



UNIVERSITÀ POLITECNICA DELLE MARCHE

PROGETTO DI RETI DI SENSORI WIRELESS PER
IOT

Realizzare la coesistenza degli applicativi “beacon Eddystone” e “iBeacon” (nRF SDK v16.0) all’interno dell’applicativo “Light Switch Server” (nRF SDK MESH v4.0.0)

Matteo Orlandini

Prof. Paola PIERLEONI

Andrea GENTILI e Marco MERCURI

18 maggio 2020

Indice

1	Introduzione	1
1.1	Bluetooth Low Energy	1
1.2	Panoramica delle operazioni Bluetooth Low Energy	7
1.3	Beacon BLE	9
1.3.1	Eddystone	12
1.3.2	iBeacon	13
1.4	Bluetooth Mesh	14
1.4.1	Concetti fondamentali del Bluetooth Mesh	15
1.4.2	Architettura del Bluetooth Mesh	22
1.5	Introduzione al BLE IOT	23
1.6	Installare l'ambiente di sviluppo e scaricare l'SDK Nordic	24
1.7	Compilare gli esempi dello stack mesh	25
1.7.1	Primo setup	25
1.7.2	Compilare con SES	25
1.7.3	Eseguire gli esempi usando SEGGER Embedded Studio	26
2	Esempio Light Switch	27
2.1	Requisiti Hardware	28
2.2	Setup	28
2.3	LED e assegnazioni dei pulsanti	29
2.4	Test con il provisioner statico	30
2.5	Test con l'app nRF Mesh	31
2.6	Interagire con le schede	33
2.7	J-Link RTT Viewer	34
3	Applicazione Eddystone Beacon	36
3.1	Protocollo Eddystone	37
3.2	Caratteristiche di sicurezza	37
3.3	Setup	38
3.4	Testing	38
4	Applicazione Beacon Transmitter	40
4.1	Configurare i valori Major e Minor	40
4.2	Testare l'applicazione	41
5	Esempio Beacons nella rete Mesh	42
5.1	Mandare Beacon	42
5.2	Callback RX	42
5.3	Requisiti software	42

5.4	Setup	43
5.5	Testare l'esempio	43
6	Implementazione di beacon statici nell'esempio Light Switch Server	44
6.1	Integrazione dell'esempio Beacon Transmitter nel Light Switch Server	44
6.2	Integrazione dell'esempio Mesh Beacons nel Light Switch Server	52
6.2.1	Implementazione di iBeacon	52
6.2.2	Implementazione dei beacon Eddystone	57
6.3	Integrazione di iBeacon e beacon Eddystone in Light Switch Server	59
7	Integrare la Mesh negli esempi dell'SDK nRF5	61
7.1	Includere le funzionalità dell'SDK Mesh nRF5 in un esempio dell'SDK nRF5	61
7.2	Integrazione delle funzionalità Mesh nell'esempio Eddystone . .	63
8	Coesistenza tra l'applicazione Eddystone e l'esempio Light Switch	65
8.1	Operazioni preliminari per configurare l'ambiente di sviluppo .	65
8.1.1	SDK configuration header file	65
8.1.2	Directory del preprocessore	70
8.1.3	Macro del preprocessore	71
8.1.4	File sorgente	71
8.2	Implementazione delle funzionalità Eddystone	73
8.3	Eliminazione delle funzionalità dell'esempio coexistence non necessarie	76
9	Conclusioni	78

1 Introduzione

Questa relazione mostra come realizzare un applicativo che implementi le funzionalità del Bluetooth Mesh e di beaconing. In particolare sono stati integrati gli applicativi "beacon Eddystone" e "iBeacon", presenti nell'SDK nRF5 v16.0, nell'esempio Light Switch dell'SDK Mesh v4.0.0.

Questo elaborato illustra tutti i passaggi necessari per il corretto funzionamento degli applicativi citati in precedenza. Sono anche presentati tutti i tentativi fatti nel corso del progetto, con la descrizione dei problemi incontrati e della relativa risoluzione, al fine di produrre l'applicazione finale funzionante.

La relazione è divisa in un'introduzione in cui viene presentata la tecnologia Bluetooth Low Energy, i beacon e il Bluetooth Mesh rispettivamente nei capitoli 1.1, 1.3 e 1.4, una descrizione degli esempi e applicativi usati e la procedura per integrare tra loro le relative funzionalità. L'esempio Light Switch è presentato nel capitolo 2, mentre gli altri esempi usati per implementare le funzioni precedentemente citate sono nei capitoli 3, 4 e 5. Una prima implementazione di applicazione che usa il Bluetooth Mesh mentre vengono trasmessi beacon è illustrata nel capitolo 6, mentre un'altra versione è descritta nel capitolo 7 e l'applicazione finale, con tutti i passaggi da seguire per sviluppare il progetto, è delineata nel capitolo 8.

1.1 Bluetooth Low Energy

La tecnologia Bluetooth è un sistema di comunicazione senza fili a corto raggio inteso a rimpiazzare i dispositivi elettronici con connessione via cavo. Le caratteristiche principali del Bluetooth sono la robustezza, il basso consumo e il basso costo.

Ci sono due forme di sistemi Bluetooth: Basic Rate (BR) e Low Energy (LE). Hanno in comune la ricerca del dispositivo, la costruzione e il meccanismo di connessione. Il Bluetooth LE include caratteristiche per lavorare con prodotti che richiedono un consumo di corrente, una complessità e un costo minori rispetto il BR.

I dispositivi che implementano entrambi i sistemi possono comunicare con altri apparati che supportano sia il Bluetooth BR sia LE. In alcuni casi è supportato solo uno dei due sistemi, quindi i dispositivi che implementano entrambi possono supportare la maggior parte delle situazioni di lavoro.

Il sistema centrale Bluetooth consiste in un host e in uno o più controller. Osservando la figura 1 si possono definire l'host come un'entità logica che contiene tutti i layer sopra la Host Controller Interface (HCI) e il controller come un'entità logica che si compone di tutti i livelli al di sotto di HCI.

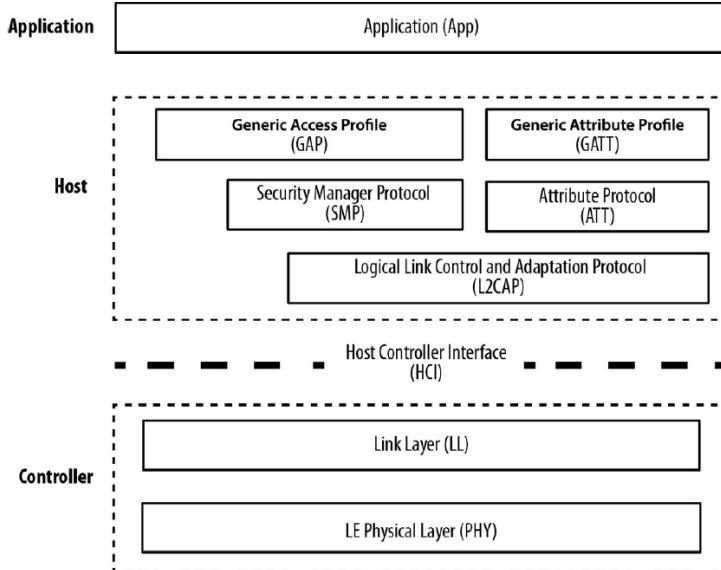


Figura 1: Stack protocollare Bluetooth Low Energy [6]

I layer inclusi nel livello host sono:

- Generic Access Profile (GAP)
- Generic Attribute Profile (GATT)
- Attribute Protocol (ATT)
- Security Manager Protocol (SMP)
- Logical Link Control and Adaptation Protocol (L2CAP)

I layer del livello controller sono:

- Link Layer (LL)
- Physical Layer (PHY)

Vengono ora descritti i vari blocchi. Il layer **Generic Access Profile** (GAP) definisce le procedure per scoprire e connettere servizi a un dispositivo Bluetooth. I servizi GAP comprendono il rilevamento dei dispositivi, le modalità di connessione, la sicurezza, l'autenticazione e i modelli di associazione.

Il **Generic Attribute Profile** (GATT) è un protocollo che definisce gli attributi e i servizi di un dispositivo BLE. Quando il profilo è configurato dal GAP, gli attributi sono assegnati a particolari servizi.

Si serve del protocollo ATT per trasportare i dati sotto forma di comandi, richieste, risposte, indicazioni, notifiche e conferme tra dispositivi. Quando si richiedono i dati sotto forma di notifiche, il server può inviare il valore di un attributo in qualsiasi momento. La figura 2 mostra la gerarchia del profilo GATT.

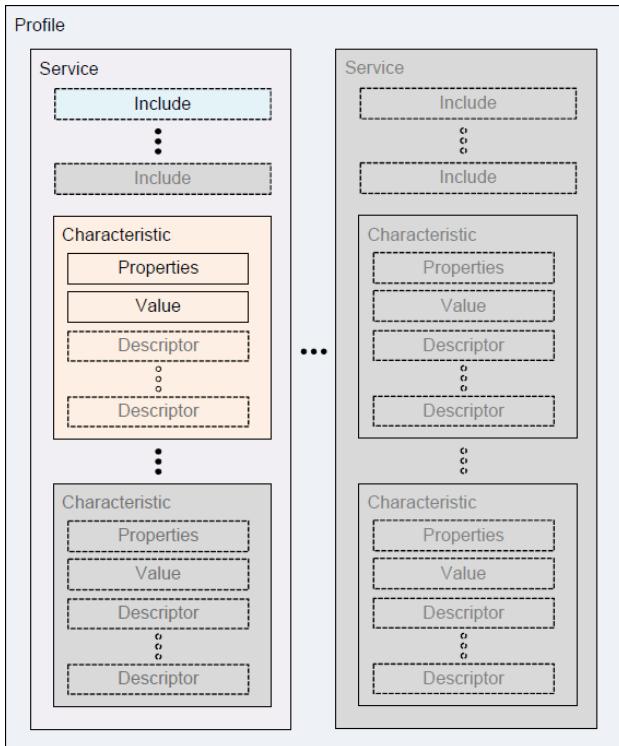


Figura 2: Gerarchia del profilo GATT [1]

Un *servizio* è una raccolta di dati e comportamenti associati per realizzare una particolare funzione o caratteristica. Nel GATT, una definizione di servizio può contenere servizi di riferimento, caratteristiche obbligatorie e caratteristiche opzionali. Esistono due tipi di servizi: primari e secondari. Un servizio primario espone la funzionalità principale utilizzabile di questo dispositivo. Un servizio secondario è un servizio a cui si intende fare riferimento solo da un servizio primario, un altro servizio secondario o altra specifica di livello superiore.

Una dichiarazione di servizio è un attributo con il tipo impostato sull'UUID per «Servizio primario» (0x2800) o «Servizio secondario» (0x2801). Il campo valore può essere un UUID a 16 o a 128 bit. Le autorizzazioni di attributo devono essere di sola lettura e non richiedono autenticazione o autorizzazione.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»]	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

Figura 3: Dichiarazione di un servizio [1]

Una caratteristica è un attributo utilizzato in un servizio con proprietà e informazioni su come viene visualizzato o rappresentato il valore e su come accedervi. Una dichiarazione di una caratteristica è un attributo con il tipo impostato sull'UUID «Caratteristica» (0x2803) e il valore dell'attributo diviso in proprietà, handle e UUID. Le autorizzazioni devono essere leggibili e non richiedono autenticazione o autorizzazione.

Attribute Handle	Attribute Type s	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Characteristic Properties	Characteristic Value Attribute Handle	Characteristic UUID	Read Only, No Authentication, No Authorization

Figura 4: Dichiarazione di una caratteristica [1]

Il campo valore di una caratteristica è disponibile solamente in lettura ed è illustrato nella figura di seguito.

Attribute Value	Size	Description
Characteristic Properties	1 octets	Bit field of characteristic properties
Characteristic Value Handle	2 octets	Handle of the Attribute containing the value of this characteristic
Characteristic UUID	2 or 16 octets	16-bit Bluetooth UUID or 128-bit UUID for Characteristic Value

Figura 5: Campo value nella dichiarazione della caratteristica [1]

- Il campo proprietà determina come è possibile utilizzare il valore o come accedere ai descrittori delle caratteristiche.
- Il campo handle contiene l'handle dell'attributo in cui è presente il valore della caratteristica.
- Il campo UUID è un UUID Bluetooth a 16 bit o UUID generico a 128 bit che descrive il tipo di valore della caratteristica.

I *descrittori* delle caratteristiche vengono utilizzati per contenere informazioni correlate al valore della caratteristica. Il profilo GATT definisce un

set standard di descrittori che possono essere utilizzati da profili di livello superiore. Ogni descrittore di caratteristiche è identificato dall'UUID.

Il *Client Characteristic Configuration Descriptor* (CCCD) è un descrittore di caratteristiche facoltativo che definisce come la caratteristica possa essere configurata da uno specifico client. Un client può scrivere nel CCCD per controllare la configurazione della caratteristica. Possono essere richieste autenticazione e autorizzazione dal server per scrivere nel descrittore. Il descrittore è contenuto in un attributo il cui tipo deve essere impostato sull'UUID come «Client Characteristic Configuration» (0x2902). Il CCCD agisce come un interruttore, abilitando o disabilitando gli aggiornamenti del campo “value” del “characteristic value” della caratteristica in cui si trova. Il suo valore è un bitfield a due bit, uno corrispondente alle notifiche e l’altro alle indicazioni.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	02902 – UUID for «Client Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Figura 6: Dichiarazione *Client Characteristic Configuration* [1]

I *Characteristic Configuration Bits* possibili sono elencati in tabella:

Configurazione	Valore	Descrizione
Notifiche	0x0001	Il campo value sarà notificato
Indicazioni	0x0002	Il campo value sarà indicato

Tabella 1: Definizione dei *Characteristic Configuration Bits* [1]

Il **protocollo ATT** definisce due ruoli: *server* e *client*. Il server è il dispositivo che accetta comandi e richieste in arrivo dal client e a cui invia risposte, indicazioni e notifiche. Il client, invece, è il dispositivo che avvia comandi e richieste verso il server e può ricevere risposte, indicazioni e notifiche inviate dal server. Gli attributi di un server possono essere scoperti, scritti o letti da un client specifico. Un attributo è composto da quattro parti: *handle*, *tipo*, *valore* e *permessi*. La figura 7 mostra una rappresentazione logica di un attributo.

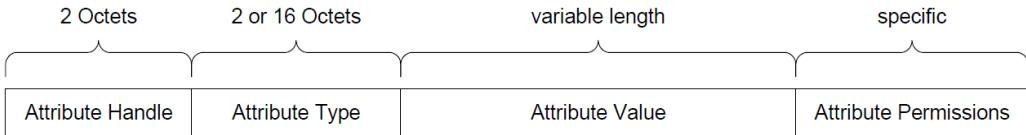


Figura 7: Rappresentazione logica di un attributo [1]

L'*handle* è un indice corrispondente a un attributo specifico. Il *tipo* dell'attributo è un UUID (universally unique identifier) che specifica ciò che esso rappresenta. Il *valore* è il dato descritto dal tipo dell'attributo e indicizzato dall'handle. Le *autorizzazioni* vengono utilizzate dal server per determinare se è consentito l'accesso in lettura o in scrittura per un determinato attributo e sono stabilite dal profilo GATT.

Il **Security Manager Protocol** (SMP) è utilizzato per generare chiavi di crittografia. Il protocollo opera su un canale L2CAP dedicato e gestisce anche l'archiviazione delle chiavi di crittografia e di identità. Si interfaccia direttamente con il controller per fornire chiavi memorizzate durante le procedure di crittografia o associazione.

Il blocco **Logical Link Control and Adaptation Protocol** (L2CAP) è responsabile della gestione dell'ordinamento dei frammenti dei pacchetti in banda base e della pianificazione dei canali L2CAP per garantire che questi non abbiano l'accesso negato al canale fisico a causa dell'esaurimento delle risorse del controller. Ciò è necessario perché l'architettura non presuppone che un controller abbia un buffering illimitato o che l'HCI abbia a disposizione una larghezza di banda infinita.

L'**Host Controller Interface** (HCI) è un'interfaccia standardizzata grazie alla quale dispositivi host possono accedere agli strati più bassi dello stack Bluetooth. Attraverso l'HCI un host può inviare dati agli altri dispositivi e ricevere informazioni dalle altre unità presenti nella piconet. Con questa interfaccia un host può ordinare al suo layer baseband di creare un link ad uno specifico dispositivo Bluetooth, eseguire richieste di autenticazione e passare una chiave per il link.

Il **Link Layer** (LL) è la parte che si interfaccia direttamente al livello fisico (PHY). È responsabile dell'advertising, della scansione e della creazione e gestione delle connessioni.

Il **Physical Layer** (PHY) è responsabile della trasmissione e della ricezione di pacchetti di informazioni sul canale fisico. Un percorso di controllo tra la banda di base e il blocco PHY consente al blocco di banda di base di controllare il timing e la portante del blocco PHY. Trasforma uno stream di dati da e verso il canale fisico e la banda base nei formati richiesti.

1.2 Panoramica delle operazioni Bluetooth Low Energy

La radio del Bluetooth LE opera nella banda 2.4 GHz con una ricetrasmissione frequency hopping, supportando il bit rate di 1 Megabit al secondo (1 Mb/s)

Il Bluetooth LE usa due tipi di accesso multiplo: Frequency Division Multiple Access (FDMA) e Time Division Multiple Access (TDMA). Nello schema FDMA sono usati 40 canali fisici separati da 2 MHz, 3 canali sono usati per advertising e 37 sono usati per i dati. La TDMA è sfruttata quando un dispositivo trasmette ad un predeterminato intervallo temporale e il corrispondente apparato risponde dopo un periodo prestabilito.

Nella figura 8 viene presentata la suddivisione dei canali nel Bluetooth Low Energy.

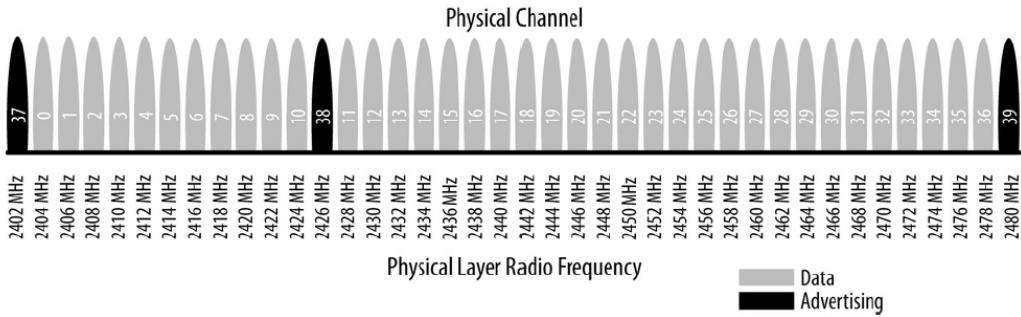


Figura 8: Suddivisione dei canali del Bluetooth LE [6]

Il canale fisico è suddiviso in unità temporali chiamate *events*. I dati sono trasmessi in pacchetti posizionati in questi eventi che si dividono in advertising e connection.

I dispositivi che trasmettono i pacchetti di advertising sono chiamati *advertisers*, quelli che il ricevono senza intenzioni di connettersi al dispositivo di advertising sono chiamati *scanners*. Ciascun dispositivo Bluetooth è identificato da un Bluetooth device address, cioè un numero di 48 bit (6 byte). Può essere di due tipi: pubblico, se è un indirizzo fisso, o random, se cambia ad ogni avvio dell'applicazione. Ogni dispositivo advertiser trasmette, ad intervalli regolari che vanno dai 20 ms ai 10.24 secondi pacchetti di grandezza fino a 31 byte contenenti: il nome, l'indirizzo, la classe del dispositivo, la lista dei servizi offerti e altre informazioni (es. marca). Questa operazione è detta advertising. In base al tipo di pacchetto, lo scanner può fare una richiesta all'advertiser, seguita eventualmente da una risposta da parte di quest'ultimo. Il canale di advertising cambia al successivo pacchetto trasmesso nello stesso evento. L'advertiser può terminare un evento in qualsiasi momento, all'inizio

del successivo evento di advertising viene usato il primo canale fisico come mostrato in figura 9

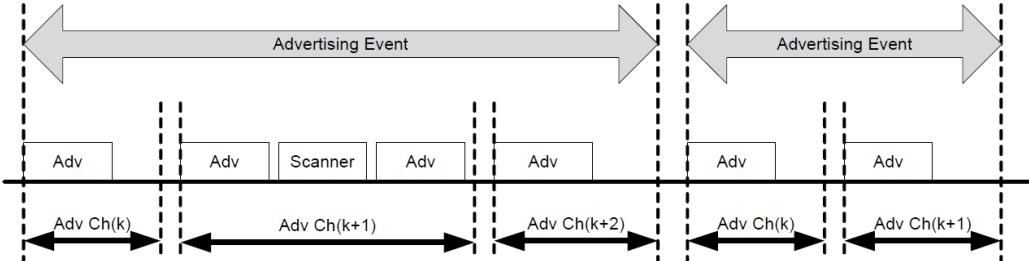


Figura 9: Eventi advertising [1]

I dispositivi che tentano di connettersi con un altro dispositivo vengono chiamati *iniziatori*. Se l'advertiser sta usando un evento di advertising, un iniziatore può effettuare una richiesta di connessione usando lo stesso canale in cui è stato ricevuto il pacchetto. L'evento è così terminato e la connessione inizia se l'advertiser riceve e accetta la richiesta manda. Una volta che la connessione è stabilita, l'iniziatore diventa il master e l'advertiser è lo slave. Gli eventi connection sono usati per mandare pacchetti di dati tra master e slave nello stesso canale. Il master inizia e finisce ogni evento di connessione.

