stato quello di rendere il più possibile sicuro l’intero sistema. Per sicurezza si intende una resistenza a vari vettori di attacco (passivi e attivi) tra cui:

1. DoS: impedisce il normale funzionamento del servizio.
2. Replay: ripetizione dei pacchetti.
3. Intercettazione e modifica dei contenuti del messaggio: intrusione nel mezzo della comunicazione per leggere o modificare i dati trasmessi.
4. Bruteforce.
5. Analisi del traffico: ottenimento di informazioni come l’identità degli interlocutori o la frequenza dei messaggi.
6. Mascheramento: ottenimento di privilegi ed esecuzione di azioni che non gli competono.

E a tal fine si comprende la necessità di implementare i seguenti meccanismi:

1. Autenticazione: garantisce che l'entità comunicante è chi afferma di essere.
2. Controllo degli accessi: garantisce che le risorse vengano utilizzate da chi ne ha il permesso.
3. Confidenzialità: protezione della riservatezza dei dati.
4. Integrità: garanzia che i dati non siano stati alterati.
5. Non ripudiabilità: garanzia contro la possibilità che una delle due entità coinvolte nella comunicazione neghi di aver partecipato alla comunicazione.

Si è anche prevista la possibilità che più utenti possano essere presenti all'interno di uno stesso *smart environment* e che ognuno di essi possa interagire con più di un oggetto. Questo vincolo pone le basi per la realizzazione di un ambiente intelligente che ha nozione di chi è presente all'interno dello spazio per eseguire comandi aggiungendo quindi un nuovo tipo di conoscenza (o intelligenza) all'intero sistema. Altro requisito di principale importanza è stato quello di garantire una elevata semplicità e flessibilità per l'implementazione e l'estensione delle funzionalità degli oggetti. Ad esempio si pensi alla possibilità di voler realizzare un proprio *smart object* con particolari caratteristiche; in tal caso sarà possibile estendere il generico dispositivo in base alle proprie esigenze (si pensi ad uno frigo intelligente dove viene implementato il comando di "imposta temperatura").



Figura 3.2 Rappresentazione di possibili interazioni con oggetti intelligenti

3.3 Progettazione e implementazione

Come è possibile immaginare, la progettazione e l'implementazione di un simile sistema coinvolge diversi ambiti e tecnologie:

- Comunicazioni Wireless
- Progettazione di algoritmi, teoria dell'informazione
- Architettura software multilivello
- Riconoscimento vocale e di immagini
- Progettazione e utilizzo di sensori
- Reti di calcolatori
- Sistemi operativi e calcolo parallelo

- Sicurezza e crittografia

Vista la numerosità e l'eterogeneità delle tematiche, in questo elaborato, si porrà particolare attenzione a quelle maggiormente considerate per la realizzazione del prodotto finale. Nella fase di progettazione è stata posta particolare attenzione sulla sicurezza, compatibilità, generalità e flessibilità. In tal senso sono state effettuate interessanti scelte che hanno consentito di evidenziare i primi vantaggi e svantaggi della soluzione proposta. In tutte le fasi si è seguito un procedimento rigoroso costituito da strategie da seguire nei vari passi e criteri di scelta in caso di alternative.

L'implementazione, seguendo le linee progettuali note, ha posto in evidenza l'esigenza di realizzare dispositivi che, in modo quasi invisibile, riuscissero a rendere intelligenti gli oggetti ai quali sono collegati. Questo concetto potrebbe essere definito come la capacità di trasferire intelligenza da un dispositivo generico passivo a un dispositivo che diviene attivo e intelligente ovvero in grado di reagire ai comandi dell'utente.

3.3.1 Basi di dati

In particolare per la progettazione dati è stata seguita una metodologia assai diffusa in tale ambito che consiste nelle seguenti fasi:

- La progettazione concettuale che ha fornito uno schema concettuale (ER) rappresentante i dati di interesse per l'applicazione.
- La progettazione logica che ha come risultato uno schema logico (relazionale), dipendente dal tipo di base di dati scelto.
- La progettazione fisica per il DBMS scelto.

Per la memorizzazione di dati necessari ai fini del funzionamento dell'intero sistema si è pensato di fare uso del noto gestore di basi di dati MySQL. Di seguito vengono mostrati lo schema logico e quello concettuale (in Fig.3.3).

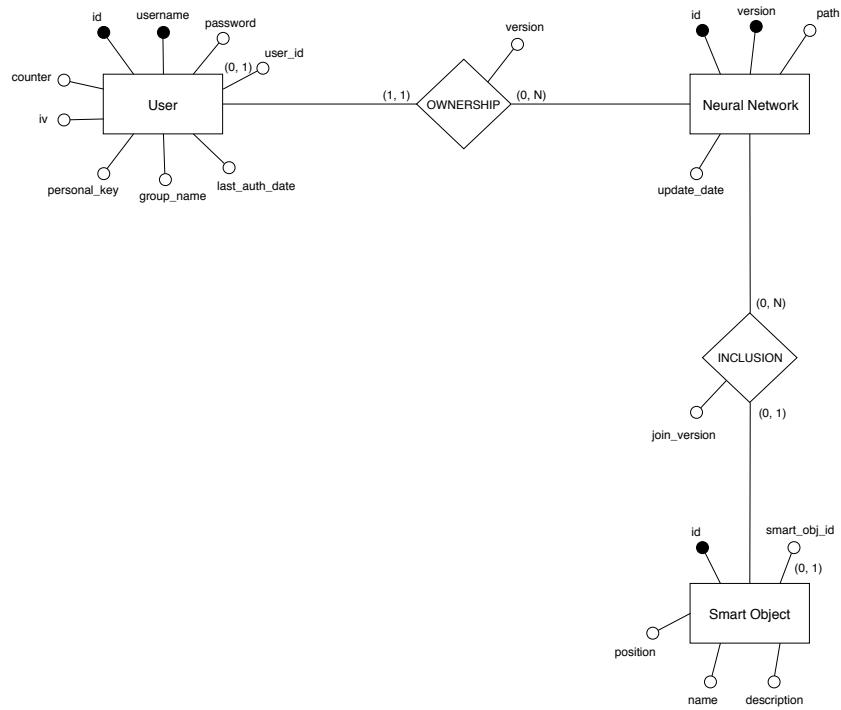


Figura 3.3 Schema concettuale (ER) realizzato in fase di progettazione

- User (**id**, **username**, **password**, **user_id**, **last_auth_date**, **group_name**, **personal_key**, **IV**, **counter**, **neural_net_version**)
- NeuralNetwork (**id**, **version**, **path**, **update_date**)
- SmartObject(**id**, **smart_obj_id**, **description**, **name**, **position**, **join_version**)

Come si può notare la base di dati è costituita da tre tabelle. La tabella Users conterrà i dati degli utenti tra cui un user_id che rappresenta univocamente un utente all'interno dello *smart environment*, il campo group_name che è indicativo dei permessi posseduti dall'utente e l'insieme di attributi (IV, counter, personal_key) che rappresenta l'insieme delle informazioni necessarie per l'implementazione dei meccanismi di sicurezza e gestione dei permessi (i dettagli verranno discussi più avanti). La tabella Smart_objects è utilizzata per tenere traccia dei dispositivi presenti nello *smart environment*. Ogni dispositivo sarà identificato all'interno dell'ambiente tramite il suo attributo obj_id (su cui è definito un vincolo intrarelazionale di

tipo unique). La relazione Neural_networks tiene traccia di tutte le versioni della rete neurale. In questo modo sarà possibile controllare la versione utilizzata dall'utente ed eventualmente avviare la procedura aggiornamento. Viene inoltre tenuta traccia della prima versione in cui uno *smart object* è incluso: infatti, nel corso del tempo, potrebbero essere aggiunti nuovi dispositivi all'interno dello *smart environment*. Bisogna notare che sia user_id sia smart_obj_id dovranno soddisfare il vincolo di *unique* ma non di *not null*.

Gli altri dettagli sulla progettazione e l'implementazione verranno discussi nei capitoli seguenti.

3.4 Funzionamento

In questa sezione si vuole esplicitare il funzionamento del sistema ad un livello più astratto così da averne un'immagine più completa. In prima istanza è possibile definire le varie fasi che, nel loro insieme, coprono tutte le funzionalità e i requisiti del progetto.

Configurazione A tale fase appartengono tutte le procedure necessarie per la messa a punto del sistema nello stato di funzionamento. In primo luogo è necessario configurare le Raspberry (che rappresentano il "cervello" degli oggetti intelligenti). La configurazione può avvenire semplicemente installando una microSD con immagine preconfigurata (Appendice B) o, alternativamente, mediante installazione e configurazione delle dipendenze necessarie al funzionamento. Un'interessante applicazione del primo metodo potrebbe essere quella utilizzato da un produttore che vuole vendere oggetti intelligenti e propone agli utenti di acquistare l'intero *smart object* compreso di microSD. Questo approccio presenta vari vantaggi come quello di poter effettuare facilmente aggiornamenti software anche in un sistema non connesso alla rete ed inoltre consente un elevato grado di flessibilità che si manifesta, ad esempio, nel caso in cui un utente volesse acquistare dei moduli che ne estendono le funzionalità. Quanto descritto fino adesso riguarda una procedura che può essere effettuata solo dal produttore del dispositivo; assumiamo per semplicità che l'amministratore del sistema acquisti direttamente gli oggetti preconfigurati e quindi pronti per la messa in funzione. Per la messa in funzione bisogna effettuare il posizionamento ed il collegamento degli oggetti in LAN. Bisogna notare che risulta necessaria la presenza di almeno uno *smart core*, un dispositivo in grado di comunicare con tutti gli altri oggetti presenti nell'ambiente. Una volta connesso lo *smart core*, sarà possibile collegarsi all'AP Wifi che verrà creato automaticamente e che consentirà la configurazione dei gruppi utente cioè la gestione delle azioni che verranno consentite o negate agli utenti. Infine sarà necessario registrare nella rete i singoli *smart object* assegnando un nome, una descrizione e una posizione. Questa

procedura avverrà ancora una volta mediante la connessione alla rete locale offerta dallo *smart core*.

Diagramma dei casi d'uso

Viene fornito per completezza e maggiore chiarezza il diagramma dei casi d'uso le cui componenti saranno analizzate brevemente di seguito e in modo più articolato nei capitoli a seguire.

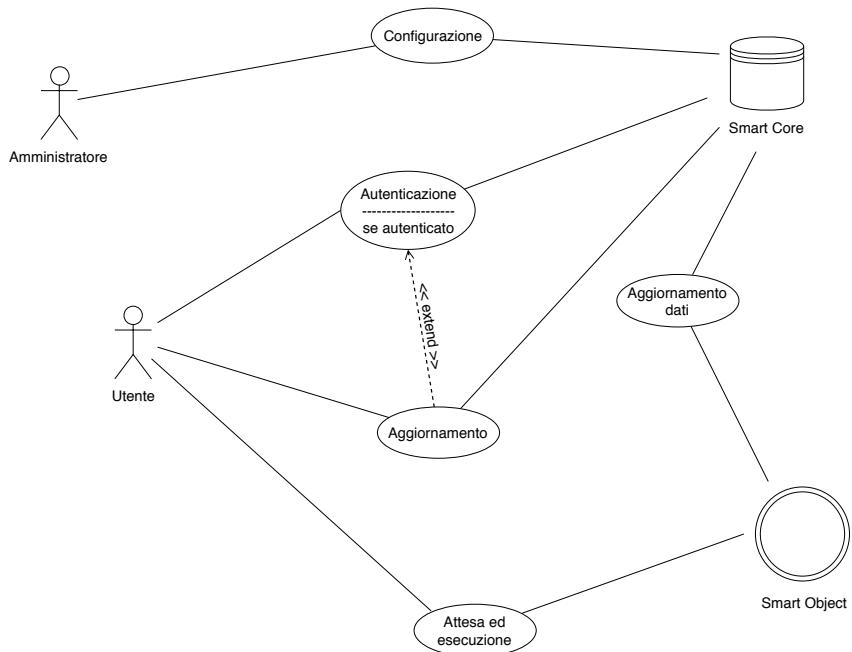


Figura 3.4 Diagramma dei casi d'uso

Registrazione Affinché un qualunque utente possa partecipare all’ambiente intelligente è necessario eseguire prima una fase di registrazione. In tale fase, che richiede una certa fiducia verso l’utente, vengono inseriti i dati di accesso (username, password), viene associato un gruppo che identificherà i permessi dell’utente e verrà restituita dal sistema una chiave a 256 bit sulla quale si fonda la sicurezza delle trasmissioni. Se pur in modo invisibile all’utente verrà anche definito un IV (vettore di inizializzazione) ed un counter (rappresentante l’ultimo numero di sequenza utilizzato durante la comunicazione). La chiave personale,

da considerarsi assolutamente privata, dovrà essere immessa nell'applicazione utilizzata dall'utente. Questa fase non dovrà più essere ripetuta per lo stesso utente.