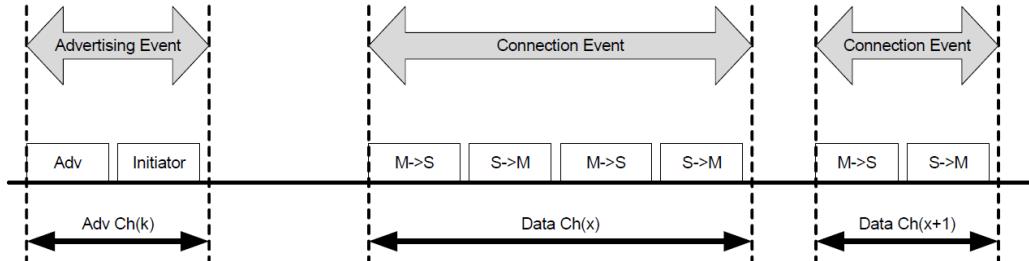


Figura 10: Eventi connection [1]

I dispositivi in una piconet, cioè un gruppo di apparati sincronizzati, usano la tecnica FHSS (Frequency Hopping Spread-Spectrum) che consiste nel commutare il canale di trasmissione con una frequenza di 1600 volte al secondo. L'algoritmo usato per cambiare le frequenze nel Bluetooth LE, condiviso tra il trasmettitore dati ed il ricevitore, è pseudo random nello scegliere le 37 frequenze disponibili, alcune delle quali possono essere escluse per evitare disturbi da altri dispositivi nella stessa area, permettendo inoltre di ridurre al minimo la probabilità di interferenza.

Nella richiesta di connessione, il master invia dei dati chiamati *connection parameters*: essi sono un set di parametri che determinano quando e

come il central (master) ed il peripheral (slave) scambieranno i dati durante la connessione. Questi parametri vengono sempre stabiliti dal central, ma la periferica può inviare una *connection parameter update request*, ovvero un pacchetto di dati contenente dei “suggerimenti” di possibili parametri da utilizzare. Questi parametri sono:

- *Connection Interval*: determina quanto spesso il central chiederà il dato alla periferica. Lo standard, infatti, stabilisce un intervallo di possibili valori compreso tra 7,5 ms e 4 s. La periferica suggerisce un ulteriore intervallo, intrinseco al connection interval, compreso tra due valori in ms (i cui estremi sono stabiliti dai parametri MIN_CONN_INTERVAL e MAX_CONN_INTERVAL), ma è il central ad adottarne uno tra i possibili.
- *Slave latency*: è il numero di richieste di lettura dei dati consecutive inviate dal central alla periferica che verranno ignorate. Il suo valore è uno dei possibili compresi nel range tra 0x0000 (cioè ad ogni richiesta di lettura la periferica risponde con l’invio dei dati) e 0x01F3 (499, cioè alla 500esima richiesta di lettura la periferica risponderà con l’invio dei dati, trascurando tutte le precedenti 499). Ciò consente alla periferica di rimanere in modalità sleep per un tempo lungo se non ha dei dati aggiornati da inviare, permettendo così di risparmiare energia.
- *Connection supervision timeout multiplier*: poiché i device coinvolti nella connessione non sono in grado di capire quando questa viene persa, è necessario che passi un tempo “abbastanza” lungo, chiamato timeout, affinché avvenga il trasferimento dei dati tra i due dispositivi prima di presupporre che la connessione sia stata persa. Ha un range compreso tra 100 ms e 32 secondi (3200 ms). È chiaro inoltre che il timeout dovrà essere maggiore del connection interval.

1.3 Beacon BLE

I *beacon* sono una tecnologia wireless basata sulla trasmissione di piccole informazioni che possono essere contenute dati ambientali (temperatura, pressione dell’aria, umidità ecc.), dati di micro-localizzazione (tracciamento delle risorse, vendita al dettaglio, ecc.) o dati di orientamento (accelerazione, rotazione, ecc.). I dati trasmessi sono in genere statici ma possono anche essere dinamici e cambiare nel tempo. Con l’uso del Bluetooth Low Energy (BLE), i beacon possono essere progettati per funzionare per anni con una singola batteria a bottone.

BLE ha la capacità di scambiare dati in due stati: connesso e advertising, come già visto nel capitolo 1.2. La modalità connessa utilizza il layer Generic Attribute (GATT) per trasferire i dati in una connessione one-to-one. La modalità advertising utilizza il livello GAP (Generic Access Profile) per trasmettere dati a chiunque sia in ascolto, in modalità one-to-many. I beacon BLE sfruttano l'advertising per trasmettere dati in pacchetti periodici e appositamente formattati.

Il beacon non collegabile viene trasmesso da un dispositivo BLE in modalità broadcast che invia semplicemente informazioni archiviate internamente. Poiché la trasmissione non collegabile non attiva alcuna capacità di ricezione, consente di ottenere il minor consumo energetico possibile perché il dispositivo si sveglia da uno stato di sleep, trasmette dati e torna in sleep.

Il beacon collegabile è trasmesso da un dispositivo BLE in modalità periferica, il che significa che può non solo trasmettere, ma anche ricevere. Ciò consente a un dispositivo centrale (ad esempio uno smartphone) di connettersi e interagire con i servizi implementati sul dispositivo beacon. I servizi forniscono una o più caratteristiche che potrebbero essere modificate da un dispositivo.

I dati trasmessi da un dispositivo BLE sono formattati in base alle specifiche Bluetooth e sono composti dalle parti mostrate in figura 11.

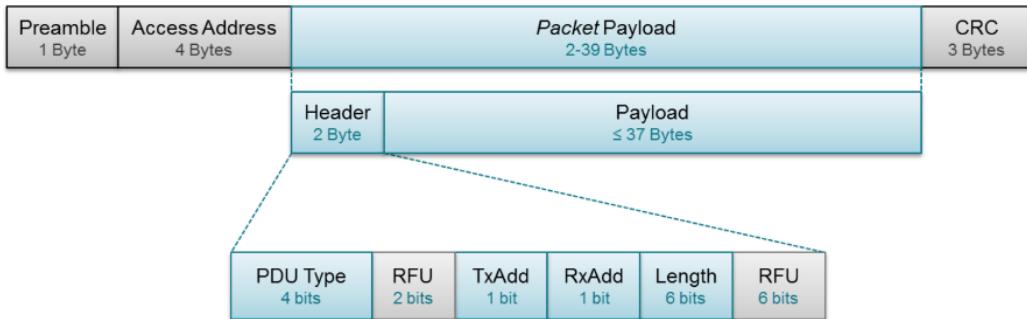


Figura 11: Pacchetto dati del BLE [5]

Il preambolo è un valore di 1 byte utilizzato per la sincronizzazione e la stima dei tempi sul ricevitore ed è sempre 0xAA per i pacchetti trasmessi. L'indirizzo di accesso è anch'esso fisso ed è impostato su 0x8E89BED6. Il payload totale del pacchetto è costituito da un'header e da un payload. L'header descrive il tipo di pacchetto e il tipo di PDU definisce la modalità di funzionamento del dispositivo. Per le applicazioni di trasmissione, esistono tre diversi tipi di PDU, come mostrato nella tabella 2. ADV_IND e ADV_NONCONN_IND sono stati descritti in precedenza, quando sono stati illustrati rispettivamente gli advertising collegabili e non collegabili.

ADV_SCAN_IND è semplicemente un broadcaster non collegabile che può fornire informazioni aggiuntive dalle risposte di scansione.

Tipo di PDU	Nome del pacchetto	Descrizione
0000	ADV_IND	Evento advertising collegabile
0010	ADV_NONCONN_IND	Evento advertising non collegabile
0110	ADV_SCAN_IND	Evento advertising scansionabile

Tabella 2: Tipi di PDU di advertising per dati broadcasting [5]

Il bit TxAdd indica se l'indirizzo dell'advertiser, contenuto nel payload, è pubblico (TxAdd = 0) o random (TxAdd = 1). RxAdd è riservato ad altri tipi di pacchetti non trattati in questa sezione, poiché non si applicano ai beacon. La parte finale del pacchetto trasmesso è il Cyclic Redundancy Check (CRC). CRC è un codice di rilevamento degli errori utilizzato per convalidare il pacchetto e garantire l'integrità dei dati per tutti i pacchetti trasmessi. Il payload del pacchetto include l'indirizzo dell'advertiser insieme ai dati definiti dall'utente, come mostrato in figura 12. Questi campi rappresentano l'indirizzo e i dati trasmessi dal beacon.

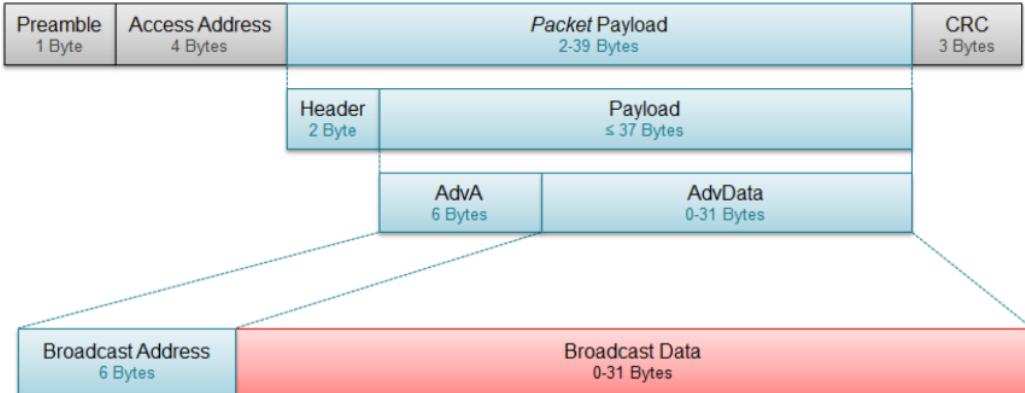


Figura 12: Dato broadcast BLE [5]

Il Broadcast Address può essere pubblico o random. Un indirizzo pubblico è uno standard IEEE802-2001 e utilizza un Organizationally Unique Identifier (OUI) ottenuto dalla IEEE Registration Authority. Gli indirizzi casuali possono essere generati direttamente dal beacon.

I dati trasmessi possono essere formattati secondo i formati di dati specificati dal Bluetooth SIG, alcuni di questi sono mostrati nella tabella 3. Il primo byte del broadcast data è la lunghezza del dato, mentre i successivi byte caratterizzano il tipo di advertising. Per gli scopi di questo capitolo

verranno presi in considerazione gli advertising che contengono Flags e dati Manufacturer-Specific.

Tipo di dato AD	Valore	Descrizione
Flags	0x01	Modalità di rilevamento del dispositivo
Service UUID	0x02 - 0x07	Servizi GATT del dispositivo
Local Name	0x08 - 0x09	Nome del dispositivo
TX Power Level	0x0A	Potenza in uscita del dispositivo
Manufacturer Specific Data	0xFF	Definito dall'utente

Tabella 3: Tipi di dati di advertising [5]

Per i flag, i primi tre byte dei dati trasmessi definiscono le capacità del dispositivo. Quando si utilizzano i dati specifici del produttore, viene utilizzato il flag 0xFF e i primi due byte dei dati stessi sono un codice identificativo dell'azienda produttrice. Ora verranno presentati due tipi di beacon esistenti: Eddystone e iBeacon.

1.3.1 Eddystone

Eddystone è un formato open beacon sviluppato da Google e progettato pensando alla trasparenza e alla robustezza. Il beacon Eddystone può essere rilevato da dispositivi Android e iOS. Il PDU Eddystone è mostrato in figura 13.

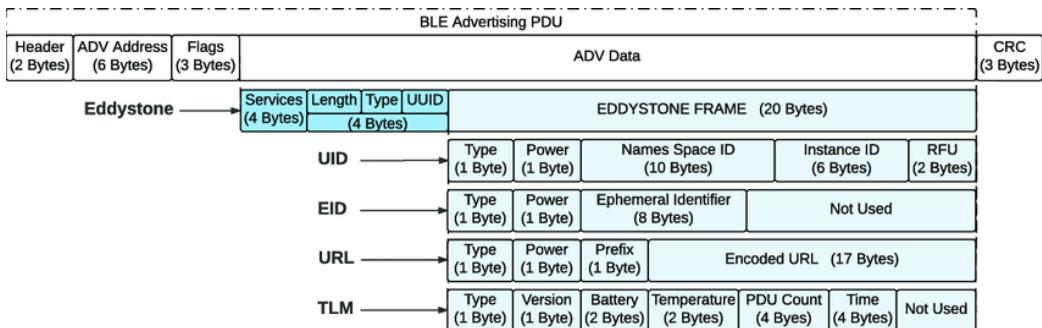


Figura 13: PDU Eddystone [2]

Diversi tipi di payload possono essere inclusi nel formato del frame, tra cui:

- Eddystone-UID: un ID statico univoco composto da un Namespace di 10 byte e un componente Instance di 6 byte.

- Eddystone-URL: un URL compresso che, una volta analizzato e decompresso, è direttamente utilizzabile dal client.
- Eddystone-TLM: dati sullo stato del beacon utili per la manutenzione e che alimentano l'endpoint diagnostico dell'API Beacon di Google Proximity. Deve essere integrato con un frame UID o EID (per cui la versione crittografata di eTLM preserva la sicurezza).
- Eddystone-EID: un frame beacon variabile nel tempo che può essere un identificatore stabile ricevuto da un resolver collegato, come l'API Proximity Beacon.

L'applicativo che mostra la trasmissione di beacon Eddystone è descritto nel capitolo 3.

1.3.2 iBeacon

L'iBeacon di Apple è stata la prima tecnologia BLE Beacon ad essere stata lanciata, quindi la maggior parte dei beacon si ispira al formato dati iBeacon. Gli iBeacon sono abilitati in molti SDK di Apple e possono essere letti e trasmessi da qualsiasi dispositivo BLE. iBeacon è uno standard proprietario e closed standard.

Gli iBeacon trasmettono quattro informazioni:

- Un *UUID* che identifica il beacon.
- Un *major number* che identifica un sottoinsieme di beacon all'interno di un gruppo più grande.
- Un *minor number* che identifica uno specifico beacon.
- Un livello di potenza di trasmissione, *TX power level*, che indica la potenza del segnale a un metro dal dispositivo. Questo numero deve essere calibrato dall'utente o dal produttore del dispositivo.

Un'applicazione che fa la scansione di beacon legge l'UUID, il major number e il minor number e fa riferimento a un database per ottenere informazioni sul beacon poiché il beacon stesso non porta informazioni descrittive. Il campo di potenza TX viene utilizzato con l'intensità del segnale misurata per determinare la distanza del beacon dallo smartphone. Un esempio di iBeacon è mostrato in figura 14.

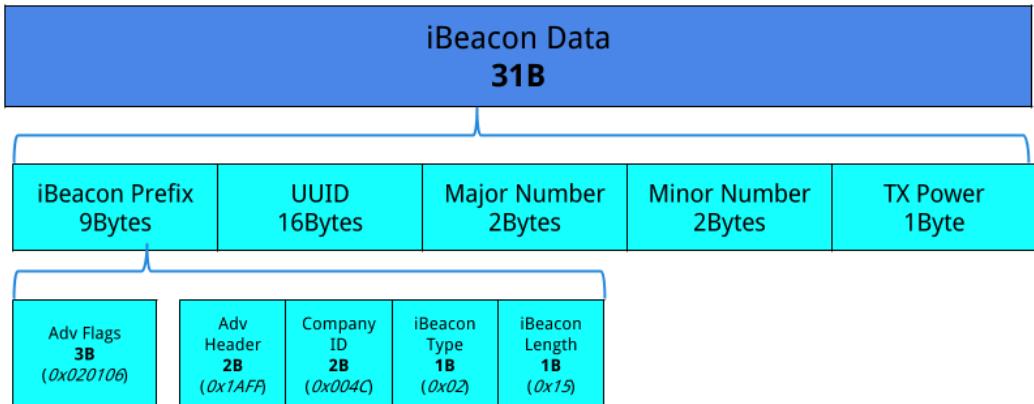


Figura 14: Dato iBeacon

L'*iBeacon Prefix* contiene i dati esadecimali: 0x0201061AFF004C0215 e si scomponete come segue:

- 0x020106 definisce il pacchetto come BLE General Discoverable, cioè si tratta di un pacchetto di sola trasmissione e non di connessione.
- 0x1AFF specifica che il dato seguente è lungo 26 byte ed è di tipo Manufacturer Specific.
- 0x004C è l'identificativo Apple del Bluetooth SIG
- 0x02 è un ID secondario che denota un beacon di prossimità
- 0x15 definisce che la lunghezza rimanente del dato è di 21 byte.

I restanti campi sono piuttosto autoesplicativi. L'UUID è un UUID BLE standard a 16 byte ed è in genere univoco per un'azienda. I numeri major e minor vengono utilizzati per indicare le risorse all'interno di tale UUID. Ad esempio il major number identifica un negozio, quindi 65.536 negozi possibili, e il minor number identifica i singoli tag all'interno dei negozi, quindi 65.536 tag possibili per ogni negozio.

L'applicazione che mostra la trasmissione di iBeacon è descritta nel capitolo 4.

1.4 Bluetooth Mesh

Il Bluetooth Mesh è uno standard di rete sviluppato e pubblicato dal Bluetooth SIG. Questa sezione spiega i concetti di base del Bluetooth Mesh che si basa sul Bluetooth Low Energy ma non ne condivide tutta l'architettura

come si può vedere in figura 15. La rappresentazione fisica del Bluetooth Mesh è compatibile con gli esistenti dispositivi Bluetooth Low Energy, in quanto i messaggi Mesh sono contenuti all'interno del payload di pacchetti di advertisement del Bluetooth Low Energy. Tuttavia, il Bluetooth Mesh specifica un layer host completamente nuovo, e anche se alcuni concetti sono condivisi, Bluetooth Mesh è incompatibile con l'host layer del BLE.

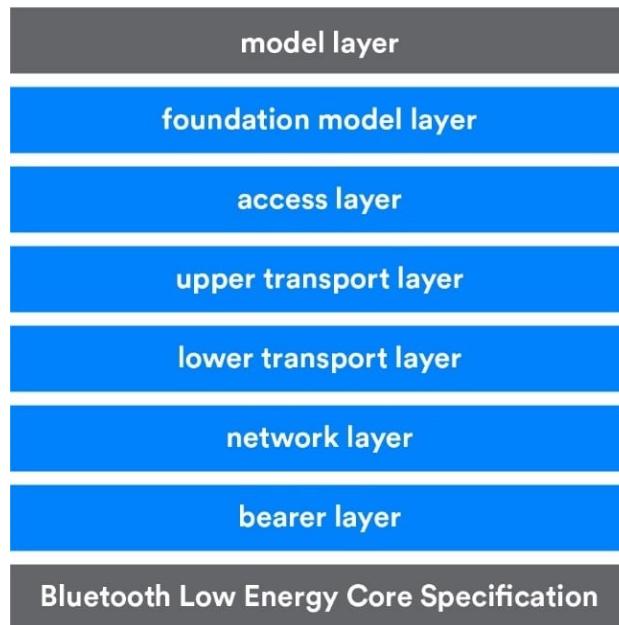


Figura 15: Architettura Bluetooth Mesh [7]

1.4.1 Concetti fondamentali del Bluetooth Mesh

Verranno ora introdotti i concetti e la terminologia del Bluetooth Mesh.

Mesh vs. Point-to-Point

La maggior parte dei dispositivi Bluetooth LE comunica tra loro utilizzando una semplice topologia di rete **point-to-point** che consente le comunicazioni one-to-one del dispositivo. Un aspetto interessante del Bluetooth è che consente ai dispositivi di impostare connessioni multiple. Al contrario, una rete **mesh** ha una topologia **many-to-many**, con ogni dispositivo in grado di comunicare con tutti gli altri dispositivi della mesh. La comunicazione avviene tramite messaggi e i dispositivi sono in grado di inoltrare messaggi ad altri dispositivi in modo tale che l'intervallo di comunicazione end-to-end ben oltre l'intervallo radio di ciascun singolo nodo.

Nodi

I dispositivi che fanno parte di una rete mesh sono chiamati **nodi**, mentre quelli che non lo sono sono chiamati “**unprovisioned devices**”. Il processo che trasforma un dispositivo che non fa parte della rete in un nodo della mesh è chiamato **provisioning**. Il provisioning è una procedura in cui un dispositivo non sottoposto a provisioning che possiede una serie di chiavi di crittografia diventa noto al dispositivo **provisioner**. Una di queste chiavi è denominata chiave di rete o NetKey in breve. Tutti i nodi in una rete mesh possiedono almeno un NetKey ed è il possesso di questa chiave che rende un dispositivo un membro della rete corrispondente.

Elementi

Alcuni nodi hanno più parti costituenti, ognuna delle quali può essere controllata in modo indipendente. Nella terminologia del Bluetooth Mesh, queste parti sono denominate **elementi**. Un prodotto di illuminazione a LED con tre luci, se aggiunto a una rete mesh, formerebbe un singolo nodo con tre elementi, uno per ciascuna delle singole luci a LED.

Messaggi

Quando un nodo deve interrogare lo stato di altri nodi o deve controllare in qualche modo altri nodi, invia un **messaggio** di tipo adatto. Se un nodo deve segnalare il suo stato ad altri nodi, invia un messaggio. Tutte le comunicazioni nella rete mesh sono "orientate ai messaggi" e vengono definiti molti tipi di messaggi, ognuno con un proprio codice operativo univoco. I messaggi possono essere di due tipi: **acknowledged** o **unacknowledged**.

I messaggi acknowledged richiedono una risposta dai nodi che li ricevono. La risposta ha due scopi: conferma che il messaggio a cui si riferisce è stato ricevuto e restituisce i dati relativi al destinatario del messaggio al mittente che può inviare nuovamente il messaggio se non riceve la risposta prevista. I messaggi unacknowledged non richiedono una risposta.

Indirizzi

I messaggi devono essere inviati da e verso un **indirizzo**. Bluetooth mesh definisce tre tipi di indirizzi. Un **indirizzo unicast** identifica in modo univoco un singolo elemento. Gli indirizzi unicast vengono assegnati ai dispositivi durante il processo di provisioning. Un **indirizzo di gruppo** è un indirizzo multicast che rappresenta uno o più elementi. Gli indirizzi di gruppo sono definiti dal Bluetooth Special Interest Group (SIG) e sono conosciuti come SIG Fixed Group Addresses o sono assegnati dinamicamente. Sono stati definiti 4 indirizzi di gruppo fisso SIG. Questi sono denominati All-proxies, All-friends, All-relays e All-nodes. I termini Proxy, Friend e Relay saranno

spiegati successivamente. Gli indirizzi di gruppo dinamici saranno stabiliti dall'utente tramite un'applicazione di configurazione e rifletteranno la configurazione fisica di una rete. Un **indirizzo virtuale** è un indirizzo che può essere assegnato a uno o più elementi che si estende su uno o più nodi. È un UUID a 128 bit a cui è possibile associare qualsiasi elemento ed è molto simile a un'etichetta. Gli indirizzi virtuali sono preconfigurati dal produttore del dispositivo.

Publish/Subscribe

Il processo di invio di un messaggio è noto come **publishing**. I nodi sono configurati per selezionare i messaggi inviati a indirizzi specifici per l'elaborazione, questo è noto come **subscribe**. In genere, i messaggi vengono indirizzati a indirizzi virtuali o di gruppo. Nella figura 16 possiamo vedere che il nodo "Switch 1" sta pubblicando nell'indirizzo di gruppo Kitchen. I nodi Light 1, Light 2 e Light 3 si abbonano ciascuno all'indirizzo Kitchen e quindi ricevono ed elaborano i messaggi pubblicati a questo indirizzo. In altre parole, Light 1, Light 2 e Light 3 possono essere accesi o spenti utilizzando l'interruttore 1. L'interruttore 2 pubblica all'indirizzo del gruppo Dining Room. Solo Light 3 si è abbonato a questo indirizzo e così è l'unica luce controllata dallo Switch 2. Si noti che questo esempio illustra anche il fatto che i nodi possono sottoscrivere messaggi indirizzati a più di un indirizzo distinto. Questo è sia potente che flessibile. Allo stesso modo, si noti come entrambi i nodi Switch 5 e Switch 6 pubblicano nello stesso indirizzo Garden. L'uso di indirizzi di gruppo e virtuali con il modello di comunicazione di pubblicazione/sottoscrizione ha un ulteriore vantaggio sostanziale in quanto la rimozione, la sostituzione o l'aggiunta di nuovi nodi alla rete non richiede la riconfigurazione di altri nodi.

Stato e Proprietà

Gli elementi possono essere in varie condizioni che nel Bluetooth Mesh sono rappresentate dal concetto di valori di **stato**. Uno stato è un valore contenuto in un elemento. Oltre ai valori, anche gli Stati hanno comportamenti associati e non possono essere riutilizzati in altri contesti. Ad esempio, considerando una luce che può essere accesa o spenta, il Bluetooth Mesh definisce uno stato chiamato **Generic OnOff**. La luce possederebbe questo stato e un valore di On corrisponderebbe e causerebbe l'illuminazione della luce, mentre un valore di stato Off rifletterebbe e causerebbe lo spegnimento della luce. Le **proprietà** sono simili agli stati in quanto contengono valori relativi a un elemento. Una proprietà fornisce il contesto per l'interpretazione di una caratteristica. Si consideri, ad esempio, la caratteristica del Bluetooth LE "Temperatura" che ha le proprietà temperatura ambiente interna e esterna

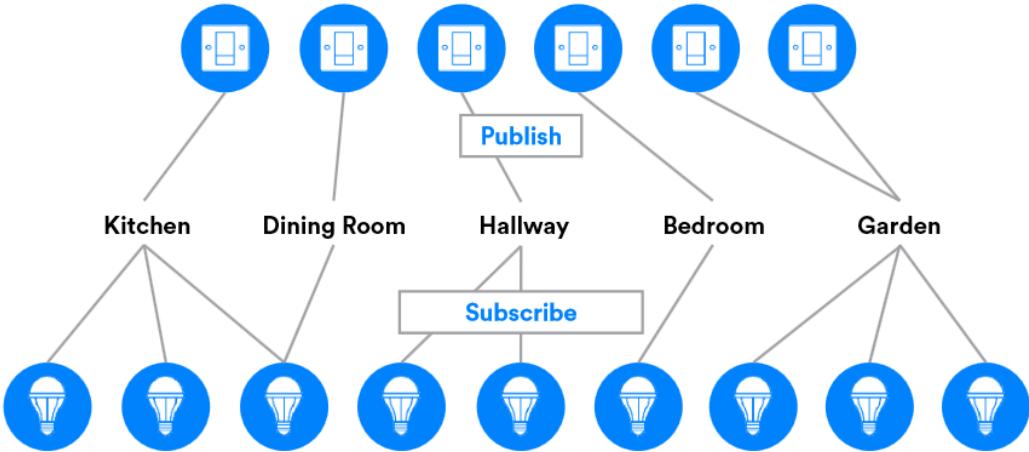


Figura 16: Publish/Subscribe.[7]

attuale associate. Queste due proprietà consentono a un sensore di pubblicare le letture in un modo che un client possa determinare il contesto in cui riceve il valore della temperatura. Le proprietà sono organizzate in due categorie: **produttore**, che è di sola lettura e **amministratore** che consente l'accesso in lettura e scrittura.

Messaggi, Stati e Proprietà

I **messaggi** sono il meccanismo mediante il quale vengono invocate le operazioni sulla mesh. I messaggi sono di tre tipi: **GET**, **SET** e **STATUS**. I messaggi GET richiedono il valore di un determinato stato da uno o più nodi. Un messaggio STATUS viene inviato in risposta a un GET e contiene il relativo valore di stato. I messaggi SET cambiano il valore di un determinato stato. Un messaggio SET riconosciuto(acknowledged) comporterà la restituzione di un messaggio STATUS in risposta, mentre un messaggio SET non riconosciuto(unacknowledged) non richiede alcuna risposta. I messaggi STATUS vengono inviati in risposta a messaggi GET, messaggi SET riconosciuti o indipendentemente da altri messaggi, ad esempio secondo un timer. Gli stati specifici a cui fanno riferimento i messaggi vengono dedotti dal codice operativo(opcode) del messaggio. Le proprietà sono referenziate esplicitamente nei messaggi generici usando un ID proprietario a 16 bit.

Modelli

I **modelli** riuniscono i concetti precedenti e definiscono alcune o tutte le funzionalità di un elemento in relazione alla rete mesh. Vengono riconosciute tre categorie di modelli. Un modello **server** definisce una raccolta di stati, transizioni di stato, associazioni di stato e messaggi che l'elemento che contiene

il modello può inviare o ricevere. Definisce inoltre comportamenti relativi a messaggi, stati e transizioni di stato. Un modello **client** non definisce alcuno stato. Al contrario, definisce i messaggi che può inviare o ricevere per fare il GET, SET o acquisire lo STATUS degli stati definiti nel modello server corrispondente. I modelli di **controllo** contengono sia un modello server, che consente la comunicazione con altri modelli client sia un modello client che consente la comunicazione con i modelli di server. I modelli possono essere creati estendendo altri modelli.

Messaggi, stati e modelli generici

Diversi tipi di dispositivi, spesso hanno stati semanticamente equivalenti, come ad esempio ON e OFF per luci, ventole e prese di corrente, che possono essere accese o spente. Di conseguenza, le specifiche del modello definiscono una serie di **stati generici** riutilizzabili come, ad esempio, Generic OnOff e Generic Level. Analogamente, vengono definite una serie di **messaggi generici** che operano sugli stati generici, come ad esempio Generic OnOff Get e Generic Level Set. Gli stati e i messaggi generici vengono utilizzati nei **modelli generici**, come il Generic OnOff Server e Generic Client Model. I messaggi, gli stati e i modelli generici consentono a un'ampia gamma di dispositivi di supportare la mesh senza la necessità di creare nuovi modelli.

Provisioning

Il **provisioning** è il processo mediante il quale un dispositivo si unisce alla rete mesh e diventa un nodo. Comprende diverse fasi, il cui risultato è la generazione di varie chiavi di sicurezza ed è esso stesso un processo sicuro. Il dispositivo utilizzato per il processo di provisioning viene definito *provisioner*. Il processo di provisioning procede attraverso cinque passaggi e questi sono descritti di seguito.

Step 1. Beaconing

A supporto di diverse funzioni del Bluetooth Mesh sono stati introdotti nuovi tipi di advertising GAP, incluso il *Mesh Beacon AD*. Un dispositivo non sottoposto a provisioning indica che è possibile eseguirlo utilizzando il tipo Mesh Beacon AD nei pacchetti di advertising.

Step 2. Invito

In questo passaggio, il provisioner invia un invito al dispositivo da sottoporre a provisioning, sotto forma di una Protocol Data Unit (PDU) chiamata Provisioning Invite. Il dispositivo che invia beacon risponde con le informazioni su se stesso tramite una PDU chiamata *Provisioning Capabilities*.

Step 3. Scambio di chiavi pubbliche

Il provisioner e il dispositivo da sottoporre a provisioning scambiano le loro chiavi pubbliche, direttamente o utilizzando un metodo out-of-band (OOB).

Step 4. Autenticazione

Durante la fase di autenticazione, il dispositivo da sottoporre a provisioning trasmette all'utente un numero casuale. L'utente inserisce le cifre nel provisioner e si verifica uno scambio di chiavi crittografate tra i due dispositivi, che coinvolge il numero casuale, per completare l'autenticazione di ciascuno dei due dispositivi all'altro.

Step 5. Distribuzione dei dati di provisioning

Una volta completata correttamente l'autenticazione, una *chiave di sessione* viene derivata da ciascuno dei due dispositivi dalle rispettive chiavi private e dalle chiavi pubbliche scambiate. La chiave di sessione viene quindi utilizzata per proteggere la successiva distribuzione dei dati richiesti per completare il processo di provisioning, inclusa una chiave di sicurezza nota come *chiave di rete* (NetKey). Una volta completato il provisioning, il dispositivo sottoposto a provisioning possiede la NetKey della rete, un parametro di sicurezza mesh, noto come *IV Index*, e un indirizzo unicast, assegnato dal provisioner. Ora il dispositivo è diventato un nodo.

Features

Tutti i nodi possono trasmettere e ricevere messaggi ma ci sono una serie di funzioni opzionali che un nodo può possedere, dandogli ulteriori funzionalità speciali. Esistono quattro di queste funzioni opzionali: **Relay**, **Proxy**, **Friend** e **Low Power**. Un nodo può non supportare o supportare una o più di queste funzionalità opzionali che, in un determinato momento, può essere abilitata o disabilitata.

Nodi Relay

I nodi che supportano la funzione di inoltro, noti come nodi **Relay**, sono in grado di ritrasmettere i messaggi ricevuti. L'inoltro è il meccanismo attraverso il quale un messaggio può attraversare l'intera rete mesh, facendo più "salti" (*hop*) tra i dispositivi. Le PDU della mesh includono un campo chiamato TTL (*Time To Live*) che contiene un valore intero e che viene utilizzato per limitare il numero di hop che un messaggio eseguirà attraverso la rete. Impostando il TTL a 3, ad esempio, il messaggio verrà inoltrato, a un numero massimo di tre hop dal nodo di origine. Impostandolo a 0, non verrà affatto trasmesso e viaggerà solo su un singolo hop. I nodi possono utilizzare il campo TTL per utilizzare in modo più efficiente la rete mesh.

Nodi Low Power e Nodi Friend

Alcuni tipi di nodo hanno una fonte di alimentazione limitata e devono conservare il più possibile l'energia. Dispositivi di questo tipo possono essere principalmente interessati all'invio di messaggi ma possono avere la neces-

sità di ricevere occasionalmente dei messaggi. Si consideri, ad esempio, un sensore di temperatura che è alimentato da una piccola batteria a bottone. Il sensore invia una lettura della temperatura una volta al minuto quando questa è al di sopra o al di sotto di alcune soglie configurate e se la temperatura rimane entro quelle soglie non invia messaggi. Questi comportamenti possono essere facilmente implementati senza particolari problemi relativi al consumo di energia, tuttavia l'utente è anche in grado di inviare messaggi al sensore che modificano i valori di stato della soglia di temperatura. La necessità di ricevere messaggi ha implicazioni sul duty cycle di funzionamento e sul consumo di energia. Un duty cycle al 100% garantirebbe che il sensore non perda alcun messaggio di configurazione ma che utilizzi una alta quantità di potenza. Un basso duty cycle risparmierebbe energia ma rischierebbe di perdere messaggi di configurazione del sensore. La risposta a questo problema è il nodo **Friend** e il concetto di *friendship*. Nodi come il sensore di temperatura nell'esempio possono essere designati come nodi **Low Power** (LPN). I nodi LPN lavorano con altri nodi con sorgenti di alimentazione fissa, ad esempio, l'alimentazione di rete. Questo dispositivo è chiamato nodo Friend e memorizza i messaggi indirizzati al LPN e li consegna ogni volta che questo interroga il nodo Friend. Il concetto di friendship è la chiave per consentire a nodi che devono ricevere messaggi, ma con alimentazione fortemente limitata, di funzionare in modo efficiente dal punto di vista energetico.

Nodi Proxy

I nodi **Proxy** espongono un'interfaccia GATT che i dispositivi Bluetooth LE possono utilizzare per interagire con una rete mesh. Viene definito un protocollo, chiamato *Proxy Protocol*, destinato ad essere utilizzato con un bearer orientato alla connessione, come il GATT. I dispositivi GATT leggono e scrivono PDU con protocollo proxy all'interno delle caratteristiche GATT implementate dal nodo proxy che trasforma queste PDU in PDU mesh o viceversa. In breve, i nodi proxy consentono ai dispositivi Bluetooth LE che non possiedono uno stack Bluetooth Mesh di interagire con i nodi in una rete mesh.

Configurazione dei nodi

Ciascun nodo supporta un set standard di **stati di configurazione** che sono implementati all'interno del Configuration Server Model e sono accessibili tramite il Configuration Client Model. I dati del Configuration State riguardano le capacità e il comportamento del nodo all'interno della mesh. Una serie di messaggi di configurazione consente al Configuration Client Model e al Configuration Server Model di supportare GET, SET e STATUS negli stati del Configuration Server Model.

1.4.2 Architettura del Bluetooth Mesh

In questa sezione verrà esaminata l'architettura del Bluetooth Mesh. Nella parte inferiore dello stack dell'architettura mesh, come mostrato nella figura 15, si ha un layer chiamato Bluetooth LE. In realtà, questo non è solo un singolo layer dell'architettura mesh, è lo stack Bluetooth LE completo, necessario per fornire funzionalità di comunicazione fondamentali sfruttate dall'architettura mesh che si trova al di sopra di essa: il sistema mesh dipende quindi dalla disponibilità di uno stack BLE. Verranno ora descritti ogni layer dell'architettura mesh, partendo da quello più basso.

Bearer Layer

I messaggi mesh richiedono un sistema di comunicazione sottostante per la loro trasmissione e ricezione. Il layer bearer definisce come le PDU mesh verranno gestite da un determinato sistema di comunicazione. Vengono definiti due bearer chiamati **Advertising Bearer** e **GATT Bearer**. L'Advertising Bearer sfrutta le funzioni di advertising e scansione GAP del Bluetooth LE per trasmettere e ricevere PDU mesh. Il GATT Bearer consente a un dispositivo che non supporta l'Advertising Bearer di comunicare indirettamente con nodi di una rete mesh utilizzando il protocollo proxy. Un nodo Proxy implementa queste caratteristiche GATT e supporta il GATT Bearer e l'Advertising Bearer in modo che possa convertire e inoltrare messaggi tra i due tipi di bearer.

Network Layer

Il livello di rete definisce i vari tipi di indirizzo di messaggio e un formato di messaggio di rete che consente alle PDU del livello di trasporto di essere trasportate dal relativo layer. Può supportare più bearer, ognuno dei quali può avere più interfacce di rete, inclusa l'interfaccia locale utilizzata per la comunicazione tra elementi che fanno parte dello stesso nodo. Il livello di rete determina le interfacce di rete su cui trasmettere i messaggi. Un filtro di input e di output viene applicato rispettivamente ai messaggi di input o output dal layer bearer per determinare se devono essere recapitati o meno al layer di rete per ulteriori elaborazioni.

Lower Transport Layer

Il livello di trasporto inferiore prende le PDU dal livello di trasporto superiore e le invia a quello inferiore. Laddove richiesto, esegue la segmentazione e il riassemblaggio di PDU. Per pacchetti più lunghi, che non rientrano in una singola PDU di trasporto, il livello di trasporto inferiore eseguirà la segmentazione, suddividendo la PDU in più PDU di trasporto. Il layer di trasporto

inferiore del dispositivo ricevente riassemblerà i segmenti in una singola PDU del layer di trasporto superiore.

Upper Transport Layer

Il livello di trasporto superiore è responsabile della crittografia, della decrittografia e dell'autenticazione dei dati che passano da e verso il livello di accesso. Gestisce anche i messaggi di controllo del trasporto, che sono generati internamente e inviati tra i livelli di trasporto superiori su diversi nodi.

Access Layer

Il livello di accesso è responsabile della definizione di come le applicazioni possono utilizzare il livello di trasporto superiore. Ciò comprende:

- Definizione del formato dei dati dell'applicazione.
- Definizione e controllo del processo di crittografia e decrittografia che viene eseguito nel livello di trasporto superiore.
- Verifica che i dati ricevuti dal livello di trasporto superiore siano per la rete e l'applicazione giuste, prima di inoltrare i dati nello stack.

Foundation Models Layer

Il layer dei modelli di base è responsabile dell'implementazione di quei modelli relativi alla configurazione e alla gestione di una rete mesh.

Models Layer

Il livello dei modelli riguarda l'implementazione dei modelli e dei comportamenti, messaggi, stati, ecc. come definiti in una o più specifiche del modello.

1.5 Introduzione al BLE IOT

L'SDK nRF5 consente ai dispositivi nRF5 di connettersi e comunicare con altri dispositivi tramite Internet utilizzando il Bluetooth low energy (BLE). Fornendo driver, librerie, esempi e API, l'SKD è uno strumento universale per iniziare con l'Internet of Things (IoT).