Autenticazione (o Autorizzazione) Questa fase dovrà essere eseguita ad ogni nuova visita dello *smart environment* o, più esattamente, allo scadere di un intervallo temporale predefinito. In questa fase avviene l'identificazione dell'utente che, qualora avvenisse con successo, restituisce un permesso temporaneo nella forma di IV. In particolare ogni volta l'utente si avvicina ad uno *smart core*, quindi interno ad un ambiente intelligente, e viene identificato con successo, verrà automaticamente stabilita una connessione Wifi all'AP fornito da tale dispositivo. A questo punto vengono forniti alcuni dati come quelli necessari per il funzionamento della rete neurale e quindi del riconoscimento degli oggetti o ancora informazioni utili per poter interagire con gli oggetti ovvero la lista dei comandi, l'IV e il numero di sequenza usato per la trasmissione dell'ultima istruzione.

Esecuzione di azioni Superata con successo la procedura di autenticazione, per poter interagire con lo spazio intelligente è necessario selezionare l'oggetto con il quale si vuole interagire, inquadrando con la fotocamera, ed una volta avvenuto il riconoscimento, segnalato visivamente, sarà possibile eseguire un'azione tramite comando vocale. Tutte queste fasi possono essere schematizzate in due sole interazioni a cui verranno dedicati due capitoli di tale elaborato:

- L'interazione tra utente e *smart core*
- L'interazione tra utente e *smart object*

Capitolo 4

Il protocollo CtrlBeacon

4.1 Introduzione

I requisiti hanno determinato la necessità di progettare un nuovo protocollo che facesse uso della tecnologia nota come beacon Bluetooth. Il protocollo progettato non è vincolato dall'utilizzo su un singolo dispositivo, in quanto può essere implementato su tutti i dispositivi compatibili con Bluetooth Low Energy che soddisfano i requisiti specificati in Bluetooth Specification Version 4.0, Volume 0, Part B, Sezione 4.4 Low Energy Core Configuration o nella sezione 4.5 Basic Rate and Low Energy Combined Core Configuration. I pacchetti *CtrlBeacon* sono incapsulati come payload di un *advertising* di tipo non collegabile e indiretto (ADV_NONCONN_IND). In particolare il nuovo protocollo sarà definito sopra un altro di più basso livello chiamato AltBeacon, standard assai diffuso che garantisce un alto livello di compatibilità. I dispositivi che trasmettono beacon di *advertising* vengono definiti *advertisers*, i ricevitori vengono invece definiti *scanners*. Di seguito verranno descritti brevemente i formati dei pacchetti utilizzati dal protocollo e una spiegazione più esaustiva (ad un maggior livello di astrazione) verrà data nei seguenti capitoli.

4.2 Descrizione dei pacchetti

Il protocollo, chiamato *CtrlBeacon*, può essere pensato come costituito da tre differenti tipi di pacchetti, in altri termini tutti i pacchetti utilizzabili dovranno seguire necessariamente uno dei tre formati presenti. Per semplicità nei diagrammi verrà mostrato l'intero pacchetto AltBeacon, tuttavia solo alcuni campi verranno modificati e a tale scopo vengono indicati quelli comuni al fine di non essere ripetuti nei seguenti sottoparagrafi.

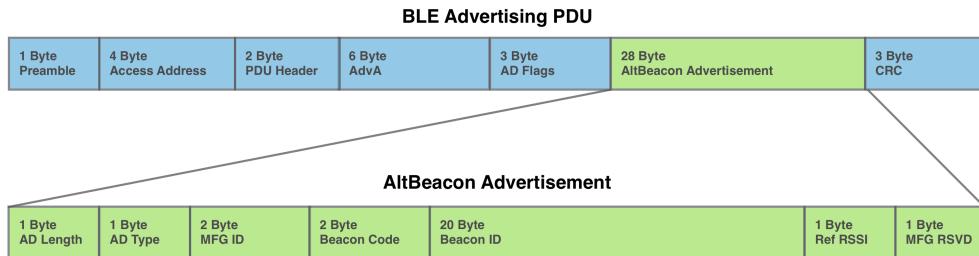


Figura 4.1 Rappresentazione del formato di un AltBeacon

In riferimento alla Fig.4.1 si possono notare i seguenti campi:

- **AD Length:** che esprime la lunghezza (espressa in byte) dell'advertisement che segue. Nel calcolo della lunghezza va escluso il byte utilizzato da questo campo. Per tale motivo in CtrlBeacon l'AD Length sarà sempre fissato a 27 (0x1B).
- **AD Type (o Advertising_Type):** come già visto nella sezione 2.6, rappresenta il tipo di advertisement. Essendo i beacon caratterizzati dall'essere indiretti e non collegabili, essi sono incapsulati come payload di un pacchetto di *advertising* ADV_NONCONN_IND (AD Type = 0x03).
- **MFG ID (o Manufacturer ID):** rappresenta l'identificatore della compagnia che ha prodotto il dispositivo. Più esattamente dovrà essere immesso come valore la rappresentazione Little Endian del produttore del dispositivo. Un elenco di tali codici è disponibile sul sito della Bluetooth SIG.[9]
- **Beacon Code:** rappresenta il tipo di beacon ed in questo caso identifica l'AltBeacon. Tale campo deve avere sempre il valore 0xBEAC pena lo scarto del pacchetto.
- **Ref RSSI (Reference RSSI):** indica la potenza media ricevuta a 1 m dall'*advertiser*. Il valore contenuto sarà della lunghezza di un byte e di tipo con segno (intervallo: [0, -127]). In caso di non calibrazione viene solitamente usato come riferimento il valore di -59.
- **MFG RSVD (Manufacturer Reserved):** un byte da utilizzare per funzionalità future. Da standard andrebbe valutato in funzione del Manufacturer ID.

Va precisato che, qualora non specificato diversamente, i valori multibyte seguiranno una rappresentazione BigEndian. Riferendoci a quanto detto prima si può notare come CtrlBeacon sia pienamente compatibile con tutti i campi di AltBeacon fatta eccezione per il Beacon ID.

4.2.1 HelloBroadcast

Il primo tipo di pacchetto è il primo pacchetto che sarà identificato dall'utente, più specificatamente verrà inviato solo dallo *smart core* allo scopo di segnalare la presenza dell'ambiente intelligente e porre le basi per l'instaurazione della connessione necessaria per l'autenticazione. Per tale motivo gli è stato dato il nome di *HelloBroadcast* (HB in forma abbreviata). Tale pacchetto avrà la seguente struttura:

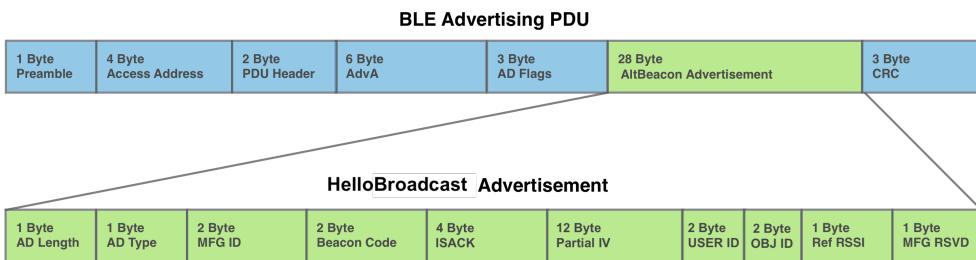


Figura 4.2 Rappresentazione del formato di un pacchetto di HelloBroadcast

Con riferimento alla Figura 4.2, possiamo considerare i vari campi di tale pacchetto.

- **ISACK:** tale valore è usato per distinguere il pacchetto che invia la password da quello utilizzato per notificare la sua avvenuta ricezione. Si avranno 4 byte settati a 0 (ISACK = 0x00) nel caso in cui il pacchetto sia inviato dallo *smart core* mentre saranno impostati tutti i bit a 1 (ISACK = 0xFFFFFFFF) nel caso di un ACK.
- **Partial IV:** rappresenta buona parte dell'IV da utilizzare per la (de)cifratura degli altri pacchetti. L'IV completo (16 byte) sarà ottenuto aggiungendo ai 12 byte del Partial IV i 4 byte dei campi seguenti (2 byte di User ID e 2 byte di Obj ID).
- **USER ID:** tale campo assume un valore fissato e pari a 0xFFFF (tutti i bit posti a uno) nel caso di un pacchetto inviato dallo *smart core* all'utente, altrimenti, nel caso sia un ACK per un pacchetto di HB, sarà indicato lo user id dell'utente. Come da convenzione, nell'ambito delle reti, anche in tale protocollo l'user id (che rappresenta più genericamente un indirizzo) che viene usato per le trasmissioni indirizzati a tutti gli utenti sarà quell'indirizzo composto da tutti i bit posti 1.
- **OBJ ID (Object ID):** questo campo sarà sempre presente ed indicherà, nel caso specifico, l'object id dello *smart core* (che riceve o invia il pacchetto).

Va notato che questo rappresenta l'unico pacchetto non cifrato e in effetti questo non causa alcun problema poiché nessuna informazione sensibile è condivisa o può essere modificata.

Del resto in mancanza della chiave, l'IV non è sufficiente ad avere abbastanza informazioni per la decifratura dei pacchetti cifrati. L'IV può, infatti, essere pubblico. Nel caso in cui il pacchetto dovesse essere un ACK verrà definito per semplicità *HelloBroadcastACK*.

4.2.2 Wifi Password

Un'altra struttura è utilizzata per la trasmissione della password necessaria per la connessione all'Access Point generato dallo *smart core*. Tale pacchetto viene inviato dallo *smart core* solo dopo che l'utente riceve un HelloBroadcast e invia l'ACK ad esso relativo che viene, a sua volta, ricevuto dallo *smart core*.

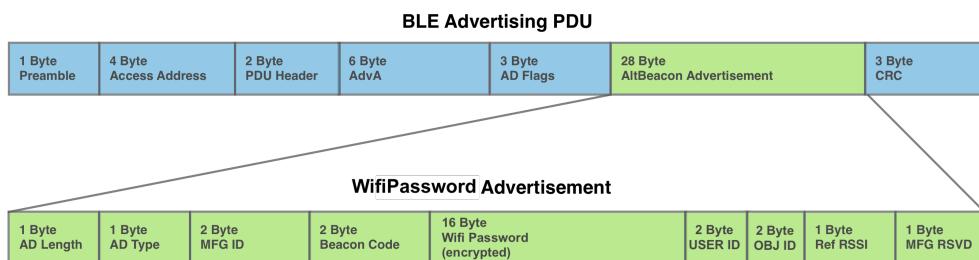


Figura 4.3 Rappresentazione del formato di un pacchetto di WiFiPassword

Il formato del pacchetto, mostrato in Figura 4.3, è caratterizzato da solo 3 campi di interesse:

- **Wifi Password:** Rappresentazione ASCII della password utilizzata dall'AP in forma cifrata. Più esattamente possiamo dire che sarà possibile ottenere il valore di tale campo nota la chiave precondivisa per l'accesso al wifi (PSK, o nel caso specifico una WPA2 PSK) tramite la seguente funzione:

$$AES_{CBC}(WPA2_PSK, PersonalKey, IV_{HelloBroadcast})$$

dove si nota come la chiave segreta dell'utente (generata in fase di registrazione) viene utilizzata come chiave di cifratura per AES in modalità CBC con chiave a 256bit e come IV vengono utilizzati i 16 byte definiti come concatenazione di Partial IV, USER ID e OBJ ID del pacchetto di HelloBroadcast.

- **USER ID:** indica lo user id dell'utente.
- **OBJ ID (Object ID):** questo campo sarà sempre presente ed indicherà, nel caso specifico, l'object id dello *smart core* (che riceve o invia il pacchetto).

4.2.3 Command

Fino adesso sono stati analizzati i due tipi di pacchetti che caratterizzano la comunicazione tra utente e *smart core*. In tale paragrafo viene analizzato il formato del pacchetto trasmesso durante la normale comunicazione, lo stesso utilizzato durante l’interazione tra l’utente e lo *smart object*.