Tutti i dispositivi nell'Internet of Things devono essere identificabili in modo univoco, in modo che la comunicazione diretta tra i dispositivi sia possibile e ogni dispositivo possa essere indirizzato individualmente. Un modo per realizzare questo è quello di assegnare un indirizzo IPv6 unico per ogni dispositivo e gestire tutte le comunicazioni tramite IPv6.

Questo SDK fornisce i mezzi per utilizzare i collegamenti BLE per collegare i dispositivi IoT a un router abilitato BLE, che a sua volta è collegato

a Internet tramite IPv6. A tutti i dispositivi IoT vengono assegnati indirizzi IPv6 individuali, e viene utilizzato il link BLE per trasmettere i pacchetti IPv6. In questo modo, l'SDK astrae le applicazioni dalla tecnologia BLE e fornisce un modo per indirizzare direttamente tutti i dispositivi che utilizzano IPv6.

Ogni dispositivo IoT può comunicare direttamente con altri dispositivi utilizzando il loro indirizzo IPv6; non importa se l'altro dispositivo è collegato allo stesso router o si trova da qualche altra parte su Internet, e non importa se l'altro dispositivo utilizza o meno un link BLE. Inoltre, tutte le applicazioni possono utilizzare lo stesso protocollo (ad esempio MQTT, CoAP, o anche HTTP) per comunicare, il che significa che tipi eterogenei di dispositivi - cablati, wireless, o in cloud - possono parlare un linguaggio comune.

La figura 17 illustra una rete IoT con dispositivi che sono connessi ad Internet tramite router abilitati al BLE.

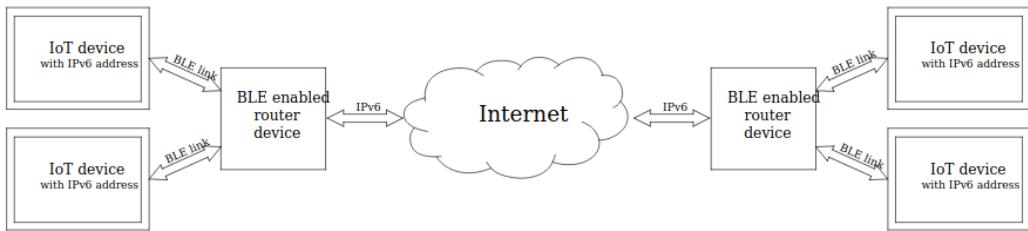


Figura 17: Rete IoT usando collegamenti BLE [3]

1.6 Installare l'ambiente di sviluppo e scaricare l'SDK Nordic

1. Installare IDE Segger Embedded Studio dal seguente link.
2. Scaricare nRF5 SDK dal seguente link ed estrarre il file zip nella directory che si desidera utilizzare per lavorare con l'SDK.
3. Scaricare nRF5 SDK for Mesh dal seguente link ed estrarre il file zip nella directory che si desidera utilizzare per lavorare con l'SDK. consigliato estrarre il contenuto nella stessa cartella del SDK scaricato al punto 2.

1.7 Compilare gli esempi dello stack mesh

1.7.1 Primo setup

SEGGER Embedded Studio (SES) fornisce funzionalità per compilare rapidamente il codice di esempio con la possibilità di debuggare il firmware.

Prima di compilare gli esempi del Bluetooth Mesh con SEGGER Embedded Studio per la prima volta, è necessario completare una configurazione della macro `SDK_ROOT` in SEGGER Embedded Studio. Questa macro viene utilizzata per trovare i file SDK nRF5.

È possibile:

- Utilizzare le impostazioni predefinite della macro `SDK_ROOT`. L'impostazione predefinita è un'istanza SDK nRF5 decompressa proprio accanto alla cartella mesh.
- Settare la macro `SDK_ROOT` a una istanza personalizzata.

Per settare la macro `SDK_ROOT` manualmente in SEGGER Embedded Studio:

1. Andare su **Tools** → **Options**.
2. Selezionare **Building**.
3. In **Build** nella lista delle configurazioni, modificare **Global macros** per contenere `SDK_ROOT=<percorso dell'SDK nRF5>`.
4. Salvare la configurazione.

È possibile verificare il percorso aprendo uno dei file sorgenti nel gruppo di file SDK nRF5. Se la macro è impostata correttamente, il file si apre nella finestra dell'editor. In caso contrario, viene visualizzato un messaggio di errore con informazioni che non è possibile trovare il file.

1.7.2 Compilare con SES

Di default, l'SDK nRF5 per Mesh include i file di progetto SES per tutti gli esempi. Ciò consente di iniziare rapidamente a compilare esempi con SES.

Comunque, se si fanno cambiamenti ad uno dei file in `CMakeLists.txt`, è necessario generare un nuovo progetto SES usando CMake.

Per compilare SEGGER Embedded Studio:

1. Aprire il progetto desiderato nella cartella `examples/`, per esempio `examples/light_switch/client/light_switch_client_nrf52832_xxAA_s132_6_1_1.emProject`.

2. Cliccare su **Build** → **Build < name of the emProject file>**, per esempio **Build light_switch_client_nrf52832_xxAA_s132_7.0.1**.
3. Aspettare il termine della compilazione.

È ora possibile eseguire gli esempi usando SEGGER Embedded Studio.

1.7.3 Eseguire gli esempi usando SEGGER Embedded Studio

La seguente procedura funziona solo se è stato compilato l'esempio SEGGER Embedded Studio.

Per eseguire gli esempi in un ambiente di sviluppo basato su SEGGER Embedded Studio (SES):

1. Connetter il Kit di Sviluppo con il cavo USB al computer.
2. In SES, connettere il kit di sviluppo con **Target** → **Connect J-Link**.
3. Cancellare il dispositivo dal menu delle opzioni SES: **Target** → **Erase all**.
4. Eseguire l'esempio con **Debug** → **Go**. Questo scarica il SoftDevice adatto e l'esempio compilato, e inizia il debugger.
5. Quando il download è completo, selezionare **Debug** → **Go** ancora per iniziare l'esecuzione del codice.

Se il debugger non parte, resettare il J-Link: **Target** → **Reset J-Link**.

2 Esempio Light Switch

Questo esempio mostra l'ecosistema mesh che contiene dispositivi che agiscono in due ruoli: Provisioner e Node (anche indicato come provisionee). Dimostra inoltre come usare i modelli Mesh utilizzando il modello OnOff generico in un'applicazione. L'esempio è composto da tre esempi minori:

- Light switch server: un server minimalista che implementa un modello server OnOff generico, che viene utilizzato per ricevere i dati sullo stato e controllare il LED 1 sulla scheda.
- Light switch client: un client minimalista che implementa quattro istanze di un modello client OnOff generico. Quando un utente preme uno qualsiasi dei pulsanti, viene inviato un messaggio OnOff Set all'indirizzo di destinazione configurato.
- Mesh Provisioner: una semplice implementazione di provisioner statico che imposta la rete e fa il provisioning di tutti i nodi in una rete mesh. Inoltre, il provisioner configura anche le combinazioni di tasti e le impostazioni di publication e di subscription delle istanze di modelli mesh su questi nodi per consentire di parlare tra di loro.

| Nota

Ai fini del provisioning, è possibile utilizzare l'esempio del provisioner statico oppure utilizzare l'app nRF Mesh.

Il client/server generic OnOff viene utilizzato per manipolare lo stato on/off. Si noti che quando il server ha un indirizzo di pubblicazione impostato (come nell'esempio), il server pubblicherà qualsiasi operazione della sua modifica di stato sul proprio indirizzo di pubblicazione.

La seguente figura 18 mostra la visione d'insieme della rete mesh che sarà impostata dallo static provisioner. I numeri tra parentesi indicano gli indirizzi che sono assegnati ai nodi dal provisioner.

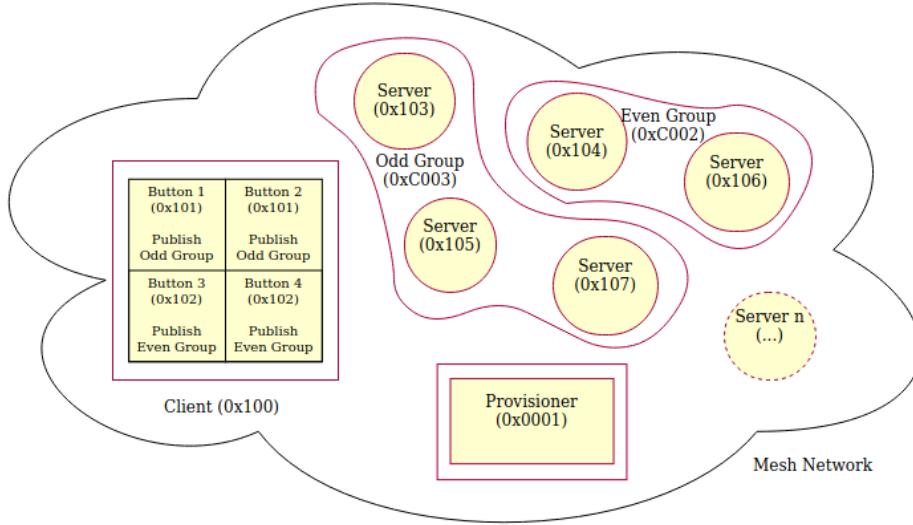


Figura 18: Dimostrazione della rete Mesh [4]

Sia il light switch server che il light switch client hanno un ruolo di provisionee. Supportano il provisioning su Advertising bearer (PB-ADV) e GATT bearer (PB-GATT) e supportano anche il Mesh Proxy Service (Server).

2.1 Requisiti Hardware

Sono necessarie almeno due schede per questo esempio:

- Una scheda di sviluppo nRF5 per il client.
- Una o più schede di sviluppo nRF5 per i server.

Inoltre, è necessario una scheda tra le seguenti:

- Una scheda di sviluppo nRF5 per il provisioner se si decide di utilizzare l'esempio del provisioner statico.
- L'app nRF Mesh (iOS o Android) se si decide di fare il provisioning usando l'applicazione.

2.2 Setup

Si può trovare il codice sorgente di questo esempio nella seguente cartella:
`\examples\light_switch`

2.3 LED e assegnazioni dei pulsanti

I pulsanti (1 a 4) sono utilizzati per avviare alcune azioni, e i LED (1 a 4) sono utilizzati per riflettere lo stato delle azioni come segue:

- Server:
 - Durante il processo di provisioning:
 - * LED 3 e 4 lampeggianti: identificazione del dispositivo attiva.
 - * LED 1 to 4: lampeggia quattro volte per indicare il processo di provisioning è completato.
 - Dopo che il provisioning e la configurazione sono finiti:
 - * LED 1: Riflette il valore dello stato OnOff sul server.
 - LED ON: Il valore dello stato OnOff è 1 (vero).
 - LED OFF: Il valore dello stato OnOff è 0 (falso).
- Client:
 - Durante il processo di provisioning:
 - * LED 3 e 4 lampeggianti: l'identificazione del dispositivo è attiva.
 - * LED da 1 a 4: lampeggia quattro volte per indicare il processo di provisioning è completato.
 - Dopo che il provisioning e la configurazione sono finiti, i pulsanti sul client sono utilizzati per inviare messaggi OnOff Set ai server:
 - * Pulsante 1: Invia un messaggio al gruppo dispari (indirizzo: 0xC003) per accendere il LED 1.
 - * Pulsante 2: Invia un messaggio al gruppo dispari (indirizzo: 0xC003) per spegnere il LED 1.
 - * Pulsante 3: Invia un messaggio al gruppo pari (indirizzo: 0xC002) per accendere il LED 1.
 - * Pulsante 4: Invia un messaggio al gruppo pari (indirizzo: 0xC002) per spegnere il LED 1.
- Provisioner:
 - Pulsante 1: Inizio del provisioning.
 - LED 1: Riflette lo stato del provisioning.
 - * LED ON: Il provisioning del nodo è in corso.

- * LED OFF: Nessun processo di provisioning in corso.
- LED 2: Riflette lo stato della configurazione.
 - * LED ON: La configurazione del nodo è in corso.
 - * LED OFF: Nessun processo di configurazione in corso.

2.4 Test con il provisioner statico

1. Flashare gli esempi seguendo le istruzioni nel capitolo 1.7.3, compreso:
 - (a) Cancellare la memoria flash delle schede di sviluppo e programmare il SoftDevice.
 - (b) Flashare il firmware provisioner e client sulle singole schede e il firmware del server su altre schede.
2. Dopo il reset, alla fine del processo di flash del firmware, premere il pulsante 1 sulla scheda provisioner per avviare il processo di provisioning:
 - Il provisioner fa inizialmente il provision e configura il client e assegna l'indirizzo 0x100 al nodo.
 - Le due istanze dei modelli client OnOff sono istanziate su elementi secondari separati. Per questo motivo, ottengono consecutivi indirizzi a partire da 0x101.
 - Infine, il provisioner rileva i dispositivi disponibili, assegna loro indirizzi consecutivi, li aggiunge a gruppi dispari e pari.
3. Osservare che il LED 1 sulla provisioner board è acceso quando il provisioner sta facendo la scansione e il provisioning di un dispositivo.
4. Osservare che il LED 2 sulla provisioner board è acceso quando la procedura di configurazione è in corso.
5. Attendere che il LED 1 sulla provisioner board rimanga acceso costantemente per pochi secondi, questo indica che tutte le schede disponibili sono state provisionate e configurate.

Se il provisioner rileva un errore durante il provisioning o il processo di configurazione di un certo nodo, è possibile resettare il provisioner per riavviare il processo per il nodo il questione.

2.5 Test con l'app nRF Mesh

Per testare l'esempio con lo smartphone è necessario scaricare l'app nRF Mesh. Quando l'applicazione finisce la scansione si possono vedere i due dispositivi client e server come in figura 19

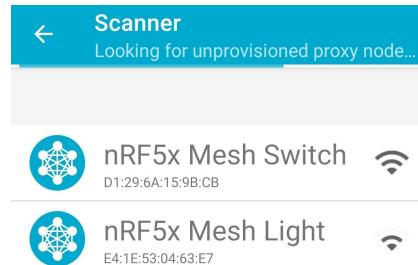


Figura 19: Fase di scanning dei dispositivi

Cliccando ad esempio su nRF5x Mesh Switch si ottiene la seguente schermata

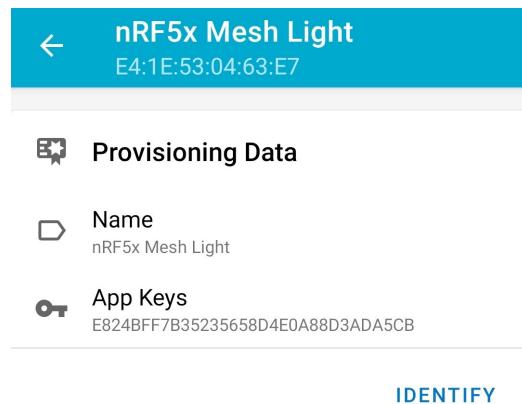


Figura 20: Provisioning Data

Cliccando su **IDENTIFY**, compaiono poi l'elenco delle **capabilities** del nodo. A questo punto cliccare su **PROVISION** per fare il provisioning del nodo.

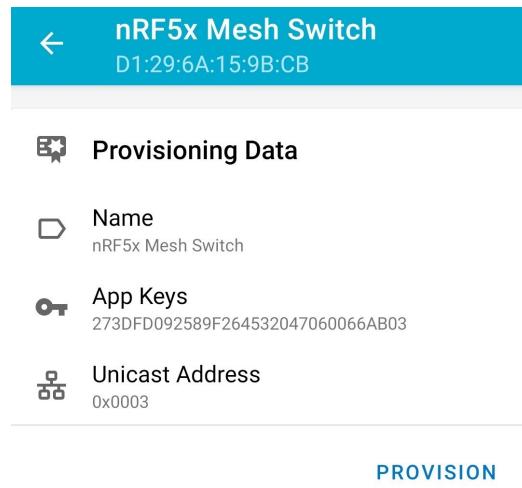


Figura 21: Provision del nodo

I nodi ora configurati e si ottiene la seguente schermata

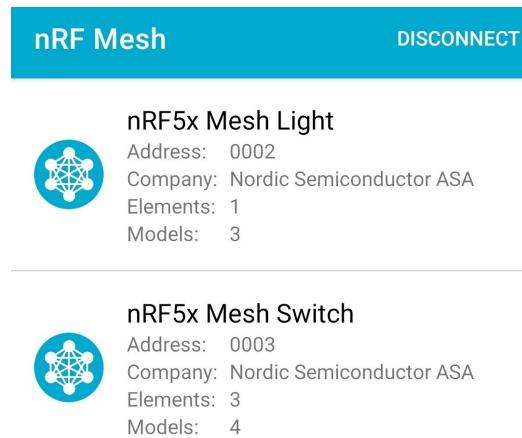


Figura 22: Nodi configurati

Per accendere o spegnere il led è necessario associare un chiave. Cliccare quindi su **BIND KEY**

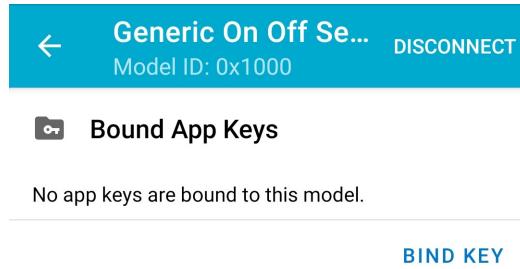


Figura 23: App key

La figura 24 mostra la schermata finale in cui si può leggere lo stato del nodo e accendere o spegnere il led.

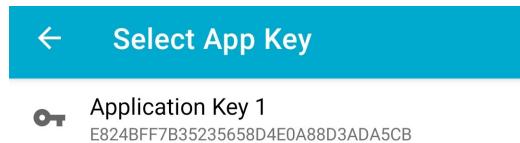


Figura 24: Application key

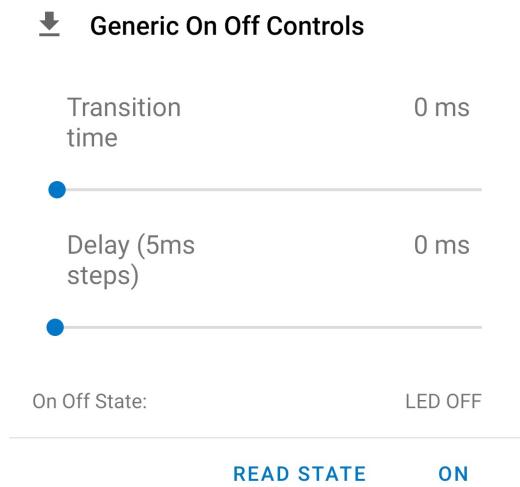


Figura 25: Controlli On Off generici

2.6 Interagire con le schede

Una volta che il provisioning e la configurazione del nodo client e almeno uno dei nodi server sono completi, è possibile premere i pulsanti sul client per vedere i LED commutare sui server associati. Guarda il capitolo 2.3

Se un terminale RTT è disponibile e collegato al client, inviando i numeri ASCII 0–3 si avrà lo stesso effetto della pressione dei pulsanti.

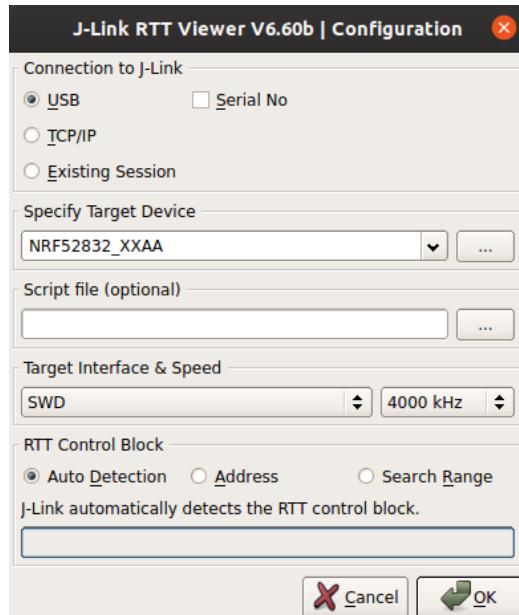
Se si utilizza RTT log, è anche possibile premere il pulsante 1 sul server per commutare localmente lo stato del proprio LED 1, e lo status che riflette questo stato verrà inviato alla scheda client. È possibile visualizzare lo status visualizzato nell'RTT log della scheda client.

Se uno qualsiasi dei dispositivi viene spento e riaccesso, questo si ricorderà la sua configurazione e si ricongiungerà alla rete.

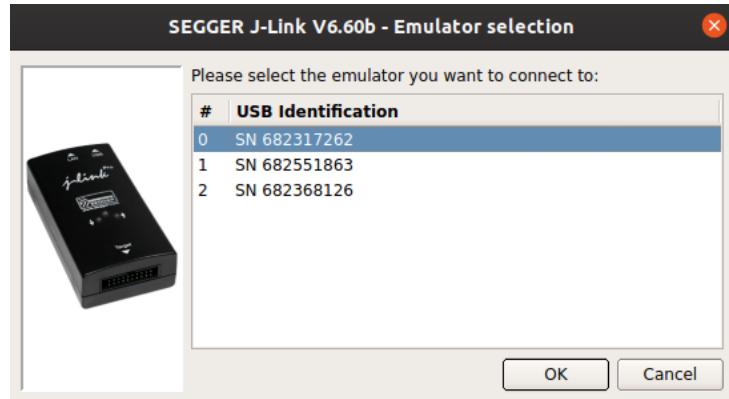
2.7 J-Link RTT Viewer

Si può usare J-Link RTT Viewer per debuggare il codice. J-Link RTT Viewer è una applicazione con interfaccia grafica che permette di usare tutte le caratteristiche RTT nel dispositivo che sta eseguendo il debug (Windows, macOS e Linux).

La figura seguente mostra la finestra principale quando viene aperta l'applicazione. Si può selezionare il dispositivo con cui interfacciarsi tramite l'opzione Specify Target Device.



L'applicazione chiede a quale scheda collegarsi, le tre schede usate per il progetto hanno i codici seriale mostrati in figura



Viene ora presentato un esempio di uso dell'applicazione con la scheda che funge da provisioner. Si può vedere che quando viene premuto il pulsante 1 inizia il provisioning. Viene configurato il client e gli viene assegnato l'indirizzo 0x0100.

J-Link RTT Viewer V6.60b

```

File  Terminals  Input  Logging  Help
Log  All Terminals  Terminal 0
00> <: main.C: 510, ----- BLE Mesh Light Switch Provisioner Demo -----
00> <: 11867>, main.C: 437, Initializing and adding models
00> <: 11904>, main.C: 390, Restored: App
00> <: 11906>, main.C: 500, Restored: Handles
00> <: 11908>, provisioner_helper.c: 332, m_netkey_handle:0 m_appkey_handle:0 m_self_devkey_handle:1
00> <: 11913>, main.C: 539, <start>
00> <: 11914>, main.C: 539, Starting application ...
00> <: 12009>, main.C: 529, Provisioned Nodes: 0 Next Address: 0x0100
00> <: 12013>, main.C: 530, Dev key : 66EA8D7686F1ED64ECCBA750201FA938
00> <: 12016>, main.C: 531, Net key : 6BF3C94E2705875B3D80849896361D3225
00> <: 12019>, main.C: 532, App key : FAED88AA3329E3F87EE64562BC9F271B
00> <: 12022>, main.C: 533, Press Button 1 to start provisioning configuration process.
00> <: 1012615>, main.C: 397, Button 1 pressed
00> <: 1012618>, main.C: 397, Waiting for Client node to be provisioned ...
00> <: 1012620>, provisioner_helper.c: 109, Scanning For Unprovisioned Devices
00> <: 1024526>, provisioner_helper.c: 138, UUID : 4741A8FE0602914AE8B98156A2911D660
00> <: 1024529>, provisioner_helper.c: 142, *RSSI:
00> <: 1024532>, provisioner_helper.c: 147, URI Hash: BE015706
00> <: 1024536>, provisioner_helper.c: 153, URI hash matched. Provisioning ...
00> <: 1032961>, provisioner_helper.c: 274, Provisioning link established
00> <: 1032975>, provisioner_helper.c: 280, Provisioning link established
00> <: 1060217>, provisioner_helper.c: 297, Provisioning completed successfully
00> <: 1060281>, provisioner_helper.c: 212, Adding device address, and device keys
00> <: 10669301>, provisioner_helper.c: 229, Addr: 0x0100 addr_handle: 0 netkey_handle: 0 devkey_handle: 2
00> <: 1066373>, provisioner_helper.c: 166, Local provisioning link closed: prov_state: 2 remaining retries: 2
00> <: 1066380>, main.C: 223, Provisioning
00> <: 1066385>, provisioner_helper.c: 196, Provisioning complete. Node addr: 0x0100 elements: 3
00> <: 1066391>, node_setup.c: 689, Config client event Node: 0x0100
00> <: 1066392>, node_setup.c: 592, Config client setup: devkey_handle:2 addr_handle:0
00> <: 1066395>, node_setup.c: 352, Getting composition data
00> <: 1066990>, main.C: 275, Config client event
00> <: 10669903>, node_setup.c: 362, Updating network transmit: count: 2 steps: 1
00> <: 1072761>, main.C: 275, Config client event
00> <: 1072763>, node_setup.c: 372, Adding
00> <: 1077880>, main.C: 275, Config client event
00> <: 1077883>, node_setup.c: 267, opcode status field: 0
00> <: 1077889>, node_setup.c: 383, App key bind: Health server
00> <: 1105738>, main.C: 275, Config client event
00> <: 1105740>, node_setup.c: 267, opcode status field: 0

```

RTT Viewer connected. 0.007 MB

3 Applicazione Eddystone Beacon

Importante: Prima di eseguire questo esempio, assicurarsi di programmare il SoftDevice.

Compilando l'esempio per la prima volta si ottiene il seguente errore: micro_ecc_lib_nrf52.lib: No such file or directory. Come descritto nel seguente link micro_ecc backend, per risolverlo occorre:

1. Installare l'ultima versione della toolchain GCC per ARM. Si può usare ARM's Launchpad per cercare la toolchain per il proprio sistema operativo.
2. Assicurarsi che "make" sia installato (per esempio, MinGW per Windows o GNU Make per Linux).
3. Clona la repository GitHub micro-ecc in \external\micro-ecc\micro-ecc.
4. Entra nella sottocartella del SoC e della toolchain che stai usando per compilare la tua applicazione:
 - \external\micro-ecc\nrf52_keil\armgcc
 - \external\micro-ecc\nrf52_iar\armgcc
 - \external\micro-ecc\nrf52_armgcc\armgcc (come nel progetto)
5. Modificare nel file Makefile posix nel percorso \components\toolchain\gcc la riga GNU_INSTALL_ROOT con il proprio percorso di installazione della toolchain arm, come ad esempio \gcc-arm-none-eabi-9-2019-q4-major\bin\
6. Eseguire make per compilare la libreria the micro-ecc.

L'esempio Eddystone Beacon può essere usato per trasformare il Kit di sviluppo nRF5 in un Beacon Eddystone. L'applicazione è intesa da utilizzare senza modifiche. Quando è in esecuzione, il beacon può essere configurato tramite Eddystone Configuration Service. La figura 26 mostra l'UUID del servizio che permette di configurare i beacon Eddystone.

Eddystone Configuration Service
UUID: a3c87500-8ed3-4bdf-8a39-a01bebede295
PRIMARY SERVICE

Figura 26: Eddystone Configuration Service

3.1 Protocollo Eddystone

Eddystone è un formato "open beacon" progettato da Google che definisce un formato di messaggio Bluetooth Low Energy(BLE) per i beacon di prossimità. Vedere il repository GitHub Eddystone per le specifiche del protocollo, la specifica del servizio Eddystone Configuration e ulteriori informazioni. L'applicazione di esempio supporta tutte le caratteristiche definite nelle specifiche ad eccezione di intervalli di advertising variabili.

Il protocollo Eddystone descrive diversi formati per i pacchetti di advertising, chiamati frame type, che possono essere utilizzati per creare beacon. Sono disponibili i seguenti tipi di frame:

- **Eddystone-UID** per la trasmissione di unica di un ID beacon a 16 byte univoco
- **Eddystone-URL** per la trasmissione di un URL in un formato di codifica compresso
- **Eddystone-TLM** per la trasmissione di informazioni di telemetria riguardanti il beacon (cifrato o non cifrato)
- **Eddystone-EID** per la trasmissione di un identificativo criptato che cambia periodicamente

L'esempio fornisce cinque slot di advertising, e l'utente può configurare ogni slot per utilizzare uno qualsiasi di questi tipi di frame. Non ci deve essere più di uno slot per ogni frame di tipo Eddystone-TLM, ma tutti gli altri tipi di frame possono essere utilizzati in più di uno slot. La configurazione degli slot è fatta attraverso Eddystone Configuration Service.

3.2 Caratteristiche di sicurezza

Mentre il frame Eddystone-UID viene trasmesso non protetto, il frame Eddystone-EID fornisce protezione contro le vulnerabilità note del beacon.

Il frame Eddystone-EID randomizza l'ID del dispositivo del beacon, così come i dati di advertising criptati. Questo processo protegge contro "spoofing" e "malicious asset tracking", poiché il dato di advertising crittografato e mutevole nel tempo rende difficile falsificare o tracciare un Eddystone-EID.

Se un beacon è configurato per trasmettere il frame Eddystone-EID, il frame Eddystone-TLM è automaticamente crittografato. In caso contrario, la trasmissione delle informazioni renderebbe il dispositivo univocamente identificabile. Utilizzando i frame TLM crittografati garantisce anche l'integrità del messaggio, ciò significa che l'utente può fare affidamento sulle infor-

mazioni telemetriche inviate dal beacon previsto. Il frame Eddystone-TLM crittografato è spesso indicato come Eddystone-eTLM.

Si noti che non si dovrebbe mai utilizzare un tipo di frame Eddystone-UID insieme a Eddystone-EID, perché l'UID potrebbe essere utilizzato per bypassare la protezione di sicurezza.

3.3 Setup

È possibile trovare il codice sorgente e il file di progetto dell'esempio nella seguente cartella: \examples\ble_peripheral\ble_app_eddystone

Non è necessario modificare il codice di esempio per testare l'applicazione perché la maggior parte della configurazione è fatta attraverso il servizio di configurazione Eddystone quando il beacon è attivo e funzionante. Tuttavia, è possibile configurare alcune impostazioni di base in `es_app_config.h`. Prima di andare in produzione, assicurarsi di modificare le seguenti impostazioni:

- Il **nome del dispositivo** (`APP_DEVICE_NAME`) trasmesso quando il beacon è in modalità collegabile. Di default è: "nRF5x_Eddystone".
- Il **tipo di frame di default** (`DEFAULT_FRAME_TYPE`) usato per tutti i cinque slot di advertising prima che il beacon sia configurato. Di default è: Eddystone-URL.
- Il **codice di blocco** (`APP_CONFIG_LOCK_CODE`) necessario per configurare il beacon. Di default è: FFFFFFFFFFFFFFFFFFFF (32 caratteri F).

3.4 Testing

Per testare l'applicazione Eddystone Beacon, è necessario installare l'app Android nRF Beacon for Eddystone. È disponibile da Google Play e dalla repository GitHub nRF Beacon for Eddystone.

Prova l'applicazione Eddystone Beacon eseguendo le seguenti fasi:

1. Compilare e programmare l'applicazione.
2. Osservare che il LED 1 sulla scheda inizia a lampeggiare e che la scheda inizia a trasmettere un Eddystone-URL. È possibile leggere l'URL con qualsiasi scanner beacon compatibile con Eddystone.
3. Premere il pulsante 1 sulla scheda per mettere la scheda in modalità collegabile per 60 secondi. Osservare che il LED 3 è acceso.

4. Aprire l'app nRF Beacon for Eddystone, quindi toccare il pulsante "connect" nella scheda "update". L'elenco dei beacon Eddystone collegabili è visualizzato.
5. Selezionare il proprio dispositivo. Per impostazione predefinita, è chiamato "nRF5x_Eddystone".
6. Inserire il codice di blocco del produttore del beacon. Per impostazione predefinita, questo codice è FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF (32 caratteri F).
7. Osservare che, dopo aver inserito il codice di blocco corretto, l'applicazione legge tutti gli slot e visualizza le informazioni per ogni slot. Osservare inoltre che il LED 2 è acceso.
8. Verificare le caratteristiche supportate dal beacon. Ad esempio, configurare tutti gli slot come mostrato nella figura seguente.

ADV Slot Data ▼ ▲

UUID: a3c8750a-8ed3-4bdf-8a39-a01beb0de295
Properties: READ, WRITE
Value: Frame type: URL <0x10>
Tx power at 0m: -9 dBm
URL: <https://www.nordicsemi.com/>

Figura 27: ADV Slot Data

Quindi eseguire un reset e assicurarsi che lo stato predefinito del beacon sia resettato.

9. Testare la disabilitazione del blocco automatico:
 - (a) Nell'app, selezionare **Lock state** e impostare **Disable automatic relock**.
 - (b) Disconnettersi dal dispositivo e ripetere i passaggi da 3 a 5. Osservare che è ora possibile sbloccare il beacon senza inserire un codice di blocco

4 Applicazione Beacon Transmitter

L'applicazione Beacon Transmitter è un esempio che implementa un trasmettitore beacon utilizzando l'hardware fornito nel Development Kit nRF5.

Il beacon trasmette informazioni a tutti i dispositivi compatibili dei pacchetti di advertising le cui informazioni comprendono:

- Un UUID a 128-bit per identificare il trasmettitore del beacon.
- Un Major value arbitrario per una differenziazione grossolana dei beacons.
- Un Minor value arbitrario per una differenziazione fine dei beacons.
- Il valore RSSI del beacon misurato a 1 metro di distanza, che può essere utilizzato per stimare la distanza dal beacon.

È possibile trovare il codice sorgente e il file di progetto dell'esempio nella seguente cartella: \examples\ble_peripheral\ble_app_beacon

4.1 Configurare i valori Major e Minor

Questo esempio implementa una funzionalità opzionale che permette la modifica di valori di Major e Minor utilizzati nei pacchetti di advertisement senza necessità per ricompilare l'applicazione ogni volta. Per usare questa funzionalità, la macro USE_UICR_FOR_MAJ_MIN_VALUES deve essere definita durante la compilazione. Se viene fatto questo, l'applicazione utilizza il valore dell'UICR (User Information Configuration Register) situato all'indirizzo 0x10001080 per popolare i valori di Major e Minor. Ogni volta che i valori devono essere aggiornati, l'utente deve impostare l'UICR ad un valore desiderato utilizzando il tool nrfjprog e riavviare l'applicazione per portare le modifiche a compimento. L'ordinamento dei byte usato per decodificare il valore UICR è mostrato nell'immagine seguente:

Byte 3	Byte 2	Byte 1	Byte 0
Major Value - MSB	Major Value - LSB	Minor Value - MSB	Minor Value - LSB

Figura 28: Ordinamento dei byte di UICR

Guarda il capitolo Testare l'applicazione per maggiori informazioni su come usare questa funzionalità.

| Nota

Questa applicazione può essere usata come punto di partenza per scrivere una applicazione iBeacon. La procedura per creare una applicazione iBeacon è disponibile su mfi.apple.com. Dopo aver ottenuto le specifiche, questa applicazione può essere modificata per soddisfare i requisiti di iBeacon.

4.2 Testare l'applicazione

Per testare l'applicazione Beacon Transmitter Sample con l'nRF Connect, seguire le seguenti fasi:

1. Compilare e programmare l'applicazione. Osservare che lo stato di **BSP_INDICATE_ADVERTISING** sia indicato. Questo fa sì che il LED 1 lampeggia con periodo di 2 secondi e duty cycle del 10%.
2. Dopo aver iniziato la scansione in nRF Connect, osservare che il beacon viene trasmesso con il "device address" Bluetooth senza un "Device Name".
3. Clicca su Details sotto l'indirizzo del beacon per visualizzare il dato di advertisement completo.
4. Osservare che **Advertising Type** è 'Non-connectable' e il campo **Manufacturer Specific Data** del dato di advertising è il seguente:
59-00-02-15-01-12-23-34-45-56-67-78-89-9A-AB-BC-CD-DE-EF-F0-01-02-03-04-C3
1. Definire la macro **USE_UICR_FOR_MAJ_MIN_VALUES**, ricompiolare e caricare il firmware.
2. Usare il tool nrfjprog per scrivere il valore 0xabcd0102 nel registro UICR come di seguito:

nrfjprog -f nrf52 -snr <Segger-chip-Serial-Number> -memwr 0x10001080 -val 0xabcd0102
3. Resettare la scheda e osservare che i byte in grassetto di seguito siano impostati nel campo **Manufacturer Specific Data** del dato di advertising. Questo indica che i valori Major e Minor sono stati presi dal registro UICR scritto nella fase precedente.
XX-AB-CD-01-02-XX

5 Esempio Beaconing nella rete Mesh

Questo esempio mostra come eseguire simultaneamente il beaconing, che consente a un'applicazione di trasmettere beacon (ad esempio iBeacon o Eddy-stone) mentre partecipa alla rete mesh. Inoltre, dimostra l'utilizzo della callback RX.

5.1 Mandare Beacon

Per inviare i beacon, l'applicazione utilizza direttamente il packet manager interno alla mesh e la struttura dell'advertiser. L'applicazione prima inizializza l'advertiser (l'inizializzazione è necessaria una sola volta). Quindi alloca e riempie i campi del pacchetto. Dopo l'inizializzazione, l'applicazione pianifica il pacchetto per la trasmissione inserendolo nella coda TX dell'advertiser, con un parametro che indica il numero di ripetizioni che l'advertiser eseguirà. In questo esempio, il conteggio delle ripetizioni è impostato su BEARER_ADV_REPEAT_INFINITE, per cui il pacchetto viene ritrasmesso per sempre o fino a quando non viene sostituito da un altro pacchetto.

5.2 Callback RX

L'esempio beaconing mostra anche l'uso della callback RX. Questa funzionalità consente di ricevere tutti i pacchetti di advertising non filtrati e conformi a BLE nel codice dell'applicazione. Questi pacchetti vengono acquisiti dallo scanner. Il tipo di dato ble_packet_type_t elenca tutti i pacchetti di advertising che è possibile ricevere. Una volta che un nuovo pacchetto viene acquisito dallo scanner, viene passato attraverso il callback RX e fornito all'applicazione utente. Per ascoltare i pacchetti, l'esempio registra il callback RX chiamando nrf_mesh_rx_cb_set (). Come input, la funzione di callback RX porta un puntatore a una struttura di parametri che contiene tutti i dati disponibili sul pacchetto in entrata. La callback RX viene richiamata per tutti i pacchetti che vengono elaborati dalla mesh che presuppone che tutti i pacchetti in entrata aderiscano al formato del pacchetto di advertising Bluetooth Low Energy.

5.3 Requisiti software

Occorre installere nRF Connect in una delle seguenti versioni:

- nRF Connect for Desktop
- nRF Connect for Mobile

5.4 Setup

Il codice sorgente dell'esempio beaconing è contenuto nella seguente cartella:
\examples\beaconing

5.5 Testare l'esempio

Per testare l'esempio beaconing:

1. Compilare l'esempio seguendo le istruzioni in Compilare gli esempi dello stack mesh
2. Programma la scheda seguendo le istruzioni in Eseguire gli esempi usando SEGGER Embedded Studio.
3. Connotti RTT Viewer per vedere l'output RTT generato dall'esempio nel log RTT.

Una volta che l'esempio viene eseguito, vengono mostrati in output tutti i pacchetti ricevuti. I beacon trasmessi possono essere osservati tramite nRF Connect for Desktop o nRF Connect for Mobile.

6 Implementazione di beacon statici nell'esempio Light Switch Server

Verranno ora illustrate le varie prove effettuate per aggiungere dei beacon statici, cioè che non possono essere cambiati una volta che la scheda è stata programmata, all'interno dell'esempio Light Switch Server.

6.1 Integrazione dell'esempio Beacon Transmitter nel Light Switch Server

Nel file main.c dell'esempio Light Switch Server è stata aggiunta la funzione `advertising_init()` presa dall'applicazione Beacon Transmitter descritta nel capitolo 4 e definita come di seguito:

```
static void advertising_init(void)
{
    uint32_t          err_code;
    ble_advdata_t     advdata;
    uint8_t           flags =
        BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED;

    ble_advdata_manuf_data_t manuf_specific_data;

    manuf_specific_data.company_identifier =
        APP_COMPANY_IDENTIFIER;

    manuf_specific_data.data.p_data = (uint8_t *)
        m_beacon_info;
    manuf_specific_data.data.size    = APP_BEACON_INFO_LENGTH;

    // Build and set advertising data.
    memset(&advdata, 0, sizeof(advdata));

    advdata.name_type          = BLE_ADVDATA_NO_NAME;
    advdata.flags               = flags;
    advdata.p_manuf_specific_data = &manuf_specific_data;

    // Initialize advertising parameters (used when starting
    // advertising).
    memset(&m_adv_params, 0, sizeof(m_adv_params));
```

```

m_adv_params.properties.type =
    BLE_GAP_ADV_TYPE_NONCONNECTABLE_NONSCANNABLE_UNDIRECTED
;
m_adv_params.p_peer_addr = NULL; // Undirected
                                advertisement.
m_adv_params.filter_policy = BLE_GAP_ADV_FP_ANY;
m_adv_params.interval =
    NON_CONNECTABLE_ADV_INTERVAL;
m_adv_params.duration = 0; // Never time out
.

err_code = ble_advdata_encode(&advsdata, m_adv_data.
    adv_data.p_data, &m_adv_data.adv_data.len);
ERROR_CHECK(err_code);

err_code = sd_ble_gap_adv_set_configure(&m_adv_handle, &
    m_adv_data, &m_adv_params);
ERROR_CHECK(err_code);
}

```

Questa funzione inizializza le funzionalità di advertising creando un beacon con il campo flag settato a 0x04, cioè non viene supportato il Bluetooth Basic Rate (BR) e Enhanced Data Rate (EDR) e il campo company identifier pari a 0x00059 che corrisponde a Nordic Semiconductor. Codifica il dato di advertising e lo passa allo stack mediante la funzione **ble_advdata_encode**. Inoltre crea una struttura da passare allo stack quando si inizia l'advertising con la funzione **sd_ble_gap_adv_set**.