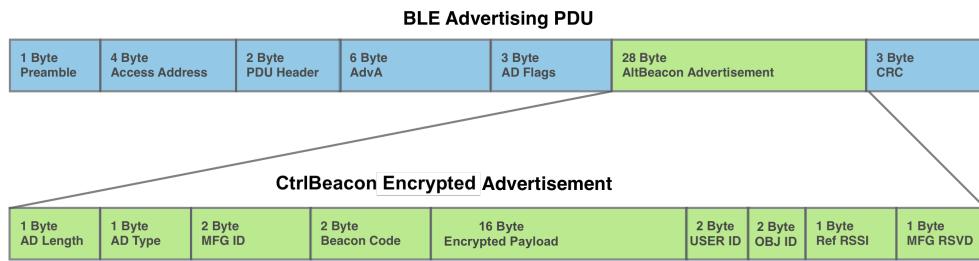


Figura 4.4 Rappresentazione del formato di un pacchetto generico cifrato usato nel protocollo CtrlBeacon

Come si può notare in Figura 4.4 tali pacchetti avranno una buona parte del contenuto in forma cifrata. Più esattamente è possibile considerare il contenuto di tale pacchetto sia in forma cifrata, sia nella sua forma decifrata, allo scopo di studiarne le informazioni trasmesse.

- **Encrypted Payload:** contiene sia informazioni sul numero di sequenza necessario per la comunicazione, sia un insieme di campi che, nel loro insieme, rappresentano il comando da inviare allo *smart object*. Possiamo immaginare che l'*Encrypted Payload* possa essere ottenuto seguendo la seguente definizione:

$$AES_{CBC}(Counter, CMD[], RES1, RES2, PersonalKey, IV_{temp})$$

dove i primi quattro parametri identificano il contenuto da cifrare e gli ultimi due argomenti sono necessari per una cifratura a chiave simmetrica AES CBC. Bisogna notare che CMD è in realtà costituito non da un solo valore ma da più campi e può, in tal senso, essere considerata un array di dati.

- **USER ID:** indica lo user id dell’utente.
- **OBJ ID (Object ID):** questo campo sarà sempre presente ed indicherà, nel caso specifico, l’object id dello *smart object* (che riceve o invia il pacchetto).

In Figura 4.5 si può osservare il pacchetto, in una forma astratta, in cui il payload è decifrato. Si presti attenzione al fatto che il pacchetto, durante la comunicazione, non si presenterà mai

in tale forma che viene mostrata solo a titolo esemplificativo. Analizzando i campi presenti in forma cifrata possiamo distinguere:

- **Counter:** il valore ad esso associato rappresenta il numero di sequenza che identifica il pacchetto.
- **CMD[]:** rappresenta una struttura dati di 6 byte che può essere scomposta in più sottocampi:
 - **CMD_Type** (1 Byte): Rappresenta la tipologia del comando. Ad esempio il valore 0xFF è stato riservato per i comandi di controllo (l'ACK rientra in tale categoria).
 - **CMD_Class** (1 Byte): Rappresenta la classe di appartenenza del comando. Degli esempi potrebbero essere la distinzione tra comandi che *eseguono azioni* da quelli che *impostano dei valori*. A tal proposito si pensi ai comandi di *apertura*, *rotazione* piuttosto che di *impostazione della temperatura* o *impostazione del volume*.
 - **CMD_OPCode** (1 Byte): In caso di appartenenza alla stessa classe e tipologia, il campo OPCode (Operation Code) specifica quale comando fra le possibili alternative si intende eseguire.
 - **CMD_Parms**(2 Byte): Due byte che possono essere utilizzati dal produttore dell'oggetto per specificare eventuali parametri. A titolo di esempio si può pensare al comando di *impostazione della temperatura* dove sarà necessario indicare un valore numerico correlato alla variazione di temperatura che si intende ottenere.
 - **CMD_Bitmask** (1 Byte): Un ulteriore byte dovrà essere utilizzato per indicare su quale attuatore eseguire l'azione. Si pensi a un byte come costituito da otto bit, se tutti i bit saranno settati a 1 tutti e gli otto dispositivi dovranno eseguire il comando. In questo modo si riesce a utilizzare un solo *smart object* per eseguire azioni su più attuatori con un singolo comando.
- **RES1** (1Byte): Campo riservato per utilizzi futuri.
- **RES2** (1Byte): Campo riservato per utilizzi futuri.

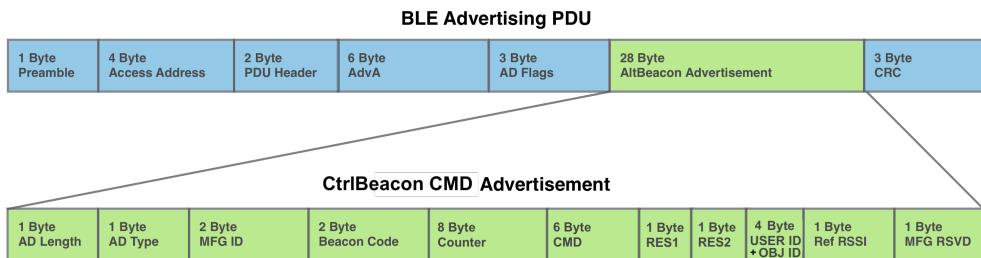


Figura 4.5 Rappresentazione del formato di un pacchetto generico da cifrare usato nel protocollo CtrlBeacon

In aggiunta bisogna notare che, essendo CtrlBeacon definito sugli AltBeacon, a loro volta definiti come payload di messaggi BLE di *advertising*, l'integrità dei dati sarà garantita grazie alla presenza del CRC.

4.2.4 ACK

Il protocollo prevede due differenti tipi di ACK, uno è già stato visto e rappresenta il pacchetto di *acknowledgment* del pacchetto *HelloBroadcast*. Esclusa tale eccezione, in tutti gli altri pacchetti il formato sarà uguale a quello di un generico comando ma avrà dei valori predefiniti. Il pacchetto di ACK sarà caratterizzato dai byte di CMD_Type, CMD_Class, CMD_OPCode tutti identici e pari al valore esadecimale 0xFF. CMD_Parms dovrà avere il valore dello user id dell'utente nel caso in cui sia lui stesso a inviare tale messaggio, alternativamente conterrà l'object id. Il counter conterrà il numero di sequenza del messaggio di cui si intende confermare la ricezione. I valori di USER ID e OBJ ID assumeranno i significati già visti identificando i due terminali della comunicazione (un utente e un oggetto).

Capitolo 5

Interazione tra utenti e smart core

In questo capitolo verranno discusse tutte le operazioni e i meccanismi coinvolti che riguardano l'interazione tra utente e *smart core*. Prima si vuole porre l'attenzione sui servizi e le funzionalità che lo *smart core* offre.

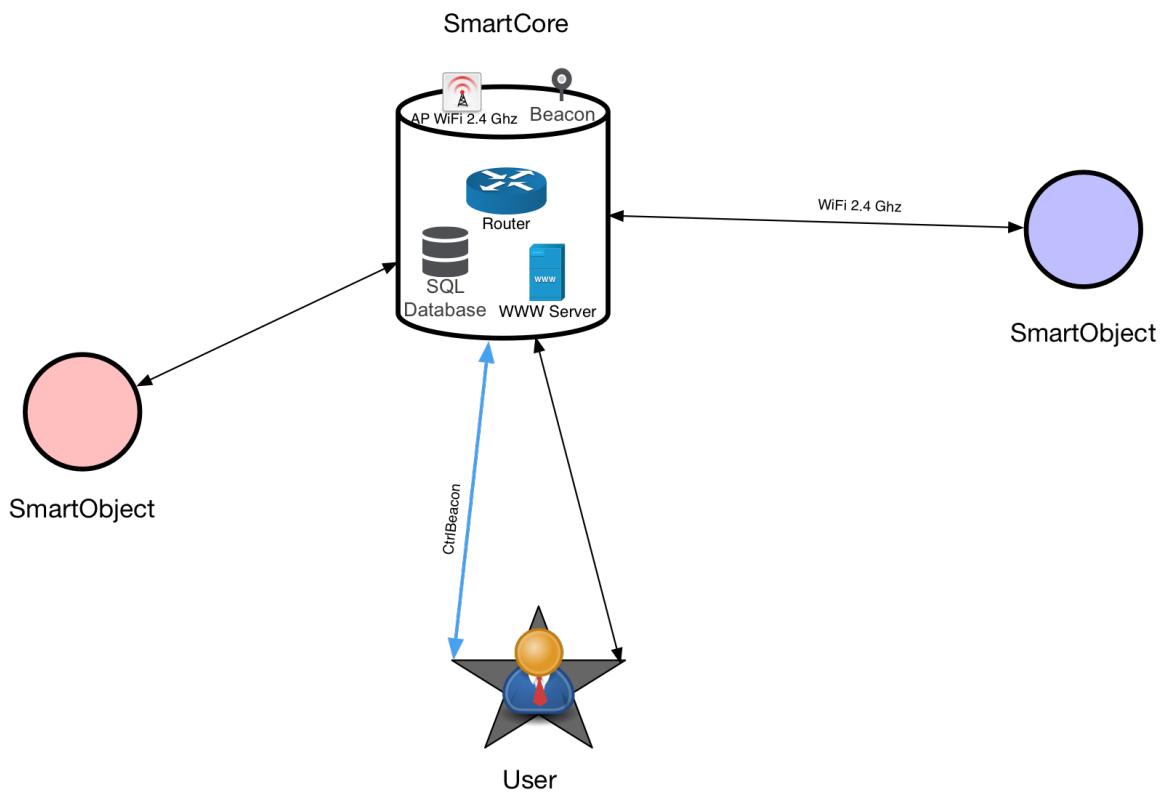


Figura 5.1 Diagramma che mostra l'interazione tra utente e smart core

Come si può notare in Figura 5.1 lo *smart core*, che ricordiamo essere fisicamente rappresentato da una Raspberry Pi 3B, possiede in realtà vari ruoli differenti che verranno

adesso brevemente elencati e successivamente discussi durante la descrizione delle varie fasi. Lo smart core fornirà diversi servizi che possono essere schematizzati nella seguente lista:

1. Server HTTP (Apache)
2. Server SQL (MySQL)
3. Router con server DHCP (Dnsmasq)
4. AP WiFi 2.4Ghz (Hostapd)
5. CtrlBeacon server e Beacon advertiser (libreria sviluppata per l'utilizzo di CtrlBeacon)

I file di configurazione utilizzati per i software sopra elencati sono presenti nell'Appendice A.

5.1 Configurazione

In questa fase verranno descritte tutte le operazioni necessarie per la messa in funzione dello smart core.

Si noti che affinché l'utente, o più correttamente il suo dispositivo, possa essere considerato pronto per l'interazione sarà sufficiente che il proprio smartphone abbia un modulo Bluetooth compatibile con BLE e sia installata un'apposita applicazione. Tale applicazione, la cui realizzazione non rientra negli obiettivi preposti, dovrà avere sia la capacità di comprendere il protocollo CtrlBeacon sia quella di identificare gli oggetti intelligenti e di tradurre comandi vocali in appositi codici identificativi (già discussi nella Sezione4).

5.1.1 Analisi delle dipendenze e dei requisiti

Per completezza si vogliono anzitutto indicare i requisiti e le dipendenze necessarie al funzionamento dello smart core. Si ricorda che in tale realizzazione si è deciso di utilizzare una Raspberry per i motivi già analizzati nella Sezione2.7. In tal senso la Raspberry può essere pensata come un requisito. In realtà l'implementazione in Python del protocollo è compatibile con tutti i dispositivi che possiedono una scheda di rete che fa uso del driver nl80211 (come ad esempio la scheda WLAN integrata di tutti i modelli di Raspberry). In realtà, più esattamente, anche tale vincolo può essere superato con l'installazione di altri driver e la modifica di una sola riga nei file di configurazione. Fatta tale premessa, per semplicità, si supporrà d'ora in avanti di utilizzare una Raspberry per l'implementazione degli oggetti intelligenti. La Raspberry Pi 3B è stata avviata con il sistema operativo scelto (Raspbian) e sono stati installati i seguenti software:

- Apache: utilizzato come server HTTP locale. Non sono state apportate modifiche alla configurazione di default.
- MySQL: necessario per la memorizzazione dei dati descritti e rappresentati nella Sezione 3.3.1.
- BlueZ (stack Bluetooth Linux ufficiale): fornisce, tra le altre cose, un interfaccia HCI utile per interagire con il Controller integrato o esterno della Raspberry.
- Hostapd (Host access point daemon): utilizzato per la configurazione di AP con nome e password variabili.
- Python > 3.4 con i moduli Crypto e PyBluez.