Nella funzione **advertising_init()** si sono presentati i seguenti errori:

1. unknown type name ‘ble_advdata_t’
2. unknown type name ‘ble_advdata_manuf_data_t’
3. ‘APP_COMPANY_IDENTIFIER’ undeclared (first use in this function)
4. ‘m_beacon_info’ undeclared (first use in this function)
5. ‘APP_BEACON_INFO_LENGTH’ undeclared (first use in this function)
6. ‘BLE_ADVDATA_NO_NAME’ undeclared (first use in this function)
7. ‘m_adv_params’ undeclared (first use in this function)

8. ‘NON_CONNECTABLE_ADV_INTERVAL’ undeclared (first use in this function)
9. ‘m_adv_data’ undeclared (first use in this function)
10. ‘m_adv_handle’ undeclared (first use in this function)

Gli errori 1 e 2 producono ulteriori errori sulle variabili advdata e manuf_specific_data in quanto le struct ‘ble_advdata_t’ e ‘ble_advdata_manuf_data_t’ non sono state dichiarate.

Per risolvere gli errori 1, 2 e 6 è stata aggiunta la libreria "ble_advdata.h" nel main:

```
#include "ble_advdata.h"
```

in questo modo si eliminano anche gli errori relativi alle variabili advdata e manuf_specific_data.

Per le costanti 3, 5 e 8 è stato aggiunto nel main.c:

```
#define NON_CONNECTABLE_ADV_INTERVAL MSEC_TO_UNITS(100,
    UNIT_0_625_MS) /*< The advertising interval for non-
        connectable advertisement (100 ms). This value can vary
        between 100ms to 10.24s). */
```



```
#define APP_BEACON_INFO_LENGTH 0x17 /*< Total length of
    information advertised by the Beacon. */
#define APP_COMPANY_IDENTIFIER 0x0059 /*< Company
    identifier for Nordic Semiconductor ASA. as per www.
    bluetooth.org. */
```

Per risolvere l'errore 4 è stato dichiarato nel main il vettore che costituisce il payload del beacon. I commenti al codice sono abbastanza esplicativi della struttura del dato.

```
static uint8_t m_beacon_info[APP_BEACON_INFO_LENGTH] = /*
    *< Information advertised by the Beacon. */
{
    APP_DEVICE_TYPE,      // Manufacturer specific information
    . Specifies the device type in this
    // implementation.
    APP_ADV_DATA_LENGTH, // Manufacturer specific information
    . Specifies the length of the
    // manufacturer specific data in this implementation.
    APP_BEACON_UUID,     // 128 bit UUID value.
    APP_MAJOR_VALUE,     // Major arbitrary value that can be
    used to distinguish between Beacons.
```

```

APP_MINOR_VALUE,      // Minor arbitrary value that can be
                      // used to distinguish between Beacons.
APP_MEASURED_RSSI    // Manufacturer specific information
                      . The Beacon's measured TX power in
                      // this implementation.
};


```

le cui costanti sono

```

#define APP_ADV_DATA_LENGTH 0x15 /*< Length of
                                manufacturer specific data in the advertisement. */
#define APP_DEVICE_TYPE 0x02 /*< 0x02 refers to Beacon.
                                */
#define APP_MEASURED_RSSI 0xC3 /*< The Beacon's measured
                                RSSI at 1 meter distance in dBm. */
#define APP_MAJOR_VALUE 0x01, 0x02 /*< Major value used
                                to identify Beacons. */
#define APP_MINOR_VALUE 0x03, 0x04 /*< Minor value
                                used to identify Beacons. */
#define APP_BEACON_UUID 0x01, 0x12, 0x23, 0x34, \
  0x45, 0x56, 0x67, 0x78, \
  0x89, 0x9a, 0xab, 0xbc, \
  0xcd, 0xde, 0xef, 0xf0           /*< Proprietary UUID
                                for Beacon. */


```

Per risolvere 7 è stata dichiarata globalmente la variabile

```

static ble_gap_adv_params_t m_adv_params; /*< Parameters
                                          to be passed to the stack when starting advertising. */


```

L'errore 9 è stato risolto dichiarando il buffer che contiene l'advertising codificato e la struttura che contiene il puntato al dato di advertising codificato.

```

static uint8_t                  m_enc_advdata[
BLE_GAP_ADV_SET_DATA_SIZE_MAX]; /*< Buffer for storing
                                  an encoded advertising set. */

// Struct that contains pointers to the encoded advertising
// data.
static ble_gap_adv_data_t m_adv_data =
{
    .adv_data =
    {
        .p_data = m_enc_advdata,
        .len     = BLE_GAP_ADV_SET_DATA_SIZE_MAX
    },
}


```

```

    .scan_rsp_data =
{
    .p_data = NULL,
    .len     = 0
}
};
```

Per 10 è stato dichiarato

```

static uint8_t m_adv_handle =
    BLE_GAP_ADV_SET_HANDLE_NOT_SET; /*< Advertising handle
                                     used to identify an advertising set. */
```

Listing 1: Dichiarazione di m_adv_handle

Successivamente viene aggiunta la funzione **advertising_start()** usata per iniziare l'advertising.

```

static void advertising_start(void)
{
    ret_code_t err_code;

    err_code = sd_ble_gap_adv_start(m_adv_handle,
                                    APP_BLE_CONN_CFG_TAG);
    ERROR_CHECK(err_code);

    err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
    ERROR_CHECK(err_code);
}
```

Nella funzione **advertising_start()** si sono verificati i seguenti errori:

1. ‘m_adv_handle’ undeclared (first use in this function)
2. ‘APP_BLE_CONN_CFG_TAG’ undeclared (first use in this function)
3. ‘BSP_INDICATE_ADVERTISING’ undeclared (first use in this function)
4. undefined reference to ‘bsp_indication_set’

L’errore 1 è stato risolto come riportato nel codice 1.

Per 2 è stata scritta la costante

```
#define APP_BLE_CONN_CFG_TAG 1 /*< A tag identifying the
                                SoftDevice BLE configuration. */
```

Per 3 è stata inclusa la libreria "bsp.h"

```
#include "bsp.h"
```

Per il corretto funzionamento delle librerie occorre settare manualmente il percorso delle librerie stesse cliccando nell'IDE Segger **Project → Options → Preprocessor → User Include Directories** ed aggiungendo:

```
..../components/libraries/bsp
..../components/libraries/button
```

Per l'errore 4 occorre aggiungere il file sorgente "bsp.c" nel progetto. Per fare questo è stata creata la cartella "Board Support" nel progetto e vi è stato inserito il file sorgente. Il file "bsp.c" necessita anche di "boards.c" presente in .../nRF5_SDK_16.0.0_98a08e2/components/boards per il corretto funzionamento. Per far partire il beaconing sono state richiamate le funzioni **advertising_init()** e **advertising_start()** all'interno del **main()**.

```
int main(void)
{
    initialize();
    start();
    advertising_init();
    advertising_start();
    for (;;)
    {
        (void)sd_app_evt_wait();
    }
}
```

Con queste modifiche si ottiene l'errore Mesh error 4 at 0x00026901 causato dalla funzione **mesh_provisionee_prov_start** contenuta in **start()**. L'errore è stato risolto disabilitando il proxy: nel codice è stata portata a zero la macro MESH_FEATURE_GATT_ENABLED che non fa compilare la seguente parte di codice

```
#if MESH_FEATURE_GATT_ENABLED
    gap_params_init();
    conn_params_init();
#endif
```

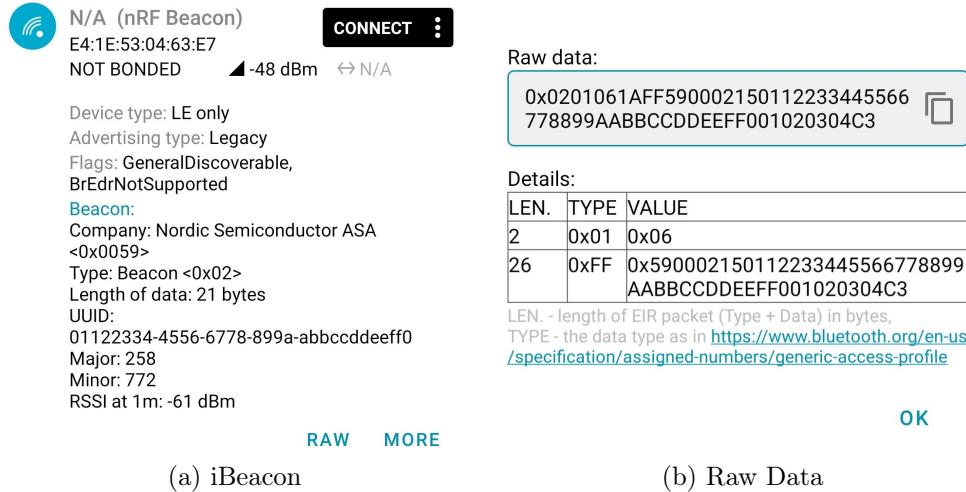
MESH_FEATURE_GATT_ENABLED è definita nella libreria "**nrf_mesh_gatt.h**" come

```
#define MESH FEATURE GATT ENABLED (
    MESH FEATURE GATT PROXY ENABLED ||
    MESH FEATURE PB GATT ENABLED)
```

quindi sono state portate a zero le due macro MESH_FEATURE_GATT_PROXY_ENABLED e MESH_FEATURE_PB_GATT_ENABLED nella libreria "nrf_mesh_config_app.h".

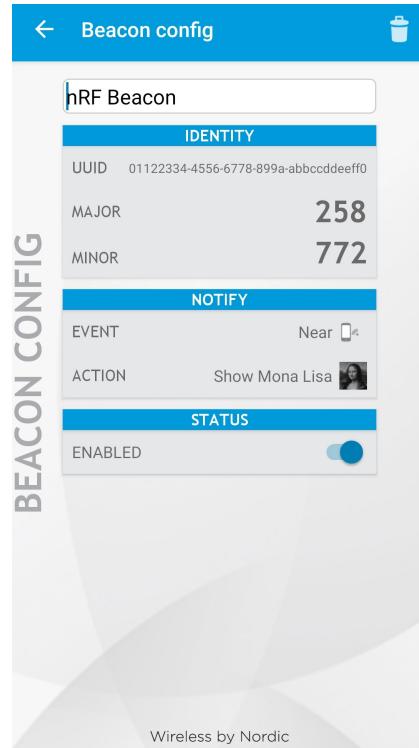
```
/** PB-GATT feature. To be enabled only in combination with
   linking GATT files. */
#define MESH FEATURE PB GATT ENABLED (0)
/** GATT proxy feature. To be enabled only in combination
   with linking GATT proxy files. */
#define MESH FEATURE GATT PROXY ENABLED (0)
```

La disabilitazione del proxy non permette agli smartphone di partecipare alla rete mesh, ma possono comunque ricevere beacon come mostrato nelle figure 29a, 29b, e 29c.



(a) iBeacon

(b) Raw Data



(c) iBeacon nell'app nRF Beacon

Figura 29

6.2 Integrazione dell'esempio Mesh Beacons nel Light Switch Server

Per risolvere il problema del proxy gatt incontrato nel capitolo precedente, ho usato le funzioni presenti nell'esempio Beacons nell'SDK Mesh 4.0 descritto nel capitolo 5.

6.2.1 Implementazione di iBeacon

Nell'esempio Beacons, nella funzione `adv_start()` viene definita una variabile chiamata `adv_data` che contiene i bit del payload del pacchetto di advertising da mandare.

Se nel campo Manufacturer ID del beacon si inserisce 0x004C si hanno beacon per dispositivi Apple. I valori di "Company Identifier" si possono trovare al seguente link Company Identifiers.

La variabile `adv_data` è stata modificata come di seguito per mandare iBeacons.

```
static const uint8_t adv_data[] =  
{  
    APP_FLAG_BEACON_LENGTH, /* Flag Beacon length (including  
                             type, but not itself) */  
    BLE_GAP_AD_TYPE_FLAGS, /* Flags for discoverability. */  
    BLE_GAP_ADV_TYPE_EXTENDED_CONNECTABLE_NONSCANNABLE_UNDIRECTED  
    , /*< Connectable non-scannable undirected  
        advertising events using extended advertising PDUs. */  
    APP_ADVERTISER_LENGTH, /* Advertiser data length (including  
                           type, but not itself) */  
    BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA, /*  
                                                Manufacturer Specific Data. */  
    APP_COMPANY_IDENTIFIER, /* Company ID */  
    APP_iBEACON_TYPE, /* iBeacon Type */  
    APP_iBEACON_LENGTH, /* iBeacon Length */  
    APP_BEACON_UUID, /* UUID */  
    APP_MAJOR_VALUE, /* Major Value */  
    APP_MINOR_VALUE, /* Minor Value */  
    APP_MEASURED_RSSI /* RSSI at 1 m */  
};
```

Listing 2: Configurazione della variabile `adv_data` per iBeacons

avendo dichiarato nel main i seguenti valori costanti

```
#define APP_FLAG_BEACON_LENGTH 0x02 /*< Length of the  
discoverability Beacon. */
```

```

#define APP_ADVERTISER_LENGTH 0x1A /* Total length of the
    iBeacon. */
#define APP_iBEACON_LENGTH 0x15 /* Length of manufacturer
    specific data in the advertisement. */
#define APP_iBEACON_TYPE 0x02 /*< iBeacon Type. */
#define APP_MEASURED_RSSI 0xC3 /* The Beacon's measured
    RSSI at 1 meter distance in dBm. */
#define APP_COMPANY_IDENTIFIER 0x4C, 0x00 /* Company
    identifier for Apple Inc. as per www.bluetooth.org. */
#define APP_MAJOR_VALUE 0x01, 0x02 /* Major value used to
    identify Beacons. */
#define APP_MINOR_VALUE 0x03, 0x04 /* Minor value used to
    identify Beacons. */
#define APP_BEACON_UUID 0x01, 0x12, 0x23, 0x34, \
                    0x45, 0x56, 0x67, 0x78, \
                    0x89, 0x9A, 0xAB, 0xBC, \
                    0xCD, 0xDE, 0xEF, 0xF0 /*<
    Proprietary UUID for Beacon. */

```

Listing 3: Costanti pacchetto iBeacon

ed avendo usato i valori nella libreria "BLE_GAP.h":

- "BLE_GAP_AD_TYPE_FLAGS" = 0x01
- "BLE_GAP_ADV_TYPE_EXTENDED_CONNECTABLE_NONSCANNABLE_UNDIRECTED" = 0x06
- "BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA" = 0xFF.

Per trasmettere i beacon, nel firmware Light Switch Server sono state aggiunte delle funzioni dell'esempio Beaconing presente nel SDK Mesh. Nella funzione **mesh_init** è stata aggiunta **nrf_mesh_rx_cb_set** definita come di seguito. La funzione di callback è stata già descritta nel capitolo 5.2.

```

void nrf_mesh_rx_cb_set(nrf_mesh_rx_cb_t rx_cb)
{
    m_rx_cb = rx_cb;
}

```

Aggiungendo questa funzione si ottiene l'errore 'rx_cb' undeclared (first use in this function) risolto aggiungendo nel main il seguente codice

```

static void rx_cb(const nrf_mesh_adv_packet_rx_data_t *
    p_rx_data)

```

```

{
    LEDS_OFF(BSP_LED_0_MASK); /* @c LED_RGB_RED_MASK on
                                pca10031 */
    char msg[128];
    (void) sprintf(msg, "RX [%@u]: RSSI: %3d ADV TYPE: %x
                        ADDR: [%02x:%02x:%02x:%02x:%02x:%02x] ",
                    p_rx_data->p_metadata->params.scanner.timestamp,
                    p_rx_data->p_metadata->params.scanner.rssi,
                    p_rx_data->adv_type,
                    p_rx_data->p_metadata->params.scanner.adv_addr.addr[0],
                    p_rx_data->p_metadata->params.scanner.adv_addr.addr[1],
                    p_rx_data->p_metadata->params.scanner.adv_addr.addr[2],
                    p_rx_data->p_metadata->params.scanner.adv_addr.addr[3],
                    p_rx_data->p_metadata->params.scanner.adv_addr.addr[4],
                    p_rx_data->p_metadata->params.scanner.adv_addr.addr[5]);
    _LOG_XB(LOG_SRC_APP, LOG_LEVEL_INFO, msg, p_rx_data->
            p_payload, p_rx_data->length);
    LEDS_ON(BSP_LED_0_MASK); /* @c LED_RGB_RED_MASK on
                                pca10031 */
}

```

Si ottengono i seguenti errori

1. ‘m_advertiser’ undeclared (first use in this function)
2. ‘m_adv_buffer’ undeclared (first use in this function)
3. ‘ADVERTISER_BUFFER_SIZE’ undeclared (first use in this function)

L’errore 1 è stato risolto aggiungendo nel main

```
static advertiser_t m_advertiser;
```

e la libreria "advertiser.h" che definisce il tipo di variabile **advertiser_t**.

L’errore 2 è stato risolto aggiungendo nel main

```
static uint8_t m_adv_buffer [ADVERTISER_BUFFER_SIZE];
```

L’errore 3 è stato risolto aggiungendo nel main

```
#define ADVERTISER_BUFFER_SIZE (64)
```

Sempre nella funzione **mesh_init** è stata aggiunta **adv_init()** definita come di seguito.

```

static void adv_init(void)
{
    advertiser_instance_init(&m_advertiser, NULL,
        m_adv_buffer, ADVERTISER_BUFFER_SIZE);
}

```

La funzione **advertiser_instance_init** può essere richiamata più volte per inizializzare differenti advertiser.

Nella funzione **start** è stata aggiunta **adv_start** definita come di seguito.

```

static void adv_start(void)
{
    bearer_adtype_add(BLE_GAP_AD_TYPE_COMPLETE_LOCAL_NAME);

    advertiser_enable(&m_advertiser);

    static const uint8_t adv_data[] =
    {
        APP_FLAG_BEACON_LENGTH, /* Flag Beacon length (
            including type, but not itself) */
        BLE_GAP_AD_TYPE_FLAGS, /*< Flags for discoverability .
        */
        BLE_GAP_ADV_TYPE_EXTENDED_CONNECTABLE_NONSCANNABLE_UNDIRECTED
        , /*< Connectable non-scannable undirected
        advertising
        events using extended advertising PDUs. */
        APP_ADVERTISER_LENGTH, /* Advertiser data length (
            including type, but not itself) */
        BLE_GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA, /*<
        Manufacturer Specific Data. */
        APP_COMPANY_IDENTIFIER, /* Company ID */
        APP_iBEACON_TYPE, /* iBeacon Type */
        APP_iBEACON_LENGTH, /* iBeacon Length */
        APP_BEACON_UUID, /* UUID */
        APP_MAJOR_VALUE, /* Major Value*/
        APP_MINOR_VALUE, /* Minor Value */
        APP_MEASURED_RSSI /* RSSI at 1 m */
    };

    /* Allocate packet */
    adv_packet_t * p_packet = advertiser_packet_alloc(&
        m_advertiser, sizeof(adv_data));
    if (p_packet)

```

```

{
    /* Construct packet contents */
    memcpy(p_packet->packet.payload, adv_data, sizeof(
        adv_data));
    /* Repeat forever */
    p_packet->config.repeats = ADVERTISER_REPEAT_INFINITE;

    advertiser_packet_send(&m_advertiser, p_packet);
}
}

```

avendo definito nel main le macro come nel codice 3. La funzione **bearer_adtype_add** aggiunge il tipo BLE_GAP_AD_TYPE_COMPLETE_LOCAL_NAME alla lista di tipi di AD accettati, mentre **advertiser_enable** abilita l'istanza m_advertiser. La funzione **advertiser_packet_alloc** alloca un buffer dalla istanza m_advertiser e **advertiser_packet_send** trasmette il pacchetto p_packet usando l'istanza m_advertiser. Il risultato è quello che si può vedere in figura 30a e 30b.