Una nota va fatta circa la possibile difficoltà di installazione della libreria BlueZ, infatti, a causa di un noto bug tutt’oggi presente, risulta necessario prestare attenzione alla compilazione della libreria Boost necessaria per il funzionamento di PyBlueZ. In conclusione si può notare come dipendenze e requisiti siano poco numerosi, questo allo scopo di garantire una maggiore compatibilità e generalità rispetto ai casi d’uso.

5.1.2 Scelte per la configurazione

In tale sottoparagrafo si vogliono indicare le configurazioni specificate accompagnate dalle motivazioni di tali scelte. Successivamente all’installazione dei software e delle dipendenze già viste sarà necessaria l’installazione del software realizzato come implementazione del protocollo *CtrlBeacon*.

Dnsmasq e interfacce di rete Il servizio DHCP è stato abilitato soltanto sull’interfaccia *wlan0*, la stessa che offre il servizio di AP WiFi. Nel caso specifico si è considerata una sottorete costituita dall’intervallo [192.168.0.1, 192.168.0.225] dove il primo indirizzo viene riservato allo *smart core* stesso. Nel file di configurazione è stato specificato un *lease time* di 12 ore. Inoltre, anche se disabilitato di default, sarà possibile connettersi ad un’altra rete via cavo per poter comunicare con lo *smart core* o per poter comunicare dallo *smart core* verso il mondo esterno (come nel caso della necessità di aggiornamento di un software). In questi casi sarà possibile effettuare la connessione avviando il client DHCP lanciando da terminale il seguente comando:

```
sudo dhcpcd eth0
```

Hostapd Il file di configurazione di questo servizio è quello che definisce il comportamento dell’AP WiFi. Sono presenti informazioni circa il canale utilizzato e il tipo di cifratura. Alcuni di questi valori come *SSID* e *password* saranno modificati a runtime dal software che provvederà successivamente a riavviare il servizio per rendere effettive tali modifiche.

5.2 Descrizione del software

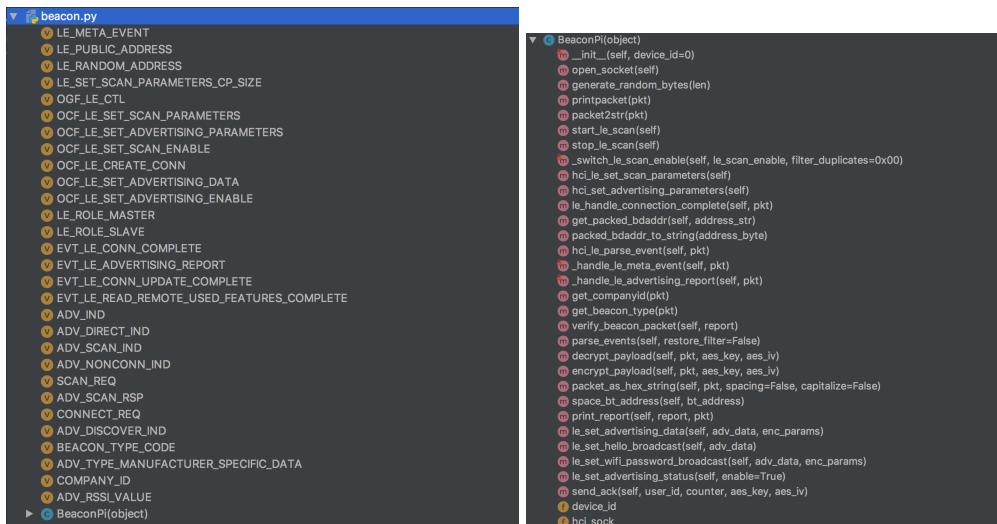
In caso di installazione di uno *smart core* il software sarà costituito da sei differenti classi principali:

1. **BeaconPi**: usata per l’implementazione di CtrlBeacon. I suoi metodi forniscono un’interfaccia al protocollo e consentono di decodificare, inviare e controllare i pacchetti inviati e ricevuti. Viene inoltre utilizzata per la preconfigurazione ovvero la fase di avvio nel sistema durante la quale vengono modificate le impostazioni del controller BLE allo scopo di minimizzare i tempi di comunicazione.
2. **SmartCommands**: classe ausiliaria definita all’interno del modulo *smartObject* allo scopo di identificare le caratteristiche dei comandi (come la classe e il tipo).
3. **SmartObject**: definisce l’insieme delle operazioni che l’oggetto deve svolgere per operare correttamente. Verrà discusso più dettagliatamente nel prossimo capitolo.
4. **SmartCore**: necessaria per il funzionamento dello *smart core*. Viene definita come classe figlia rispetto a *SmartObject* di cui ne eredita di conseguenza tutti i metodi e gli attributi. Vengono inoltre definiti nuovi metodi che fanno uso di quelli offerti dalla classe Hostapd. In effetti uno *smart core* può essere pensato come uno *smart object* particolare, la cui particolarità consiste nella capacità di consentire alcune funzionalità quali la configurazione, la registrazione e l’autenticazione (discusse nella Sezione 3.4).
5. **Hostapd**: costituita da un insieme di metodi atti ad automatizzare le operazioni con il demone di Hostapd. Tra le altre cose si trova l’implementazione del meccanismo necessario per la modifica a runtime della password relativa all’AP WiFi.
6. **AESCipher**: utilizzata per le operazioni di cifratura e decifratura come descritto nella Sezione 2.8.

Di seguito sono indicati i metodi e gli attributi delle varie classi, divisi per i moduli che le contengono.

5.2.1 Modulo beacon

In tale modulo è presente la classe BeaconPi costituita dai metodi e dagli attributi visibili in figura oltre la definizione di alcune costanti necessarie per l'interpretazione e la creazione di pacchetti tramite HCI. Si pensi, ad esempio, alla costante ADV_NONCONN_IND già discussa in precedenza (si veda la Sezione 2.4).



```

beacon.py
▼ LE_META_EVENT
▼ LE_PUBLIC_ADDRESS
▼ LE_RANDOM_ADDRESS
▼ LE_SET_SCAN_PARAMETERS_CP_SIZE
▼ OGF_LE_CTL
▼ OCF_LE_SET_SCAN_PARAMETERS
▼ OCF_LE_SET_ADVERTISING_PARAMETERS
▼ OCF_LE_SET_SCAN_ENABLE
▼ OCF_LE_CREATE_CONN
▼ OCF_LE_SET_ADVERTISING_DATA
▼ OCF_LE_SET_ADVERTISING_ENABLE
▼ LE_ROLE_MASTER
▼ LE_ROLE_SLAVE
▼ EVT_LE_CONN_COMPLETE
▼ EVT_LE_ADVERTISING_REPORT
▼ EVT_LE_CONN_UPDATE_COMPLETE
▼ EVT_LE_READ_REMOTE_USED_FEATURES_COMPLETE
▼ ADV_IND
▼ ADV_DIRECT_IND
▼ ADV_SCAN_IND
▼ ADV_NONCONN_IND
▼ SCAN_REQ
▼ ADV_SCAN_RSP
▼ CONNECT REQ
▼ ADV_DISCOVER_IND
▼ BEACON_TYPE_CODE
▼ ADV_TYPE_MANUFACTURER_SPECIFIC_DATA
▼ COMPANY_ID
▼ ADV_RSSI_VALUE
► BeaconPi(object)

```

```

BeaconPi(object)
  __init__(self, device_id=0)
  open_socket(self)
  generate_random_bytes(len)
  printpacket(pkt)
  packet2str(pkt)
  start_le_scan(self)
  stop_le_scan(self)
  _switch_le_scan(enable(self, le_scan_enable, filter_duplicates=0x00)
  hc1_le_set_scan_parameters(self)
  hc1_set_advertising_parameters(self)
  le_handle_connection_complete(self, pkt)
  get_packed_bdaaddr(self, address_str)
  packed_bdaaddr_to_string(address_byte)
  hc1_le_parse_event(self, pkt)
  _handle_le_meta_event(self, pkt)
  _handle_le_advertising_report(self, pkt)
  get_companyid(pkt)
  get_beacon_type(pkt)
  verify_beacon_packet(self, report)
  parse_events(self, restore_filter=False)
  decrypt_payload(self, pkt, aes_key, aes_iv)
  encrypt_payload(self, pkt, aes_key, aes_iv)
  packer_as_hex_string(self, pkt, spacing=False, capitalize=False)
  space_bt_address(self, bt_address)
  print_report(self, report, pkt)
  le_set_advertising_data(self, adv_data, enc_params)
  le_set_hello_broadcast(self, adv_data)
  le_set_wifi_password_broadcast(self, adv_data, enc_params)
  le_set_advertising_status(self, enable=True)
  send_ack(self, user_id, counter, aes_key, aes_iv)
  device_id
  hc1_sock

```

(a) Contenuto del modulo beacon

(b) Rappresentazione della classe Beacon

La descrizione dei metodi più importanti avverrà nelle sezioni seguenti.

5.2.2 Modulo smartObject

Il modulo *smartObject* fornisce i meccanismi di interazione. Come già anticipato, la classe *SmartObject* è la classe genitore di *SmartCore* e proprio per questo motivo vengono mostrate due classi contenute nel modulo *smartObject*; tuttavia la sua descrizione verrà rimandata al capitolo successivo.

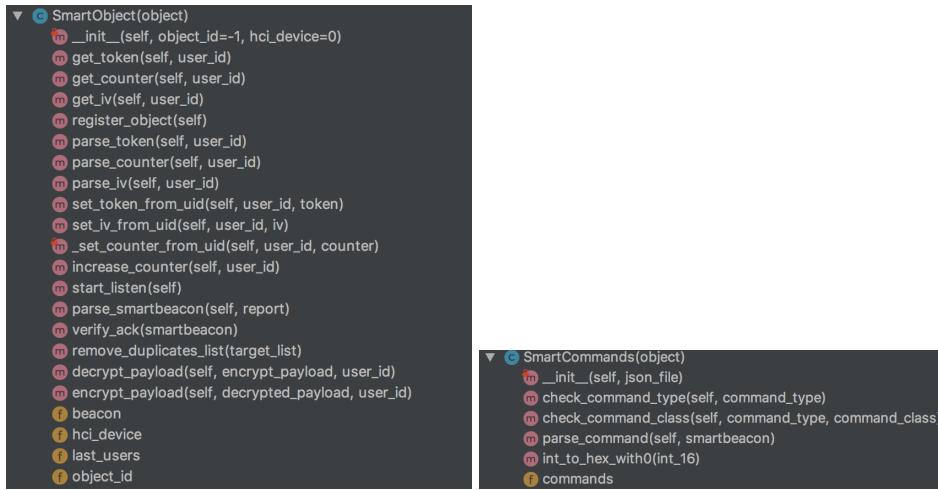


Figura 5.3 Composizione del modulo smartObject

5.2.3 Modulo smartCore

Il modulo *smartCore* contiene la classe omonima che ricordiamo essere definita come figlia della classe *SmartObject* ed è caratterizzata da metodi che consentono le *particolari* operazioni svolte dallo smart core tra cui la creazione dell'AP WiFi con l'ausilio della classe *Hostapd* e l'invio del pacchetto di *HelloBroadcast*. Nella figura sottostante si può notare la struttura del modulo *smartCore* con inclusi, in grigio, i nomi dei metodi e degli attributi ereditati.

```

▼ smartCore.py
  c SmartCore(SmartObject)
    m __init__(self, object_id=-1, hci_device=0, wlan_device="wlan0")
    m new_partial_iv(self)
    m generate_new_psk_str(length=16)
    m new_password(self, restart=True)
    m change_wifi_password(self, psk)
    m send_wifi_password(self, user_id)
    m restore_hellobroadcast(self, adv_time=0.7)
    m send_hellobroadcast(self)
    m check_for_hello_ack(self, report)
    m start_listen(self)
    m parse_smartbeacon(self, report)
    f beacon
    f hostapd
    f iv
    f partial_iv
    f wifi_psk
    f wlan_device
    m get_token(self, user_id)
    m get_counter(self, user_id)
    m get_iv(self, user_id)
    m register_object(self)
    m parse_token(self, user_id)
    m parse_counter(self, user_id)
    m parse_iv(self, user_id)
    m set_token_from_uid(self, user_id, token)
    m set_iv_from_uid(self, user_id, iv)
    m _set_counter_from_uid(self, user_id, counter)
    m increase_counter(self, user_id)
    m verify_ack(smartbeacon)
    m remove_duplicates_list(target_list)
    m decrypt_payload(self, encrypt_payload, user_id)
    m encrypt_payload(self, decrypted_payload, user_id)
    f hci_device
    f last_users
    f object_id
  
```

Figura 5.4 Rappresentazione del modulo smartCore

5.2.4 Moduli ausiliari

Infine troviamo un insieme di moduli che possono essere definiti ausiliari in quanto costituiti da altre classi utilizzate da quelle già viste per comodità e maggiore chiarezza. In particolare ci si riferisce a quei moduli che contengono la classe *AESCipher* e *Hostapd*.

```

▼ hostapdHandler.py
  c HostapdException(Exception)
  c HostapdHandler
    m __init__(self, wlan_device="wlan0", init_ssid="SmartAP", hostapd_conf_path="/etc/hostapd/hostapd.conf")
    m get_restart_min_interval(self)
    m check_hostapd_status(self)
    m hostapd_restart_cmd(self)
    m restart_hostapd(self, wait_time=0)
    m change_wifi_visibility(self, stealth=True, restart=True)
    m change_wifi_password(self, psk, restart=True)
    m change_wifi_ssid(self, ssid, restart=True)
    f hostapd_conf_path
    f last_restart_time
    f waiting_restart
    f wlan_device
  
```

```

▼ AESCipher.py
  c AESCipher
    m __init__(self, key)
    m set_iv(self, iv)
    m pad(self, s)
    m unpad(self, s)
    m encrypt(self, raw, padding=False)
    m decrypt(self, enc, padding=False)
    f iv
    f key
  
```

(a) Contenuto del modulo hostapd
(b) Rappresentazione del contenuto del modulo AESCipher

Figura 5.5 Moduli ausiliari

In Figura 5.5 vengono mostrati gli attributi e i metodi coinvolti.

5.3 Sequenza delle operazioni

Per descrivere le varie fasi di funzionamento del sistema e le interazioni che coinvolgono tutti i dispositivi si vuole seguire la reale sequenza temporale. A tale scopo si illustra la lista dei dispositivi e delle rispettive operazioni che verranno eseguite per consentire l'interazione tra utente e *smart core*. A seguire, la lista delle operazioni svolte da entrambi i dispositivi.

Lo *smart core* dovrà, in ordine, offrire le seguenti funzionalità:

1. Invio del pacchetto di *HelloBroadcast*.
2. Attesa dell'ACK relativo al messaggio di *HelloBroadcast* e creazione dell'AP con la nuova password.
3. Fornire un nuovo IV assieme ad altre informazioni ed eventualmente nuove versioni per la rete neurale.

L'utente dovrà, in ordine, eseguire le seguenti azioni in risposta allo *smart core*:

1. Stare in ascolto per il pacchetto di *HelloBroadcast* ed alla ricezione inviare il relativo ACK.
2. Connetersi all'Access Point generato dallo smart core.
3. Scaricare le informazioni fornite dallo smart core e successivamente effettuare la disconnessione WiFi. Fermare l'ascolto dei pacchetti broadcast per un periodo limitato.

La coppia di azioni elencate sopra sarà analizzata più nel dettaglio consentendo di distinguere tre differenti momenti temporali.

5.3.1 Attesa e risposta

La prima fase consiste nella reciproca segnalazione della propria presenza messa in atto da entrambi i dispositivi tramite l'invio dei due differenti pacchetti: l'*HelloBroadcast* e l'*HelloBroadcastACK*. Nella figura seguente si mostra il *sequence diagram* relativo a tale fase.

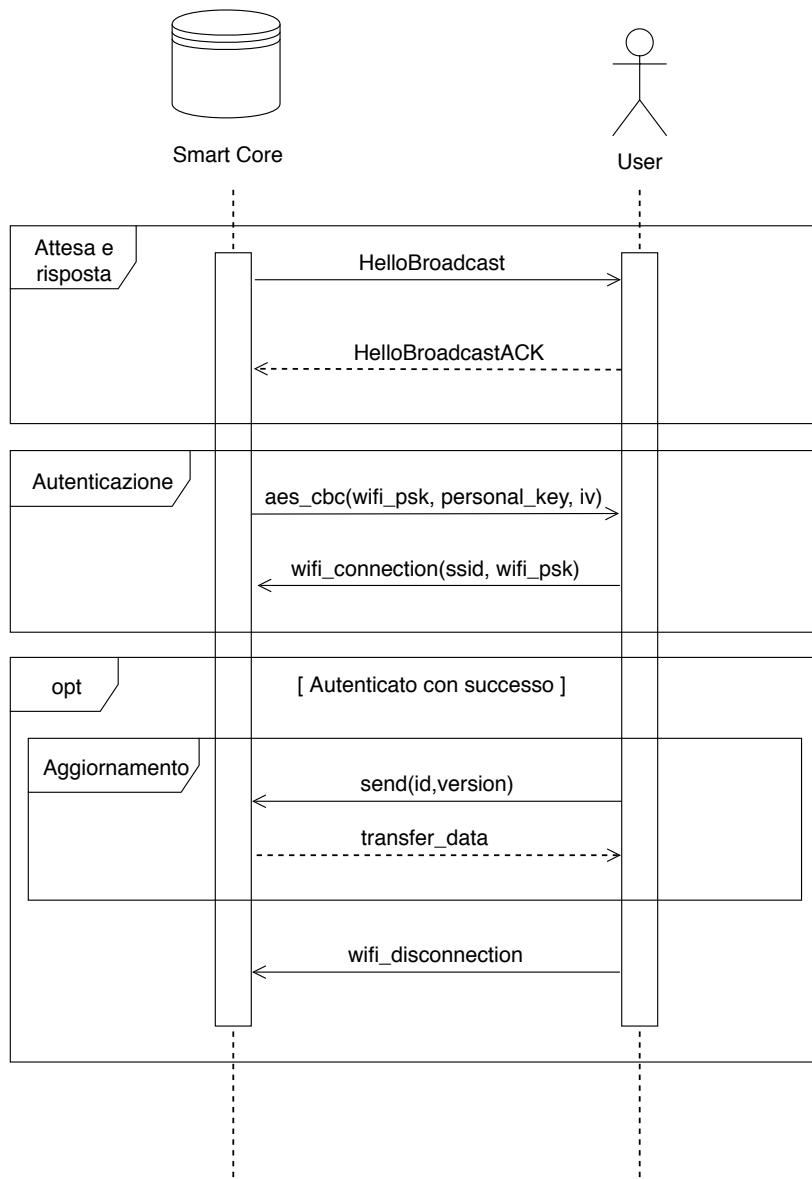


Figura 5.6 Sequence diagram relativo all'interazione tra utente e smart core

Si analizza adesso il comportamento delle due entità coinvolte.

Smart core Lo *smart core* si occuperà di inviare continuamente i pacchetti di *HelloBroadcast* al fine di segnalare la propria presenza e fornire l'IV da utilizzare durante la procedura di *handshake*. L'utilizzo della classe fornita risulta piuttosto semplice, infatti il codice python necessario per porre lo *smartcore* in stato di funzionamento è il seguente:

```
from smartCore import SmartCore
sc = SmartCore(0)
sc.start_listen()
```

Analizziamo il metodo che viene eseguito alla creazione dell'oggetto *SmartCore*.

```
def __init__(self, object_id=0, hci_device=0, wlan_device="wlan0"):
    """An extended SmartObject class used to handle a smart core

    Keywords argument:
    hci_device - number of the hci to use (ex. 0 or 1)[INT]
    iv - the iv that will be used for the next communication
    wifi_psk - the wifi password used by the current AP
        (bytes, utf-8 encoded)
    wlan_device - the device used for the current wlan AP
        (ex. wlan0, mon0)

    """
    super().__init__(object_id, hci_device)
    # Generate a new iv and wifi_password
    self.hostapd = HostapdHandler(wlan_device)
    self.wlan_device = wlan_device
    self.new_partial_iv()
    self.new_password(restart=False)
    self.hostapd.restart_hostapd()
    #self.change_wifi_visibility(stealth=True) # Hidden AP
```

Si nota che viene creata un'istanza dell'oggetto genitore ed inoltre che il valore di default per *object_id* sia 0. Infatti per convenzione assumiamo che lo *smart core*, in quanto *smart object* sia identificato da tale codice. Successivamente viene generato un nuovo *partial_iv* ed una nuova password per il WiFi. Al termine verrà riavviato *hostapd*. A proposito della classe *Hostapd* bisogna precisare che il metodo *restart_hostapd()* possiede un meccanismo di blocco che non consente riavvii ad una frequenza maggiore di quella definita (di default) da Linux; questo viene fatto allo scopo di evitare situazioni di stallo del servizio di *Hostapd*. Un'altra porzione di codice interessante e relativa a tale fase è data dal metodo *start_listen()* di cui si mostra la parziale composizione a titolo esemplificativo.

```
def start_listen(self):
    beacon = BeaconPi(self.hci_device)
    self.beacon = beacon
    sock = beacon.open_socket()
    beacon.hci_le_set_scan_parameters()
    beacon.start_le_scan()
    beacon.hci_set_advertising_parameters()
    # Start to advertise
    beacon.le_set_advertising_status(enable=True)
    self.send_hellobroadcast()
    smart_command_handler = SmartCommands()
    sending_ack = False
    while True:
        smartbeacon_list = beacon.parse_events()
        for smartbeacon in smartbeacon_list:
            if smartbeacon['minor'] == self.object_id:
                clear_user_id = smartbeacon['major']
                if self.parse_smartbeacon(smartbeacon):
                    if not smartbeacon['smartbeacon']['is_ack']:
                        ...

```

Rispetto all'implementazione mostrata è possibile eseguire diverse considerazioni. In prima istanza si nota l'apertura del socket (HCI) che consente successivamente la chiamata ai metodi `hci_le_set_scan_parameters()`, `start_le_scan()`, `hci_set_advertising_parameters()` ed infine `le_set_advertising_status(enable=True)` che avvia la trasmissione in broadcast dei beacon di *HelloBroadcast*. Successivamente si ha un ciclo infinito in cui viene espressa la capacità di ascolto continua dello *smart core* che sarà sempre in attesa di un pacchetto (sia esso un comando o un *HelloBroadcast*). Si noti inoltre che il codice è stato tagliato in quanto il contenuto mancante verrà discusso nel capitolo seguente. Del resto, come è facile intuire, la porzione di codice assente, indicata dal simbolo dei tre punti (...), implementa la capacità di esecuzione dei comandi tipica dello *Smart Object*.

Per comprendere la differenza con lo *smartObject* viene mostrata la porzione di codice di interesse del metodo `parse_smartbeacon()` e `check_for_hello_ack()` appartenente alla classe in questione.

```

def parse_smartbeacon(self, report):
    """Unpack all field and return True if packet is valid
    at the end update report dict
    """
    if self.check_for_hello_ack(report):
        report['smartbeacon'] = {'is_ack': True}
        return True
    ...

def check_for_hello_ack(self, report):
    """Check and do some action if an HelloBroadcast is received

    Note: the format of the HelloBroadcast ACK is different
    like the HelloBroadcast packet itself.
    """
    is_ack = False
    clear_report = report["payload_binary"]
    #Skip manufacturerID and BEAC identifier -> 0:6
    cmd_id, = struct.unpack(">I", clear_report[6:10])
    recv_partial_iv = report["payload_binary"][10:22]
    user_id = report['major']
    HELLO_BROADCAST_ACK_CMD_ID = 0xFFFFFFFF
    if cmd_id == HELLO_BROADCAST_ACK_CMD_ID:
        # Compare the partial IV (12/16 bytes).
        # Note: the user_id (12:14) is different now
        if recv_partial_iv == self.partial_iv: # It's a real HB ACK
            self.send_wifi_password(user_id)
            self.new_password()
            self.new_partial_iv()
            # After the advertising time, need to rebroadcast the HB
            # Note, call this AFTER changing the partial_iv
            self.restore_hellobroadcast()
            is_ack = True
    return is_ack

```

La prima operazione effettuata dal metodo `parse_smartbeacon()` consiste nel verificare se il pacchetto sia oppure no un *HelloBroadcastACK*. Se non lo fosse si proseguirà ad effettuare i

controlli circa l’eventuale validità del comando e per questo motivo le istruzioni successive non sono presenti nella porzione di codice presentata. Se il pacchetto ricevuto sembra un *HelloBroadcastACK* (e quindi possiede tutti i campi relativi al comando posti a 0xFF) verrà confrontato il *partial_IV* ricevuto con quello inviato dallo *smart core* nel pacchetto di HB. Se tale confronto ha esito positivo lo *smart core* provvederà a cambiare il pacchetto di *WifiPassword* all’utente. Solo successivamente verrà generata una nuova password WiFi e un nuovo *partial_IV* da usare per la prossima connessione. Infine verrà aggiornato l’invio dei pacchetti di HB con i nuovi valori generati.