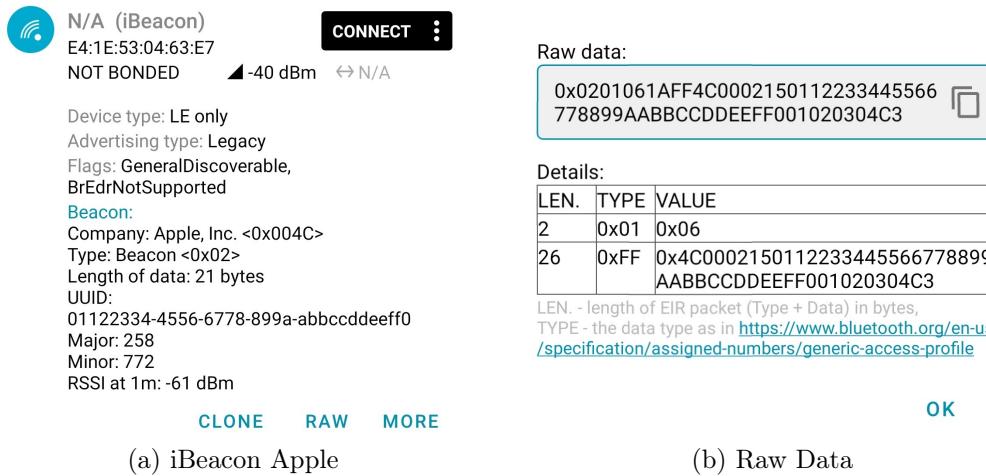


Figura 30: Beacon ricevuto dall'app nRF Connect

Se la macro APP_COMPANY_IDENTIFIER viene cambiata in 0x0059 nel campo Manufacturer ID, allora il produttore è Nordic Semiconductor.

```
#define APP_COMPANY_IDENTIFIER 0x59, 0x00 /* Company
identifier for Nordic Semiconductor ASA. as per www.
bluetooth.org. */
```

Listing 4: Manufacturer ID Nordic Semiconductor

I beacon trasmessi con Manufacturer ID pari a 0x0059 sono mostrati in figura 31a e 31b.

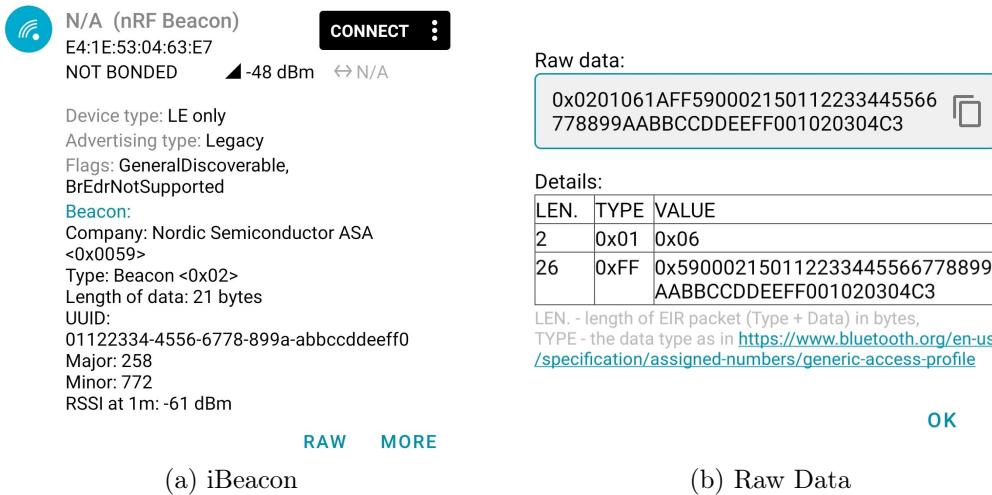


Figura 31: Beacon ricevuto dall'app nRF Connect

6.2.2 Implementazione dei beacon Eddystone

Per costruire il beacon Eddystonè stata modificata la variabile adv_data precedentemente descritta nel seguente modo

```

static const uint8_t adv_data[] =
{
    APP_FLAG_BEACON_LENGTH, /* Flag Beacon length (including
                             * type, but not itself) */
    BLE_GAP_AD_TYPE_FLAGS, /*< Flags for discoverability. */
    BLE_GAP_ADV_TYPE_EXTENDED_CONNECTABLE_NONSCANNABLE_UNDIRECTED
    , /*< Connectable non-scannable undirected
       advertising events using extended advertising PDUs. */
    APP_SERVICE_UUID_LENGTH, /*< Server UUID advertising
                             * data length */
    BLE_GAP_AD_TYPE_16BIT_SERVICE_UUID_COMPLETE, /*<
                                                 * Complete list of 16 bit service UUIDs. */
    APP_EDDYSTONE_UUID, /*< Eddystone UUID. */
    APP_EDDYSTONE_DATA_LENGTH, /*< Eddystone data length. */
    BLE_GAP_AD_TYPE_SERVICE_DATA, /*< Service Data - 16-bit
                                 * UUID. */
    APP_EUUID, /*< Eddystone UUID. */
    APP_EDDYSTONE_URL_FRAME_TYPE, /*< Frame Type: Eddystone
                                 * URL. */
}

```

```

APP_MEASURED_RSSI, /*< Ranging data. */
APP_URL_PREFIX, /*< URL prefix http://www. */
'n',
'o',
'r',
'd',
'i',
'c',
's',
'e',
'm',
'i',
'.',
'c',
'o',
'm',
'/
};


```

Listing 5: Configurazione della variabile adv_data per beacon Eddystone
avendo definito nel main le seguenti macro

```

#define APP_SERVICE_UUID_LENGTH 0x03 /*< Complete list of
    UUID service advertising data length. */
#define APP_EDDYSTONE_DATA_LENGTH 0x15 /*< Eddystone
    Frame Length. */
#define APP_EDDYSTONE_UUID 0xAA, 0xFE /*< Eddystone UUID
    . */
#define APP_EDDYSTONE_URL_FRAME_TYPE 0x10 /*< Eddystone
    URL Frame Type. */
#define APP_URL_PREFIX 0x00 /*< Eddystone URL Prefix. */

```

ed avendo usato i valori nella libreria "BLE_GAP.h":

- "BLE_GAP_AD_TYPE_FLAGS" = 0x01
- "BLE_GAP_ADV_TYPE_EXTENDED_CONNECTABLE_
NONSCANNABLE_UNDIRECTED" = 0x06
- "BLE_GAP_AD_TYPE_16BIT_SERVICE_UUID_COMPLETE" =
0x03
- "BLE_GAP_AD_TYPE_SERVICE_DATA" = 0x16.

Per implementare la trasmissione di beacon Eddystone nell'esempio Light Switch Server il procedimento è uguale a quello descritto sezione Implementazione di iBeacon con la variabile `adv_data` descritta sopra.

Il risultato è quello che si può vedere nelle due figure 32a e 32b.

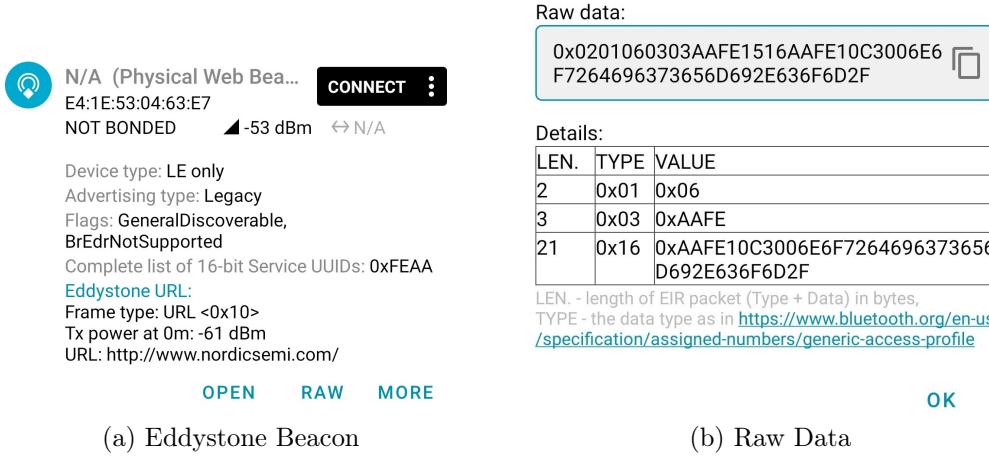


Figura 32: Eddystone Beacon ricevuto dall'app nRF Connect

6.3 Integrazione di iBeacon e beacon Eddystone in Light Switch Server

Per trasmettere sia gli iBeacon sia i beacon Eddystone sono state inserite due variabili chiamate `iBeacon` e `eddystone`, che hanno la stessa funzione di `adv_data`. Le due variabili sono state dichiarate rispettivamente come nei codici 2 e 5.

Con una sola istanza non è possibile mandare sia iBeacon sia Eddystone, quindi sono state dichiarate due istanze necessarie per trasmettere i due messaggi di advertising

```
static advertiser_t m_advertiser;
static advertiser_t m_advertiser2;
```

e due vettori che contengono i dati

```
static uint8_t m_adv_buffer [ADVERTISER_BUFFER_SIZE];
/* */
static uint8_t m_adv_buffer2 [ADVERTISER_BUFFER_SIZE];
```

Nella funzione `adv_init()` è stato aggiunto

```

advertiser_instance_init(&m_advertiser2, NULL,
    m_adv_buffer2, ADVERTISER_BUFFER_SIZE);

```

mentre nella funzione **adv_start()** è stato aggiunto

```

advertiser_enable(&m_advertiser2);

adv_packet_t * p_packet_eddystone = advertiser_packet_alloc
    (&m_advertiser2, sizeof(eddystone));
if (p_packet_eddystone)
{
    memcpy(p_packet_eddystone->packet.payload, eddystone,
        sizeof(eddystone));
    p_packet_eddystone->config.repeats =
        ADVERTISER_REPEAT_INFINITE;
    advertiser_packet_send(&m_advertiser2, p_packet_eddystone
    );
}

```

Il risultato finale che si può vedere nell'app nRF Connect è quello che si può osservare nell'animazione 33.¹

Figura 33: Animazione beacon

¹È possibile visualizzare correttamente l'animazione usando Adobe Acrobat Reader o un qualsiasi lettore PDF che abbia i plugin appropriati abilitati.

7 Integrare la Mesh negli esempi dell'SDK nRF5

L'SDK nRF5 per Mesh è compatibile con l'SDK nRF della Nordic. Questo permette di includere le funzionalità dell'SDK nrRF5 in un progetto mesh esistente o includere le funzionalità dell'SDK nRF5 per Mesh in un esempio dell'SDK nRF5.

7.1 Includere le funzionalità dell'SDK Mesh nRF5 in un esempio dell'SDK nRF5

1. Includere i seguenti file sorgente dall'SDK nRF per Mesh in un esempio dell'SDK:

- Tutti i file C in `mesh/core/src`
- Tutti i file C in `mesh/bearer/src`
- Tutti i file C in `mesh/prov/src` tranne `nrf_mesh_prov_bearer_gatt.c`
- Tutti i file C in `mesh/access/src`
- Tutti i file C in `mesh/dfu/src`
- Tutti i file C in `mesh/stack/src`
- `models/foundation/config/src/config_server.c`
- `models/foundation/config/src/composition_data.c`
- `models/foundation/config/src/packed_index_list.c`
- `models/foundation/health/src/health_server.c`
- Ogni altro modello mesh che viene usato nell'applicazione (ad esempio Generic OnOff model)
- `external/micro-ecc/uECC.c`
- `examples/common/src/assertion_handler_weak.c`
- `examples/common/src/mesh_provisionee.c`
- `examples/common/src/app_onoff.c`
- `examples/common/src/rtt_input.c`
- `examples/common/src/simple_hal.c`
- `examples/common/src/mesh_app_utils.c`
- `examples/common/src/mesh_adv.c`
- `examples/common/src/ble_softdevice_support.c`
- `examples/nrf_mesh_weak.c`

- config/sdk_config.h
- mesh/gatt/src/mesh_gatt.c
- mesh/gatt/src/proxy.c
- mesh/gatt/src/proxy_filter.c
- mesh/friend/src/friend.c
- mesh/friend/src/friend_queue.c
- mesh/friend/src/friend_sublist.c
- mesh/friend/src/core_tx_friend.c

| Nota

Se non sono necessarie varie funzioni mesh (ad esempio, DFU), i file corrispondenti possono essere semplicemente omessi dal file di progetto. Tuttavia è necessario aggiungere examples/nrf_mesh_weak.c al loro posto per fornire gli stub per le funzioni API mancanti.

2. Aggiungi le seguenti cartelle al progetto includendo il percorso dell'esempio dell'SDK nRF5:

- mesh/core/api
- mesh/core/include
- mesh/bearer/api
- mesh/bearer/include
- mesh/prov/api ok
- mesh/prov/include
- mesh/access/api
- mesh/access/include
- mesh/dfu/api
- mesh/dfu/include
- mesh/stack/api
- models/foundation/config/include
- models/foundation/health/include
- I percorsi a cartelle o modelli mesh che sono usati nell'applicazione, in questo caso models/model_spec/generic_onoff/include e models/model_spec/common/include
- external/micro-ecc

- examples/common/include
 - I percorsi a risorse usate nell'esempio mesh che sono usate nell'applicazione, come mesh/friend/api, mesh/friend/include, mesh/-gatt/api e mesh/gatt/include, examples/light_switch/include, examples/light_switch/server/include
3. Aggiungere i seguenti simboli del preprocessore al progetto dell'esempio dell'SDK nRF5:
 - NRF52_SERIES
 - NRF_MESH_LOG_ENABLE=NRF_LOGUSES_RTT
 - CONFIG_APP_IN_CORE
 - NO_VTOR_CONFIG
 - USE_APP_CONFIG
 - NRF52832
 - NRF52832_XXAA
 - S132
 - SOFTDEVICE_PRESENT
 - NRF_SD_BLE_API_VERSION=7
 - BOARD_PCA10040
 - CONFIG_GPIO_AS_PINRESET

7.2 Integrazione delle funzionalità Mesh nell'esempio Eddystone

Dopo aver eseguito i passaggi riportati nel capitolo precedente all'esempio Eddystone, ho iniziato ad aggiungere nel main le funzioni che gestiscono la Mesh. In particolare **start()**, presa dall'esempio Beaconsing e definita come di seguito, permette di configurare lo stack mesh.

```
static void start(void)
{
    rtt_input_enable(app_rtt_input_handler,
                     RTT_INPUT_POLL_PERIOD_MS);

    if (!m_device_provisioned)
    {
        static const uint8_t static_auth_data[NRF_MESH_KEY_SIZE]
            = STATIC_AUTH_DATA;
```

```

mesh_provisionee_start_params_t prov_start_params =
{
    .p_static_data      = static_auth_data,
    .prov_complete_cb = provisioning_complete_cb,
    .prov_device_identification_start_cb =
        device_identification_start_cb,
    .prov_device_identification_stop_cb = NULL,
    .prov_abort_cb = provisioning_aborted_cb,
    .p_device_uri = EX_URI_LS_SERVER
};

ERROR_CHECK(mesh_provisionee_prov_start(&
    prov_start_params));
}

mesh_app_uuid_print(nrf_mesh_configure_device_uuid_get());
;

ERROR_CHECK(mesh_stack_start());
;

hal_led_mask_set(LEDs_MASK, LED_MASK_STATE_OFF);
hal_led_blink_ms(LEDs_MASK, LED_BLINK_INTERVAL_MS,
    LED_BLINK_CNT_START);
}

```

Dopo aver aggiunto questa funzione, si verificano i seguenti errori:

1. ‘STATIC_AUTH_DATA’ undeclared (first use in this function)
2. unknown type name ‘mesh_provisionee_start_params_t’

L’errore 1 è stato risolto aggiungendo la libreria `light_switch_example_common.h`

```
#include "light_switch_example_common.h"
```

mentre per 2 ho aggiunto la libreria `mesh_provisionee.h`.

```
#include "mesh_provisionee.h"
```

Per il corretto funzionamento della funzione `start()` ho dovuto aggiungere il file `nrf_mesh_prov_bearer_gatt.c` contrariamente a quanto indicato in Includere le funzionalità dell’SDK Mesh nRF5 in un esempio dell’SDK nRF5. Questo porta ad un malfunzionamento del programma durante l’esecuzione. I problemi riscontrati sono risolti nel capitolo 8.

8 Coesistenza tra l'applicazione Eddystone e l'esempio Light Switch

Nel capitolo 7.1 si suggerisce di non aggiungere il file nrf_mesh_prov_bearer_gatt.c al progetto perché aggiungere il provisioning PB-GATT ad una applicazione BLE non è consigliato. Questo thread spiega meglio il problema. Come si può vedere, non è possibile usare i servizi BLE e il Bearer GATT contemporaneamente. Occorre che il dispositivo faccia parte della rete mesh, poi resettarlo e alla fine aggiungere il servizi BLE.

Se si vogliono usare sia le funzionalità Mesh sia BLE contemporaneamente, si suggerisce di usare l'esempio coexistence dell'SDK per avere un'idea migliore di come far coesistere le funzionalità. L'esempio in cui si mostra la coesistenza tra Mesh e BLE presente nell'SDK nRF5 può essere un ottimo punto di partenza per gli obiettivi di questo progetto. Si può usare l'app nRF Connect per vedere i beacon trasmessi e i servizi BLE, mentre si può usare l'app nRF Mesh solo se è presente il PB-GATT o se il proxy è abilitato.

8.1 Operazioni preliminari per configurare l'ambiente di sviluppo

Per iniziare il nuovo progetto è stato copiato l'esempio Coexistence in una nuova cartella e sono state fatte diverse modifiche di seguito presentate.

8.1.1 SDK configuration header file

Il file header di configurazione dell'SDK (sdk_config.h) aiuta a gestire la configurazione statica di un'applicazione che si basa sull'SDK nRF5. Le opzioni di configurazione incluse in questo file possono essere modificate rapidamente in una procedura guidata usando l'interfaccia grafica dello strumento Java open source CMSIS Configuration Wizard. Ogni modulo dell'SDK contiene almeno un'opzione di configurazione che lo abilita, se il modulo è disabilitato, anche se il codice sorgente viene aggiunto al progetto, questo non viene compilato perché il modulo non è implementato. Osservando il file sdk_config.h, è possibile verificare quali moduli sono utilizzati nell'applicazione:

```
example_module.c
#include "sdk_config.h"
#if EXAMPLE_MODULE_ENABLED
...
#endif //EXAMPLE_MODULE_ENABLED
```

Per ogni dispositivo supportato, esiste un file sdk_config.h generico che contiene tutte le opzioni di configurazione disponibili.

- sdk/nrf5/config/nrf52810/config/sdk_config.h per nRF52810
- sdk/nrf5/config/nrf52832/config/sdk_config.h per nRF52832
- sdk/nrf5/config/nrf52840/config/sdk_config.h per nRF52840

Per ogni dispositivo supportato, esistono file di linker generici per Segger Embedded Studio e GCC ARM. Questi file linker contengono tutte le sezioni di memoria utilizzate dai moduli SDK.

- Per GCC ARM: config/<device_name>/armgcc/generic_gcc_nrf52.ld
- Per Segger Embedded Studio: config/<device_name>/ses/flash_placement.xml

Lo strumento open source Java, CMSIS Configuration Wizard, può essere integrato con Segger Embedded Studio attraverso la seguente procedura

1. **File → Open Studio Folder... → External Tools Configuration.**
2. Verrà aperto il file tools.xml nell'editor.
3. Inserisci il seguente testo prima del tag </tools>:

```
<item name="Tool.CMSIS_Config_Wizard" wait="no">
<menu>&#x26amp; CMSIS Configuration Wizard</menu>
<text>CMSIS Configuration Wizard</text>
<tip>Open a configuration file in CMSIS Configuration
    Wizard</tip>
<key>Ctrl+Y</key>
<match>*config *.h</match>
<message>CMSIS Config</message>
<commands>
    java -jar "$CMSIS_CONFIG_TOOL" "$(
        InputPath)";
</commands>
</item>
```

Ho integrato i file header sdk_config.h dell'esempio eddystone e coexistence. Le differenze sono elencate in tabella. Partendo dal file sdk_config.h dell'esempio coexistence, i moduli che sono abilitati nell'esempio Eddystone ma sono disabilitati o non definiti nel file originale, sono stati abilitati.