Utente Al contempo l’utente, quando entra in uno *smart environment*, deve rispondere al pacchetto ricevuto inviando l’*HelloBroadcastACK* che segnala allo *smart core* la presenza di un nuovo utente nell’ambiente. Bisogna notare che il meccanismo di *HelloBroadcastACK* segue la stessa sequenza di stati dell’ACK generico che verrà mostrata nel capitolo seguente.

5.3.2 Autenticazione

In questa fase verranno poste le basi per la successiva connessione WiFi e verranno messi in atto i meccanismi necessari per l’autenticazione. Se l’*HelloBroadcastACK* non dovesse essere stato ricevuto dallo *smart core* o il pacchetto *WifiPassword* dall’utente, le operazioni fino adesso descritte verranno ripetute. Supponiamo quindi di giungere nello stato caratterizzato dalla ricezione di entrambi questi pacchetti. Sotto tali ipotesi l’utente avrà ricevuto la password dell’AP in forma cifrata come descritto nella Sezione 4.2.2. Inoltre si può affermare che solo il legittimo utente, nota la propria *personal_key* (chiave utilizzata per la cifratura AES), potrà decifrare la password corretta ed effettuare l’autenticazione connettendosi all’AP WiFi. Se l’utente è connesso alla rete WLAN allora sarà sicuramente stato autenticato e sarà pronto per le operazioni di aggiornamento dei propri dati.

5.3.3 Aggiornamento

Questa fase è composta dall’insieme delle operazioni necessarie per l’aggiornamento dei dati necessari all’utente. I dati coinvolti sono i seguenti:

- Un nuovo IV necessario per la seguente comunicazione. Questo avviene in quanto per aumentare la velocità di connessione si è ridotta l’entropia del vettore di inizializzazione (da 16 a 12 byte ricordando la definizione del *partial_iv*).
- Il counter da utilizzare per l’eventuale invio di comandi.

- I dati necessari per la rete neurale. Potrebbe essere stato aggiunto un nuovo oggetto o un nuovo comando allo *smart environment* e quindi l’utente non sarebbe in grado né di riconoscere i nuovi comandi né i nuovi oggetti presenti nell’ambiente.

Bisogna notare che i dati che dovranno essere forniti all’utente saranno esclusivamente quelli che sono previsti per il gruppo (*usergroup*) al quale appartiene l’utente. Un’ulteriore nota interessante può essere fatta in relazione a tale campo che, come si è visto sino adesso, non viene mai inviato dall’utente che, nel caso fosse malintenzionato, potrebbe fingersi appartenente a un gruppo con meno restrizioni. Piuttosto, il gruppo che identifica i permessi dell’utente sarà sempre ottenuto a partire dallo *user_id* che non può essere alterato pena la non comprensione dei pacchetti (grazie al meccanismo di cifratura).

5.3.4 Termine della connessione

Infine l’utente dovrà terminare, il prima possibile, la connessione all’AP. Del resto la connessione verrà chiusa, allo scadere di un timeout, dallo stesso *smart core*. Questo avviene per evitare una congestione nelle fasi di *autenticazione* e *aggiornamento*, dove un solo utente per volta può essere servito.

Capitolo 6

Interazione tra utenti e smart object

In questo capitolo si intendono descrivere le entità e le fasi necessarie al funzionamento di uno *smart object*. Per una migliore comprensione il capitolo sarà diviso in sezioni che rappresentano e descrivono le varie fasi.

6.1 Configurazione

Gli *smart object* saranno connessi, possibilmente tramite ethernet, alla LAN gestita dallo *smart core*. Tramite tale connessione sarà possibile richiedere informazioni utili sugli utenti, i comandi e i permessi. Gli *smart object* dovranno essere forniti di una propria configurazione che indichi i comandi che sono in grado di eseguire. Tali comandi verranno poi tradotti in una coppia di file in formato json o, più correttamente, una coppia per ogni usergroup. Infine ogni *smart object* avrà uno script python contenente l'implementazione delle funzioni che esso deve svolgere. Un esempio della struttura di tale script viene fornito nell'Appendice A. Nell'Appendice B si possono trovare le istruzioni da utilizzare per l'installazione e la creazione di un'immagine del disco(come potrebbe essere un aggiornamento software).

6.1.1 Analisi delle dipendenze e dei requisiti

I requisiti degli *smart object* sono meno stringenti di quelli visti per lo *smart core* nel capitolo precedente, infatti esso dovrà svolgere la sola funzione di *CtrlBeacon server*. In altri termini il suo compito sarà quello di attendere indefinitamente *advertisement BLE*, filtrarli prelevando solo i *CtrlBeacon* e, in caso di pacchetto e autorizzazioni valide, eseguire il comando. Nel caso specifico la Raspberry Pi 0W è stata avviata con il sistema operativo scelto (Raspbian) e con installati i seguenti software:

- BlueZ (stack Bluetooth Linux ufficiale): fornisce, tra le altre cose, un interfaccia HCI utile per interagire con il Controller integrato o esterno della Raspberry
- Python > 3.4 con i moduli Crypto e PyBluez

6.1.2 Scelte per la configurazione

Nessuna particolare configurazione è stata effettuata sulla Raspberry Pi 0W se non quella provvisoria di aumentare il file di swap per eseguire la compilazione delle dipendenze necessarie. A tal proposito si riporta un tempo di compilazione molto elevato. Per questo motivo si consiglia di utilizzare un approccio di tipo *cross compiling*.

6.2 Descrizione del software

Nel caso di uno *smart object* il software sarà costituito da quattro differenti classi principali:

1. **BeaconPi**: usata per l'implementazione di *CtrlBeacon*. I suoi metodi forniscono un'interfaccia al protocollo e consentono di decodificare, inviare e controllare i pacchetti inviati e ricevuti. Viene inoltre utilizzata per la preconfigurazione ovvero la fase di avvio nel sistema durante la quale vengono modificate le impostazioni del controller BLE allo scopo di minimizzare i tempi di comunicazione.
2. **SmartCommands**: classe ausiliaria definita all'interno del modulo *smartObject* allo scopo di identificare le caratteristiche dei comandi (come la classe e il tipo).
3. **SmartObject**: definisce l'insieme delle operazioni che l'oggetto deve svolgere per operare correttamente. Questa classe rappresenta il cuore dello *smart object*.
4. **AESCipher**: utilizzata per le operazioni di cifratura e decifratura come descritto nella Sezione 2.8

I metodi e gli attributi delle varie classi, divisi per i moduli che le contengono sono già stati mostrati nella Sezione 5.2. Si riporta per facilità di consultazione solo il modulo di maggiore interesse.

6.2.1 Modulo smart object

Il modulo *smartObject* fornisce i meccanismi di interazione garantendo al contempo affidabilità e sicurezza.



Figura 6.1 Composizione del modulo smartObject

Iniziamo considerando il codice necessario per la creazione di un oggetto *smart object*:

```

from smartObject import SmartObject
so = SmartObject()
so.start_listen()

```

Analizziamo quindi cosa avviene nel metodo `__init__()` (eseguito alla creazione di un oggetto) e di cui si riporta il contenuto:

```

def __init__(self, object_id=-1, hci_device=0):

    if os.getuid() != 0:
        raise NotSudo("Elevated privileges are required.")
    if object_id == -1: # Check configuration file
        object_id = self.register_object
    self.object_id = object_id
    self.last_users = UpdateOrderedDict()
    self.hci_device = hci_device

```

Per prima cosa si noti che l'esecuzione richiede l'elevazione dei privilegi pena la terminazione dello script con eccezione *NotSudo*. In effetti i privilegi elevati servono per comunicare mediante il socket HCI. Il secondo if rappresenta la condizione per cui lo *smart object* deve registrarsi sulla rete affinché venga fornito un *object_id*.

Tuttavia il server *CtrlBeacon* viene effettivamente lanciato soltanto quando viene chiamato il metodo `start_listen()`. Si riporta a tale scopo la porzione di codice che vede coinvolti i meccanismi di riconoscimento e filtraggio dei pacchetti:

```

def start_listen(self):
    beacon = BeaconPi(self.hci_device)
    self.beacon = beacon
    sock = beacon.open_socket()
    beacon.hci_le_set_scan_parameters()
    beacon.start_le_scan()
    beacon.hci_set_advertising_parameters()
    beacon.le_set_advertising_status(enable=True) # Start adv.
    smart_command_handler = SmartCommands()
    sending_ack = False
    while True:
        smartbeacon_list = beacon.parse_events()
        for smartbeacon in smartbeacon_list:
            if smartbeacon['minor'] == self.object_id:
                clear_user_id = smartbeacon['major']
                if self.parse_smartbeacon(smartbeacon):
                    if not smartbeacon['smartbeacon']['is_ack']:
                        # Get encryption data of the user
                        aes_key = self.get_token(clear_user_id)
                        aes_iv = self.get_iv(clear_user_id)
                        beacon.send_ack(clear_user_id,
                                         self.get_counter(clear_user_id), aes_key, aes_iv)
                        sending_ack = True
                    smart_command_handler.parse_command(smartbeacon['smartbeacon'])

def parse_smartbeacon(self, report):
    """Unpack all field and return True if packet is valid, and update report dict"""

    if len(report["payload_encrypted_data"]) == 16:
        report["decrypted_payload"] = \
            self.decrypt_payload(report["payload_encrypted_data"], report['major'])
        dec_payload = report["decrypted_payload"]
        counter, = struct.unpack(">Q", dec_payload[0:8])
        cmd_type, cmd_class, cmd_opcode, cmd_params, cmd_bitmask = \
            struct.unpack(">BBBhB", dec_payload[8:14])
        res1, res2 = struct.unpack(">BB", dec_payload[14:16])
        report['smartbeacon'] = {'counter': counter, 'cmd_type': cmd_type,
                                 'cmd_class': cmd_class, 'cmd_bitmask': cmd_bitmask,
                                 'cmd_opcode': cmd_opcode, 'cmd_params': cmd_params,
                                 'res1': res1, 'res2': res2,
                                 'is_ack': False, 'user_id': report['major']}
        if not self.verify_ack(report['smartbeacon']): # is an ack?
            user_id = report['smartbeacon']['user_id']
            counter_received = report['smartbeacon']['counter']
            if counter_received == self.get_counter(user_id):
                self.increase_counter(user_id) # New packet
            elif report['smartbeacon']['counter'] == \
                self.get_counter(report['smartbeacon']['user_id']) - 1:
                return False # Duplicated packet
            else: # Counter not synchronized
                return False
        else:
            report['smartbeacon']['is_ack'] = True
    return True

```

Il metodo *start_listen()* inizia con la configurazione del Controller BLE per poi entrare in un ciclo senza terminazione. Infatti lo *smart object* sarà sempre in ascolto di pacchetti a lui indirizzati (tramite il campo *object_id* dei *CtrlBeacon*) che dovranno essere filtrati e validati. Nel caso in cui tali verifiche abbiano buon fine si effettueranno differenti operazioni a seconda che sia un ACK oppure no. Nel caso in cui il pacchetto non fosse di ACK esso verrà decifrato e verranno effettuate ulteriori due validazioni:

- Il controllo della validità del pacchetto e di conseguenza della legittimità dell’utente.
- Il controllo della esistenza del comando e quindi della validità dei permessi dell’utente.

Il meccanismo di ACK verrà descritto nelle sezioni a seguire. Proseguiamo la trattazione tornando a descrivere gli altri metodi presenti. In particolare nella classe *SmartObject* si può notare la presenza di metodi omonimi a meno del prefisso *get* o *parse* e questo viene giustificato dall’utilizzo di un meccanismo di *caching*. In altri termini i metodi che iniziano con *parse* (come ad esempio *parse_iv()*) effettuano una interrogazione allo *smart core* mentre quelli che possiedono il prefisso *get* cercano prima l’eventuale presenza dell’informazione nella cache e, solo in caso di esito negativo, effettuano la chiamata al corrispettivo metodo *parse*. L’utilizzo di un sistema di *caching* consente di evitare di interrogare frequentemente lo *smart core* per ottenere informazioni che erano state date in precedenza o informazioni ricavabili a partire da quelle ottenute in precedenza (basti pensare al counter). Si noti che il limite massimo dei contenuti nella cache può essere variato.

6.3 Sequenza delle operazioni

Come già anticipato lo *smart object* dovrà effettuare solo un’azione definita di *attesa ed esecuzione*. Nella figura seguente si mostra il *sequence diagram* relativo a tale fase.

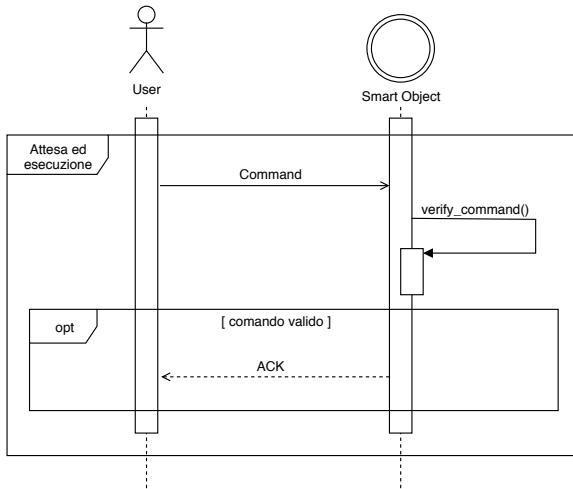


Figura 6.2 Sequence diagram relativo all’interazione tra utente e smart object

6.3.1 Attesa ed esecuzione

Questa fase descrive totalmente il funzionamento dello *smart object* che si trova sempre in uno stato di attesa e pronto per l’esecuzione di un comando. La struttura dei comandi e le loro potenzialità sono state descritte nella Sezione 4. In questo paragrafo si intende descrivere quali sono i controlli e le operazioni che consentono il funzionamento dello *smart object*. Quando arriva un nuovo pacchetto avverranno, in ordine, le seguenti operazioni:

1. Controllo di integrità e altri generici filtri hardware
2. Controlli sulla validità del pacchetto e verifica del destinatario
3. Controlli sulla validità della sintassi e sui reali permessi dell’utente

Si analizza adesso il meccanismo di ACK utile per comprendere cosa avviene ad un più basso livello.

Meccanismo di ACK

Uno dei problemi più interessanti nell’ambito delle comunicazioni è quello che si pone nel caso in cui si volesse trasmettere un’informazione (e quindi un pacchetto) su un canale

inaffidabile avendo la certezza della sua ricezione. Questo problema può essere risolto introducendo, tra le altre cose, un messaggio di ACK (attestazione) che conferma la ricezione. Tuttavia cosa accade se tale messaggio va perso o è corrotto? Il problema della corruzione può essere tralasciato in quanto, come già specificato, gli *advertisement* posseggono un CRC che consente di scartare i pacchetti con errori e che garantisce l'integrità.

Di seguito si riportano le FSM del ricevitore e del trasmettitore allo scopo di comprendere il meccanismo e la sequenza degli stati assunti dalle due entità durante l'invio o l'attesa di un pacchetto. Per una più semplice comprensione il problema verrà generalizzato con le seguenti ipotesi:

- Si fa uso di un canale inaffidabile nel quale è possibile inviare un pacchetto con una generica chiamata $u_send(packet)$. Si noti che la u viene associata ad *unreliable*.
- Quando un pacchetto dovrà essere inviato si avrà un evento di tipo *send call* che possiamo immaginare come una chiamata ad un API. Più semplicemente si può immaginare di sapere quando si vuole inviare un pacchetto tramite il protocollo affidabile definito sopra un canale inaffidabile.

Possiamo analizzare cosa avviene da entrambi i lati della comunicazione nei paragrafi seguenti.

Ricevitore Il ricevitore sarà sempre in uno stato di attesa di un nuovo messaggio ma alla ricezione di un pacchetto dovrà distinguere se esso è:

- *Valido*: se rispetta la semantica e la sintassi definita.
- *Permesso*: se l'utente è davvero chi dice di essere e possiede i privilegi necessari.

Soltanto se tali requisiti sono soddisfatti sarà possibile procedere con la sua interpretazione.

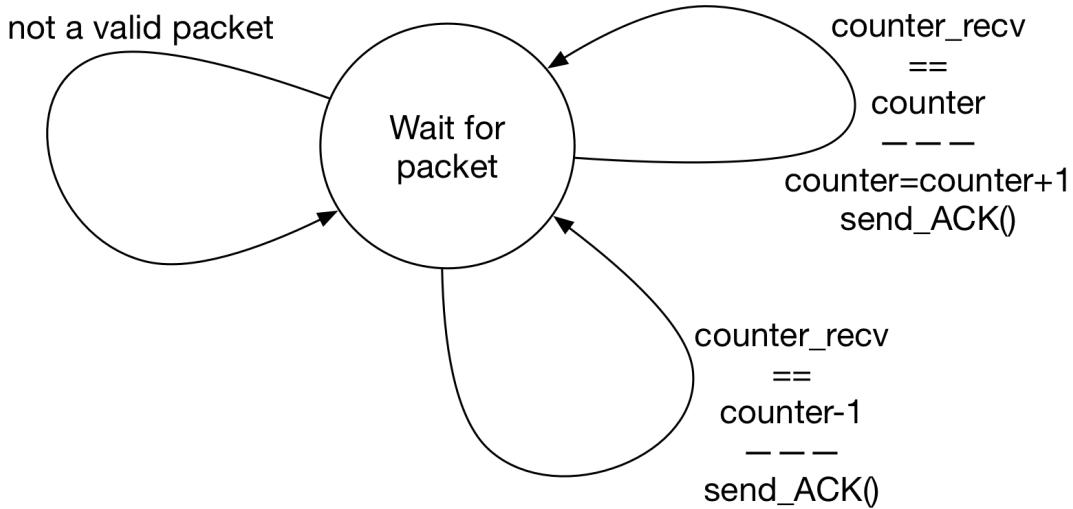


Figura 6.3 Macchina a stati finiti di un ricevitore in ascolto su un canale inaffidabile

Nella Figura 6.3 si nota come esiste un solo stato definito come stato di attesa di un nuovo pacchetto. In tale circostanza infatti si possono distinguere tre differenti casi (rappresentati dalle tre rispettive frecce in figura) che identificano tre possibili tipi di pacchetto:

1. *Non valido*: il pacchetto viene scartato.
2. *Valido e ripetuto*: il pacchetto è valido ma possiede il numero di sequenza precedente, questo significa che rappresenta un *duplicato* ovvero che è già stato ricevuto. Nel dubbio che ancora il trasmettitore non sappia della sua ricezione sarà inviato nuovamente l'ACK.
3. *Valido e nuovo*: il pacchetto possiede le caratteristiche per essere definito valido e possiede un valore del counter uguale a quello atteso ovvero uguale al counter relativo all'utente in questione. In tal caso il counter sarà incrementato e verrà inviato l'ACK relativo.

Questo approccio consente di garantire le condizioni desiderate a fronte di tutte le possibili combinazioni stato-evento. Queste tre condizioni possono essere ritrovate anche nell'ultima porzione di codice appartenente al metodo `parse_smartbeacon()`.

Trasmettitore Come si è discusso in precedenza bisogna considerare l'eventualità in cui il pacchetto di ACK non giunga a destinazione. Una soluzione può essere quella per cui

se, dopo l'invio di un messaggio, non viene ricevuto un ACK (e quindi una conferma di ricezione) entro un tempo ragionevolmente lungo, si considererà il pacchetto come non ricevuto dal mittente e il pacchetto verrà rinviato. Si noti come questo sistema introduce dei pacchetti duplicati che verranno scartati dal ricevitore. Il trasmettitore dovrà, per questo motivo, implementare un meccanismo di timeout. Quanto detto finora viene descritto dalla macchina a stati finiti in figura sottostante.

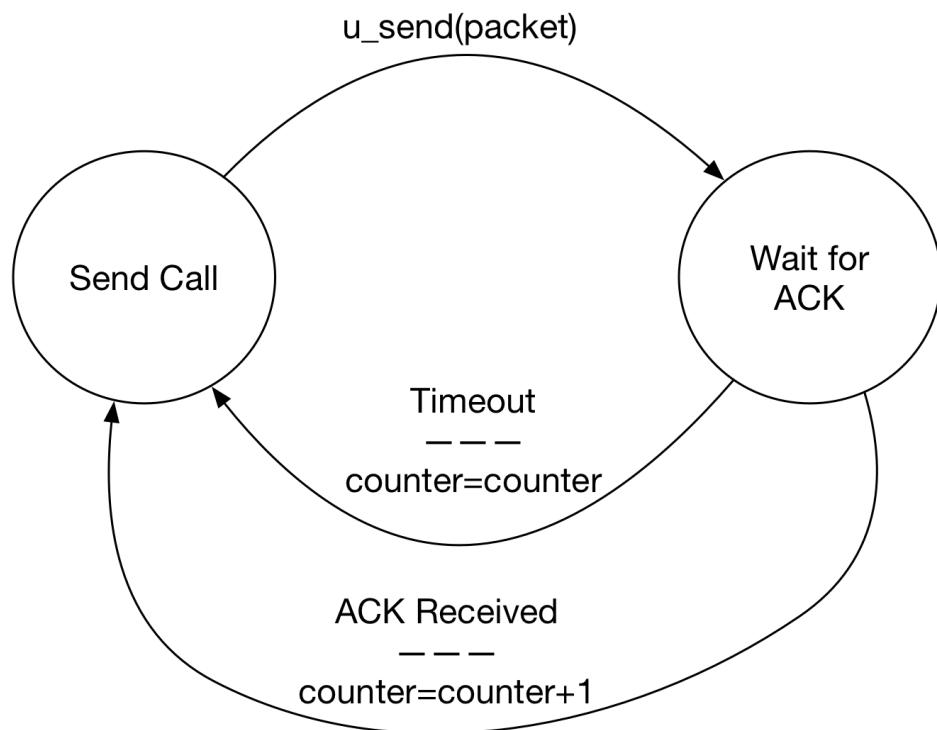


Figura 6.4 Macchina a stati finiti di un trasmettitore che invia pacchetti su un canale inaffidabile

Va notato che l'introduzione del timeout necessita di qualche considerazione più approfondita. La presenza del timeout aumenta infatti i ritardi di comunicazione. In realtà si è giunti a un buon compromesso ottenendo dei tempi di risposta dell'ordine dei decimi di secondo e quindi sicuramente soddisfacenti.

Capitolo 7

Conclusione

In tale elaborato si è discussa la progettazione e la possibile implementazione di uno *smart environment*, ovvero di un ambiente costituito da oggetti *intelligenti* che acquisiscono la capacità di *interazione* con il mondo esterno.

7.1 Analisi dei requisiti e degli obiettivi raggiunti

Nell'introduzione sono stati definiti alcuni vincoli da rispettare e obiettivi da raggiungere. In questo paragrafo verrà descritto il prodotto finale usando come criterio il soddisfacimento dei requisiti imposti e la bontà del prodotto se paragonato alle alternative presenti.

Offline Uno dei requisiti era quello di ottenere un sistema che funzionasse senza accesso alla rete e utilizzando un approccio che non necessitasse di fiducia verso il fornitore del servizio. In effetti questo requisito è stato efficacemente superato mediante la combinazione della tecnologia BLE e delle reti neurali che, eseguite sul dispositivo dell'utente, effettuano il riconoscimento e quindi l'associazione sia tra voce e codifica del comando, sia tra oggetto inquadrato e l'identificativo dell'oggetto. La natura locale del sistema consente di ottenere sia una maggiore sicurezza sia una maggiore privacy rispetto ad altre possibili soluzioni, in cui tutti i dati vengono elaborati da un server online appartenente a terzi.

Sicurezza Il sistema finora discusso rappresenta in effetti un collegamento tra mondo reale e mondo virtuale ed è per questo motivo che risulta necessario imporre un elevato standard di sicurezza che eviti che utenti non autorizzati possano creare disservizi, ascoltare le azioni eseguite dall'utente o ancora eseguirne delle altre non autorizzate. Una lista più dettagliata dei requisiti di sicurezza è stata fornita nella Sezione 3.2. In particolare è stata stilata una lista,

che viene adesso riportata completa dei relativi meccanismi implementati utili a contrastare i possibili vettori di attacchi considerati.