Configuration	Coexist	Eddystone
Queue size for BLE GATT Queue module	4	undefined
Database discovery module	enabled	undefined
BLE GATT Queue Module	enabled	undefined
Queued writes support module (prepare/execute write)	enabled	disabled
Peer Manager	enabled	disabled
central-specific Peer Manager functionality	disabled	enabled
Battery Service	enabled	disabled
Eddystone Configuration Service	undefined	enabled
Immediate Alert Service Client	enabled	disabled
Immediate Alert Service	enabled	disabled
Link Loss Service	enabled	disabled
TX Power Service	enabled	disabled
AES CBC mode using CC310	enabled	disabled
AES CTR mode using CC310	enabled	disabled
AES CBC_MAC mode using CC311	enabled	disabled
AES CMAC mode using CC310	enabled	disabled
AES CCM mode using CC310	enabled	disabled
AES CCM* mode using CC310	enabled	disabled
CHACHA-POLY mode using CC310	enabled	disabled
secp160r1 elliptic curve support using CC310	enabled	disabled
secp160r2 elliptic curve support using CC310	enabled	disabled
secp192r1 elliptic curve support using CC310	enabled	disabled
secp224r1 elliptic curve support using CC310	enabled	disabled
secp256r1 elliptic curve support using CC310	enabled	disabled
secp384r1 elliptic curve support using CC310	enabled	disabled
secp521r1 elliptic curve support using CC310	enabled	disabled
secp160k1 elliptic curve support using CC310	enabled	disabled
secp192k1 elliptic curve support using CC310	enabled	disabled
secp224k1 elliptic curve support using CC310	enabled	disabled
secp256k1 elliptic curve support using CC310	enabled	disabled
Ed25519 curve support using CC310	enabled	disabled
CC310 backend implementation for hardware-accelerated SHA-256	enabled	disabled
CC310 backend implementation for SHA-512	enabled	disabled
CC310 backend implementation for HMAC using SHA-512	enabled	disabled
Cifra backend	disabled	enabled

mbed TLS backend	disabled	enabled
AES CBC mode mbed TLS	enabled	disabled
AES CTR mode using mbed TLS	enabled	disabled
AES CFB mode using mbed TLS	enabled	disabled
AES CBC MAC mode using mbed TLS	enabled	disabled
AES CMAC mode using mbed TLS	enabled	disabled
AES CCM mode using mbed TLS	enabled	disabled
AES GCM mode using mbed TLS	enabled	disabled
secp192r1 (NIST 192-bit) support using MBEDTLS	enabled	disabled
secp224r1 (NIST 224-bit) support using MBEDTLS	enabled	disabled
secp256r1 (NIST 256-bit) support using MBEDTLS	enabled	disabled
secp384r1 (NIST 384-bit) support using MBEDTLS	enabled	disabled
secp521r1 (NIST 521-bit) support using MBEDTLS	enabled	disabled
secp192k1 (Koblitz 192-bit) support using MBEDTLS	enabled	disabled
secp224k1 (Koblitz 224-bit) support using MBEDTLS	enabled	disabled
secp256k1 (Koblitz 256-bit) support using MBEDTLS	enabled	disabled
bp256r1 (Brainpool 256-bit) support using MBEDTLS	enabled	disabled
bp384r1 (Brainpool 384-bit) support using MBEDTLS	enabled	disabled
bp512r1 (Brainpool 512-bit) support using MBEDTLS	enabled	disabled
Curve25519 support using MBEDTLS	enabled	disabled
mbed TLS backend implementation for SHA-256	enabled	disabled
mbed TLS backend implementation for SHA-512	enabled	disabled
mbed TLS backend implementation for HMAC using SHA-256	enabled	disabled
mbed TLS backend implementation for HMAC using SHA-512	enabled	disabled
nRF HW RNG backend	disabled	enabled
Oberon backend	disabled	enabled

CHACHA-POLY mode using Oberon	enabled	disabled
secp256r1 curve support using Oberon library	enabled	disabled
Ed25519 signature scheme	enabled	disabled
Oberon backend implementation for SHA-256	enabled	disabled
Oberon backend implementation for SHA-512	enabled	disabled
Oberon backend implementation for HMAC using SHA-512	enabled	disabled
Use static memory buffers for context and temporary init buffer	undefined	enabled
Initialize the RNG module automatically when nrf_crypto is initialized	undefined	enabled
RNG peripheral driver	disabled	enabled
RNG peripheral driver - legacy layer	disabled	enabled
SAADC_CONFIG_LP_MODE Enabling low power mode	disabled	enabled
Enable scheduling app_timer events to app_scheduler	disabled	enabled
Enable RTC always on	disabled	enabled
Number of virtual flash pages to use	3	10
Size of the internal queue	4	10
Dynamic memory allocator	disabled	enabled
Log RTT backend	enabled	disabled

Poiché più di un modulo backend è stato abilitato, ho disabilitato i seguenti:

```
#define NRF_CRYPTO_BACKEND_OBERON_ECC_SECP256R1_ENABLED 0

#define NRF_CRYPTO_BACKENDMBEDTLS_ECC_CURVE25519_ENABLED 0

#define NRF_CRYPTO_BACKENDMBEDTLS_HASH_SHA256_ENABLED 0

#define NRF_CRYPTO_BACKENDMBEDTLS_HASH_SHA512_ENABLED 0

#define NRF_CRYPTO_BACKEND_OBERON_HASH_SHA512_ENABLED 0

#define NRF_CRYPTO_BACKENDMBEDTLS_HMAC_SHA256_ENABLED 0

#define NRF_CRYPTO_BACKENDMBEDTLS_HMAC_SHA512_ENABLED 0
```

Inoltre ho riportato

```
#define FDS_VIRTUAL_PAGES 3
```

```
#define FDS_OP_QUEUE_SIZE 4
```

Debuggando il codice, si ha il seguente errore <error> app: ERROR 9 [NRF_ERROR_INVALID_LENGTH] at 0x00066647. È stato risolto scrivendo in sdk_config.h

```
#ifndef APP_TIMER_CONFIG_USE_SCHEDULER
#define APP_TIMER_CONFIG_USE_SCHEDULER 0
#endif
```

A questo punto il file header sdk_config.h è stato configurato correttamente.

8.1.2 Directory del preprocessore

In **Options → Preprocessor → User Include Directories** sono state aggiunte le seguenti directory:

- /components/ble/ble_services/eddystone
- /components/ble/ble_services/ble_escs
- /components/libraries/crypto/backend/cc310
- /components/libraries/crypto/backend/cc310_bl
- /components/libraries/crypto/backend/cifra
- /components/libraries/crypto/backend/mbedtls
- /components/libraries/crypto/backend/micro_ecc
- /components/libraries/crypto/backend/nrf_hw
- /components/libraries/crypto/backend/nrf_sw
- /components/libraries/crypto/backend/oberon
- /components/libraries/crypto/backend/optiga
- /components/libraries/stack_info
- /external/mbedtls/include
- /external/micro-ecc/micro-ecc

- /external/nrf_cc310/include
- /external/nrf_oberon
- /external/nrf_oberon/include
- /external/nrf_tls/mbedtls/nrf_crypto/config

8.1.3 Macro del preprocessore

In **Options → Preprocessor → Preprocessor Definitions** sono state aggiunte le macro MBEDTLS_CONFIG_FILE="nrf_crypto_mbedtls_config.h" e DEBUG, quest'ultima necessaria per abilitare il debug.

8.1.4 File sorgente

Ho aggiunto le seguenti cartelle (e il rispettivo contenuto) presenti nel progetto Eddystone al nuovo progetto

- nRF_Crypto
- nRF_Crypto backend cifra
- nRF_Crypto backend mbed TLS
- nRF_Crypto backend nRF HW
- nRF_Crypto backend Oberon
- nRF_Crypto backend uECC
- nRF_micro-ecc
- nRF_Oberon_Crypto
- nRF_TLS

Ho aggiunto il file sorgente nrf_ble_escs.c nella cartella nRF_BLE_Services e i seguenti file nella cartella nRF_Drivers:

- nrf_drv_rng.c
- nrf_nvmc.c
- nrfx_rng.c

Nella cartella nRF_Libraries ho aggiunto i seguenti file contenuti in /components/ble/ble_services/eddyStone/

- es_adv.c
- es_adv_frame.c
- es_adv_timing.c
- es_adv_timing_resolver.c
- es_battery_voltage_saadc.c
- es_flash.c
- es_gatts.c
- es_gatts_read.c
- es_gatts_write
- es_security.c
- es_slot.c
- es_slot_reg.c
- es_stopwatch.c
- es_tlm.c
- nrf_ble_es.c

i seguenti file contenuti in /external/cifra_AES128-EAX/

- cifra_eax_aes.c
- modes.c
- eax.c
- blockwise.c
- cifra_cmac.c
- cifra_AES128-EAX/gf128.c

e il file mem_manager.c contenuto in /components/libraries/mem_manager/.

A questo punto le operazioni preliminari per fare coesistere il servizio Eddystone con la Mesh sono completate.

8.2 Implementazione delle funzionalità Eddystone

In questa sezione sarà illustrato come implementare le funzioni che gestiscono il servizio Eddystone nel progetto. Nel file main.c è stato inserita la libreria nrf_ble_es.h

```
#include "nrf_ble_es.h"
```

e le macro

```
#define NON_CONNECTABLE_ADV_LED_PIN      BSP_BOARD_LED_0
    //!< Toggles when non-connectable advertisement is
    sent.

#define CONNECTED_LED_PIN                BSP_BOARD_LED_1
    //!< Is on when device has connected.

#define CONNECTABLE_ADV_LED_PIN          BSP_BOARD_LED_2
    //!< Is on when device is advertising connectable
    advertisements.
```

Nella funzione **bsp_event_handler** è stato aggiunta **nrf_ble_es_on_start_connectable_advertising** necessaria per mandare "connectable advertising" quando si preme il pulsante 2 (BSP_EVENT_KEY_2).

```
switch (event)
{
    case BSP_EVENT_KEY_2:
    {
        nrf_ble_es_on_start_connectable_advertising();
    } break;
    default:
    break;
}
```

Inoltre è stata implementata la funzione **on_es_evt** che gestisce gli eventi Eddystone accendendo dei led per dare un segnale visivo sullo stato degli advertising trasmessi.

```
static void on_es_evt(nrf_ble_es_evt_t evt)
{
    switch (evt)
    {
        case NRF_BLE_ES_EVT_ADVERTISEMENT_SENT:
            bsp_board_led_invert(NON_CONNECTABLE_ADV_LED_PIN);
        break;

        case NRF_BLE_ES_EVT_CONNECTABLE_ADV_STARTED:
```

```

    bsp_board_led_on(CONNECTABLE_ADV_LED_PIN) ;
break;

case NRF_BLE_ES_EVT_CONNECTABLE_ADV_STOPPED:
    bsp_board_led_off(CONNECTABLE_ADV_LED_PIN) ;
break;

default:
break;
}
}

```

A questo punto, nel **main()** dopo la funzione **services_init()** ho aggiunto la chiamata alla funzione **nrf_ble_es_init(on_es_evt)** la cui definizione è presente nel file **nrf_ble_es.c** precedentemente aggiunta come descritto nel capitolo 8.1.4. Tutte le funzionalità presenti nell'esempio Eddystone sono ora state implementate.

Per trasmettere gli iBeacon, nel file **mesh_main.c** sono state aggiunte le funzioni **adv_init()** e **adv_start()** per implementare e trasmettere iBeacon come descritto nel capitolo 6.2.1.

Debuggando il codice si presentano alcuni errori. Il primo è ERROR 4 [NRF_ERROR_NO_MEM] quando il programma ritorna dalla chiamata alla funzione **nrf_ble_escs_init()** che contiene **sd_ble_uuid_vs_add()**. Quest'ultima ritorna NRF_ERROR_NO_MEM quando non ci sono slot liberi per "vendor UUID". Per risolvere questo errore in **sdk_config.h** ho impostato Per risolverlo, nel file **sdk_config.h** si deve mettere

```
#define NRF_SDH_BLE_VS_UUID_COUNT 1
```

Dopo aver modificato questa macro, ho cambiato le dimensioni della ram dedicata al programma come suggerito dalle informazioni di debug. I valori da impostare nell'esempio sono riportati in tabella 5. Ulteriori informazioni sulle impostazioni della RAM e della FLASH sono disponibili nella discussione Adjustment of RAM and Flash memory. Per cambiare le dimensioni della

Softdevice	Version	Minimum RAM Start	FLASH Start
S132	7.0.0	0x20001668	0x26000

Tabella 5: RAM e FLASH start

RAM occorre andare in **Project → Edit Options → Common Configuration → Linker → Section Placement Macros**. Nel progetto corrente è composto da

- FLASH_PH_START=0x0
- FLASH_PH_SIZE=0x80000
- RAM_PH_START=0x20000000
- RAM_PH_SIZE=0x10000
- FLASH_START=0x26000
- FLASH_SIZE=0x5a000
- RAM_START=0x20002430
- RAM_SIZE=0xdbd0

sono state cambiate le macro RAM_START e RAM_SIZE precedentemente uguali a 0x20002420 e 0xdbe0 rispettivamente. Infatti per ogni nuovo UUID che si aggiunge occorre aumentare di 16 byte RAM_START e sottrarre 16 byte da RAM_SIZE, come indicato nella discussione `sd_ble_uuid_vs_add` error code 4. Si noti che **FLASH_SIZE** non deve essere più grande di (**FLASH_PH_SIZE - FLASH_START**), e che (**RAM_SIZE + RAM_START**) non deve essere più grande di (**RAM_PH_START + RAM_PH_SIZE**).

Successivamente si presenta l'errore ERROR 8 [NRF_ERROR_INVALID_STATE] dovuto alla chiamata alla funzione `es_battery_voltage_init()` in `nrf_ble_es_init()` perché inizializza il convertitore analogico digitale a successive approssimazioni (SAACD) che è stato già inizializzato precedentemente nel main.c dentro la funzione `adc_configure()`, già presente nell'esempio coexistence. Rimuovendo quest'ultima funzione si può risolvere l'errore perché l'ADC sarà inizializzato in `es_battery_voltage_init()`.

Risolto l'errore precedente, si ha ERROR 4 [NRF_ERROR_NO_MEM] nella funzione `sd_ble_gap_adv_set_configure()` contenuta in `adv_start`, nel file `es_adv.c`, che è richiamata da `es_adv_start_non_connectable_adv()` in `nrf_ble_es.c` dovuto al fatto che non c'è abbastanza memoria per configurare un nuovo handle di advertising. Per eliminare l'errore si possono rimuovere le funzioni `advertising_init()` e `advertising_start()`, già presenti nell'esempio coexistence, che aggiungono handle di advertising non necessari in questo progetto.

Continuando a debuggare il codice, quando si preme il pulsante 2 per mandare "connectable advertising" si ottiene ERROR 5 [NRF_ERROR_NOT_FOUND] quando viene chiamata la funzione `sd_ble_gap_adv_start` dichiarata come

```
sd_ble_gap_adv_start(uint8_t adv_handle, uint8_t
conn_cfg_tag);
```

Dalla documentazione si può osservare che se il parametro conn_cfg_tag non è stato trovato si ottiene questo errore. Poiché la funzione viene precedentemente chiamata per mandare iBeacon con conn_cfg_tag pari a 1, occorre che le macro MESH_SOFTDEVICE_CONN_CFG_TAG e BLE_CONN_CFG_TAG_DEFAULT siano uguali. Questo non è così di default perché nella libreria mesh_adv.h si ha

```
#define MESH_SOFTDEVICE_CONN_CFG_TAG 1
```

e in ble.h

```
#define BLE_CONN_CFG_TAG_DEFAULT 0
```

Risolto questo errore, si possono usare le funzionalità dell'esempio Eddystone nella rete Mesh.

8.3 Eliminazione delle funzionalità dell'esempio coexistence non necessarie

Per alleggerire il codice si possono eliminare le funzionalità dell'esempio coexistence che non sono necessarie. Sono stati rimosse le seguenti funzioni:

- saadc_event_handler
- adc_configure
- tps_init
- ias_init
- lls_init
- bas_init
- ias_client_init
- alert_signal
- on_ias_evt
- on_lls_evt
- on_ias_c_evt

- **on_bas_evt**

queste abilitano il convertitore ADC e servizi GATT non necessari e gestiscono i relativi eventi. Una volta rimosse queste funzioni, si possono eliminare i seguenti file dal progetto:

- ble_bas.c
- ble_ias.c
- ble_ias_c.c
- ble_lls.c
- ble_tps.c

e le seguenti librerie:

- ble_tps.h
- ble_ias.h
- ble_lls.h
- ble_bas.h
- ble_ias_c.h

Una volta rimosse le funzionalità non necessarie, le risorse utilizzate dal programma sono 373.8 KB di memoria flash su 512 KB disponibile e 44.6 KB di memoria RAM su 64 KB disponibili come si può vedere dalla figura 34.

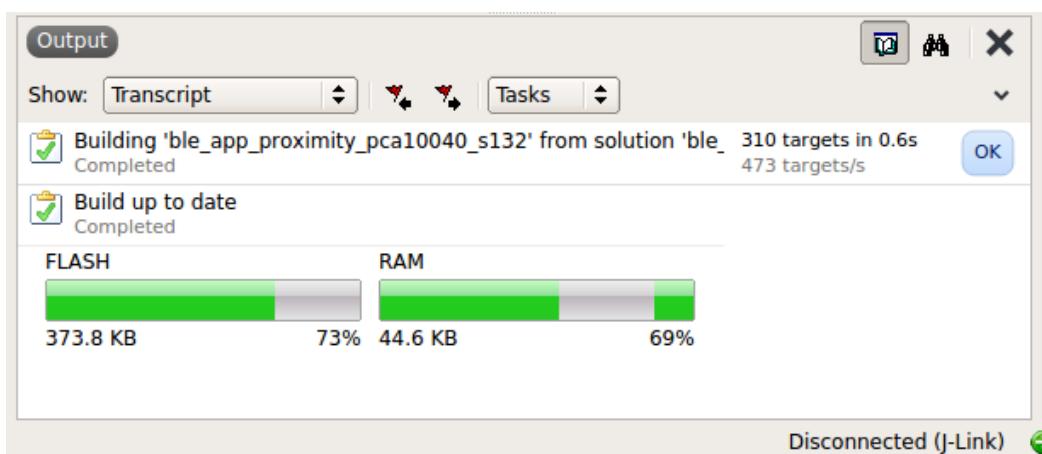


Figura 34: Risorse utilizzate

9 Conclusioni

L'applicazione realizzata nel progetto funziona correttamente. Nella relazione vengono descritte diverse modalità di implementazione delle funzionalità tra cui l'ultimo capitolo descrive l'implementazione finale.

Caricando il firmware nella board, usando l'applicazione nRF Connect si può anche notare che viene inviato un iBeacon, inoltre è possibile connettersi alla scheda e tramite il servizio GATT Eddystone Configuration Service si può configurare il beacon Eddystone trasmesso. Con questo programma non è possibile usare l'app nRF Mesh per fare il provisioning del nodo in quanto non è abilitato il Proxy GATT.

Per modificare l'intervallo di advertising per i beacon Eddystone è possibile cambiare le macro APP_CFG_NON_CONN_ADV_INTERVAL_MS e PP_CFG_CONN_ADV_INTERVAL_MS nella libreria es_app_config.h. Per cambiare l'intervallo di advertising degli iBeacon si può usare la macro BEARER_ADV_INT_DEFAULT_MS nella libreria nrf_mesh_config_bearer.h.

Riferimenti bibliografici

- [1] Ericsson, Lenovo, Intel Corporation, Microsoft Corporation, Motorola, Nokia Corporation e Toshiba Corporation. *Specification of the Bluetooth System*. Vol. 4.0. 2010. URL: <https://www.bluetooth.com/specifications/bluetooth-core-specification>.
- [2] Dixys Hernandez, Tiago Fernández-Caramés, Paula Fraga-Lamas e Carlos Escudero. «Design and Practical Evaluation of a Family of Lightweight Protocols for Heterogeneous Sensing through BLE Beacons in IoT Telemetry Applications». In: *Sensors* 18 (dic. 2017), p. 57. DOI: 10.3390/s18010057.
- [3] *Introduction to BLE IoT*. 2019. URL: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsdk_nrf5_v16.0.0%2Fiot_intro.html&cp=6_1_1_7.
- [4] *Light switch example*. 2019. URL: https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.meshsdk.v4.0.0/md_examples_light_switch_README.html?cp=7_2_3_0.
- [5] Joakim Lindh. *Bluetooth low energy Beacons*. Texas Instruments. 2015.
- [6] K. Townsend, C. Cufi, Akiba e R. Dividson. *Getting Started with Bluetooth Low Energy*. O'Reilly, 2014.
- [7] Martin Woolley. *Bluetooth Mesh Networking*. 2017. URL: <https://www.bluetooth.com/specifications/mesh-specifications>.