1. Autenticazione: garantisce che l'entità comunicante è chi afferma di essere. La fase di autenticazione è svolta in fase di connessione all'AP WiFi; soltanto un utente provvisto della sua chiave personale potrà decifrare il pacchetto contenente la password necessaria per la connessione. Si noti che per poter interagire con lo *smart environment* sarà necessario passare sempre, almeno una volta, per lo *smart core* che effettuerà l'autenticazione e l'aggiornamento dei dati in possesso dell'utente.
2. Controllo degli accessi: garantisce che le risorse vengano utilizzate da chi ne ha il permesso. Questo meccanismo trova la sua implementazione nel protocollo stesso. La divisione in *gruppi* degli utenti consente una gestione dei permessi atta a garantire il controllo dei permessi. Si noti inoltre che qualora un utente malintenzionato volesse attaccare il sistema saltando la fase di autenticazione, ad esempio mediante un attacco *replay*, il pacchetto ripetuto verrebbe scartato grazie al numero di sequenza.
3. Confidenzialità: protezione della riservatezza dei dati. La confidenzialità viene garantita mediante il sistema di cifratura a chiave simmetrica introdotto. Infatti l'utilizzo di AES CBC con chiave a 256 bit va nettamente oltre i limiti di attacco a forza bruta che è possibile effettuare. Si noti che la sicurezza si basa su un segreto condiviso da entrambe le parti che è stato definito come *chiave personale*. Si noti che l'ottenimento di tale chiave a partire da un pacchetto cifrato con la modalità suddetta richiederebbe, utilizzando sistemi (oggi ancora non esistenti) in grado di verificare 10^{18} chiavi per secondo, un tempo pari a $3 \cdot 10^{51}$ anni per analizzare l'intero insieme delle chiavi.
4. Integrità: garanzia che i messaggi non siano alterati. Questo meccanismo si trova implementato nello standard Bluetooth che prevede un CRC per ogni *advertisement* inviato. I pacchetti contenenti errori verranno scartati direttamente dal *controller*.
5. Non ripudiabilità: garanzia contro la possibilità che una delle due entità coinvolte nella comunicazione neghi di aver partecipato alla comunicazione. Stando a quanto visto fino adesso risulta chiaro che se un utente esegue una particolare azione esso viene identificato univocamente dalla coppia di dati *user_id* e *personal_key*. In tal senso l'unica violazione a tale principio si ha solo nel caso di furto di dati (ad esempio se il dispositivo dell'utente venisse rubato). Si noti che anche in tale scenario, avvertendo per tempo l'amministratore della rete, sarà possibile eliminare la *chiave personale* dell'utente dal sistema e di conseguenza bloccare l'utente che non sarà più in grado di effettuare alcuna operazione.

Si può concludere il paragrafo affermando di aver soddisfatto pienamente i requisiti di sicurezza imposti.

7.2 Possibili miglioramenti ed estensioni future

Un’ulteriore sezione va dedicata a quelli che potrebbero essere dei miglioramenti o delle estensioni dell’applicazione da implementare in futuro. Per quanto riguarda il protocollo sarebbe interessante utilizzare uno dei due byte riservati (*RES1* e *RES2*) per effettuare una stima del ritardo e modificare di conseguenza i timeout allo scopo di migliorare la velocità e la stabilità della connessione. Per quanto riguarda l’applicazione, un’idea potrebbe essere quella di aumentare le prestazioni tramite un approccio *multithreading*. Va precisato che in ogni caso si avrebbe come collo di bottiglia la connessione all’AP WiFi e quindi questa modifica gioverebbe soltanto nel caso di ambienti ricchi di *beacon* consentendone un filtraggio più veloce. Nello *smart core* sarebbe utile l’aggiunta di un secondo *controller* Bluetooth così che si possa distinguere l’interfaccia da usare per il servizio di *HelloBroadcast* ed un’altra da utilizzare per offrire le funzionalità di *smart object*. Una più interessante se pur più complessa idea potrebbe essere quella di estendere il protocollo e renderlo in grado di supportare reti a maglia (*mesh network*). Con un simile approccio si avrebbero sicuramente risvolti interessanti ma anche tempi di propagazione maggiori.

7.3 Vantaggi

Possono essere identificati diversi vantaggi che rappresentano altresì dei parametri per la scelta dello contesto di utilizzo dell’applicazione. Tra i vantaggi troviamo sicuramente la flessibilità che il sistema offre per diversi aspetti come la possibilità di avere requisiti poco stringenti o ancora l’estensione degli *smart object* con la definizione di una nuova funzione. Un ulteriore vantaggio è dato dal grado di compatibilità offerto dal sistema che può essere eseguito su varie piattaforme hardware e su vari sistemi operativi. Un altro punto di forza è dato dalla facilità di distribuzione e installazione (o aggiornamento) del software in esecuzione sugli oggetti, infatti l’intero sistema operativo è contenuto in una microSD che consente una facile installazione e semplifica le operazioni di backup e ripristino.

Le prestazioni sono da ritenersi pienamente soddisfacenti e garantiscono un’interazione pressoché istantanea in termini di esperienza utente. Più esattamente i tempi di risposta del sistema variano ma si mantengono nell’ordine dei decimi di secondo.

7.4 Limiti

Il sistema realizzato presenta chiaramente anche delle limitazioni che saranno descritte in tale paragrafo e verranno divise in due diverse categorie costituite dai limiti imposti dal protocollo e quelli imposti dall'implementazione (software e hardware). I campi *user_id* e *object_id* sono entrambi rappresentati come interi positivi di 2 byte e consentono quindi di differenziare al più $2^{16} = 65536$ utenti e *smart_object* differenti. Si noti che una condizione più stringente sul numero di *smart_object* si ha nella configurazione del server DHCP ed in particolare ci si riferisce alla sottorete fornita, dove si è considerato un massimo di 224 dispositivi connessi. Chiaramente questo limite può essere superato semplicemente assegnando una sottorete di dimensioni maggiori. Inoltre ricordando che gli *advertisement* sono trasmessi a frequenze nell'intorno dei 2.4Ghz si può notare che in ambienti con un elevato disturbo su tale frequenza si riscontra una maggiore perdita di pacchetti.

7.5 Confronto con le tecnologie esistenti

Infine si può effettuare un confronto con le tecnologie più diffuse che posseggono analoghe funzionalità. Tra queste spiccano sicuramente soluzioni come Google Home o Amazon Echo. La principale differenza consiste nell'approccio *cloud* che viene adottato da questi sistemi. Questo determina numerose differenze tra cui:

1. L'utilizzo di server che collezionano ed elaborano i dati con un approccio maggiormente *centralizzato*.
2. Un round trip time (RTT) abbastanza elevato.
3. L'introduzione di un nuovo elemento vulnerabile a guasti o attacchi.
4. Il possibile utilizzo solo in luoghi che dispongono di una connessione.
5. Problemi inerenti la privacy e la condivisione dei propri dati personali.

In aggiunta le alternative considerate non consentono una generalità e flessibilità come quello descritto da tale elaborato.

In conclusione si può affermare che il prodotto finale, soddisfacendo pienamente i requisiti imposti e le funzionalità richieste, rappresenta, in diversi scenari, una valida alternativa alle soluzioni preesistenti e si propone come un nuovo approccio al problema.

Bibliografia

- [1] Paolo Atzeni. *Basi di dati*. McGraw Hill.
- [2] C. Bisikian. *Bluetooth Protocol Architecture*. URL https://www.bluetooth.org/foundry/sitecontent/document/Protocol_Architecture.
- [3] M. Eftimakis D. Chomienne. *Bluetooth Tutorial*. URL <http://www.newlogic.com/products/Bluetooth-Tutorial-2001.pdf>.
- [4] Naresh Kumar Gupta. *Inside Bluetooth Low Energy*, 2016.
- [5] Texas Instrument. *Bluetooth Low Energy Scanning and Advertising*, 2016. URL http://dev.ti.com/tirex/content/simplelink_academy_cc2640r2sdk_1_12_01_16/modules/ble_scan_adv/ble_scan_adv_basic.html.
- [6] J. Kardach. *Bluetooth architecture overview*.
- [7] Donald E. Knuth. *The Art of Computer Programming*. Addison Wesley.
- [8] Cambridge University Press. *Cambridge Advanced Learner's Dictionary & Thesaurus*. Cambridge University Press.
- [9] Bluetooth SIG. *ManufacturerID*. URL <https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers>.
- [10] Bluetooth SIG. *Bluetooth Core v5*, 2016. URL <https://www.bluetooth.com/specifications/bluetooth-core-specification>.
- [11] William Stallings. *Crittografia e sicurezza delle reti*. McGraw Hill.
- [12] J. Stripp. *Bluetooth: Value Adds and Opportunities*, 2001. URL http://www.palowireless.com/bluearticles/docs/bluetooth_value_adds.pdf.
- [13] Guido van Rossum. Pep 8 – style guide for python code, July 2001. URL <https://www.python.org/dev/peps/pep-0008/>. [online] <https://www.python.org/dev/peps/pep-0008/>.
- [14] M. Weiser. *The origins of ubiquitous computing research at PARC in the late 1980s*.

Appendice A

File di configurazione

In questa appendice vengono mostrati i file di configurazione. Nel caso di file di configurazione di servizi viene fornito anche il loro percorso (nel caso di Raspbian Strech).

Dnsmaq

File di configurazione dnsmaq (*/etc/dnsmaq.conf*):

```
except-interface=eth0
interface=wlan0
dhcp-authoritative
bind-interfaces
#listen-address=192.168.0.1
dhcp-range=192.168.0.2,192.168.0.225,12h
```

Interfaccia WLAN

File di configurazione per l’interfaccia WLAN (*/etc/network/interfaces*):

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d
auto wlan0
iface wlan0 inet static
```

```
address 192.168.0.1
netmask 255.255.255.0
```

Hostapd

File di configurazione per il servizio di Hostapd (*/etc/network/interfaces*):

```
interface=wlan0
#driver=nl80211
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ssid=SmartAP
wpa=2
wpa_passphrase=I7AGy5W991eH0zTW
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

File di configurazione smart object

Esempio del file json generato per la trasmissione dei comandi effettuata dallo *smart core* verso gli *smart object* e gli utenti.

```
{
    "cmd_type": [
        "0x01",
        "0x02"
    ],
    "0x01": {
        "0x00": {},
        "0x01": "sb_open",
        "0x02": "turn",
        "0x03": "rotate"
    },
}
```

```

        "0x01": {
            "0x01": "sb_timer"
        },
        "0x02": {
            "0x01": "set_temperature",
            "0x02": "set_volume"
        }
    }
}

```

Si mostra inoltre una porzione del file di configurazione simmetrico, in altri termini quello utilizzato dall'applicazione dell'utente per poter inviare i pacchetti con i campi *CMD[]* corretti a partire da un *alias*, ovvero un nome simbolico che viene associato all'output della rete neurale che si occupa del riconoscimento vocale.

```

{
    "cmd_alias1": {
        "cmd_type": "0x01",
        "cmd_class": "0x00",
        "cmd_opcode": "0x01",
        "function_name": "sb_open"
    }
}

```

Si noti che l'ultimo campo (*function_name*) non è strettamente necessario ed è stato aggiunto per una maggiore chiarezza e facilità di associazione. Infine si mostra un esempio dello script *smartbeacon_command.py* associato ad uno *smart object* e contenente tutte le procedure che devono essere effettuate nel caso di un comando.

```

from threading import Timer
def sb_open():
    print("Gate opened")
    return 1

def sb_timer():
    t = Timer(10.0, sb_open)
    t.start()
    print("Timer was set (10sec), wait")

```

Appendice B

Generazione e installazione del file di immagine

Per comodità si è ritenuto conveniente fornire le istruzioni utilizzate per la generazione del file di immagine che consente l'installazione veloce. Bisogna notare che, poiché le microSD possono presentare deterioramenti nel corso del tempo, nel caso di una partizione definita su tutto il disco, risulta necessario restringere la partizione ext4 di Raspbian di un centinaio di MB. Questa operazione può essere svolta con un software quale parted (o la sua versione con gui *gparted*) direttamente dalla Raspberry.

Successivamente risulta necessario estrarre la microSD ed eseguirne una copia nella sua interezza per poi tagliare i bit salvati in eccesso. Su sistemi *unix-like* è possibile eseguire queste due operazioni tramite i seguenti comandi (alle volte presenti in forme e versioni differenti):

```
sudo dd bs=4m if=backup_dd.img of=/dev/rdiskX
```

dove *X* è il numero relativo alla posizione del disco considerato (la microSD). Successivamente bisogna verificare, per ulteriore conferma, il settore di inizio e fine partizione specificata al momento del ridimensionamento con il comando:

```
fdisk -lu myimg.img
```

Infine sarà sufficiente tagliare i dati in eccesso dal file di immagine tramite l'esecuzione di:

```
truncate --size=$((end_partition_sector+1)*512) backup_dd.img
```

dove si è assunta una dimensione del blocco (*blocksize*) di 512 byte (standard comune per le microSD) e dove *end_partition_sector* rappresenta l'ultimo settore appartenente alla partizione.

Installazione veloce del sistema Per l'installazione sulla microSD sarà sufficiente un solo comando che effettua la copia dall'immagine alla microSD collegata:

```
sudo dd bs=4m if=backup_dd.img of=/dev/rdiskX conv=noerror,sync
```