Università degli Studi di Brescia
Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

TESI DI LAUREA

Sistemi flessibili per la cattura di traffico bluetooth

Laureando:
Stefano Orioli

Matricola:
72452

Relatore:
Prof. Francesco Gringoli

Anno Accademico 2016/2017

Indice

1 Introduzione

Il Bluetooth è una tecnologia di comunicazione a corto raggio che esiste ormai da parecchi anni sul mercato, ma con l'avvento della specifica Low Energy, introdotta nel 2010 nella versione 4.0 e caratterizzata da un notevole risparmio in termini energetici, è stata adottata da una gran quantità di dispositivi, anche destinati a usi differenti. Basti pensare a tutti quei dispositivi che si definiscono 'Smart', dalle Televisioni agli Smartphone, dai PC agli orologi, la tecnologia ha un grande impatto sulla nostra vita di tutti i giorni. Questa rivoluzione comunicativa prende il nome di IoT[1], il mondo dei dispositivi fisici, dove anche il più piccolo di essi, con capacità di calcolo limitate, è in grado di comunicare la propria presenza ed ottenere informazioni sulla rete proprio grazie a queste nuove tecnologie.

É quindi corretto chiedersi: queste tecnologie sono sicure? I danni che un malintenzionato sarebbe capace di provocare se riuscisse a manipolare a piacimento il comportamento di questi dispositivi, sono innumerevoli; Basti pensare ad una serratura intelligente che sblocca la porta di casa quando il proprietario si trova di fronte ad essa, e se non fosse il proprietario quello davanti alla porta, ma tramite un dispositivo di ritrasmissione si spacciasse per esso? Lo stesso discorso si può applicare alle serrature ed al sistema di avvio delle recenti automobili dotate di sistema Keyless[2]. Restando in campo informatico, sempre più aziende stanno adottando soluzioni Smart per accedere ai PC, senza dover inserire manualmente la classica password; Apple da la possibilità ai propri utenti di sbloccare il MacBook tramite il loro Apple Watch, semplicemente avvicinandolo allo schermo; un malintenzionato riuscirebbe con facilità a rubare i dati personali di un utente se riuscisse a sfruttare una falla di questo meccanismo di sblocco.

É proprio su questi presupposti che si basa questo progetto di tesi, testare la sicurezza del protocollo Bluetooth tramite la creazione di uno Sniffer a basso costo che permetta di visualizzare ed analizzare i pacchetti scambiati da due dispositivi durante tutta la durata di una connessione. L'implementazione di uno Sniffer è utile anche per un'attività di Debug nello sviluppo delle applicazioni; permette di vedere realmente la composizione del pacchetto inviato e quindi di trovare facilmente errori nell'applicativo. Uno degli sviluppi futuri è quello riuscire a creare un ripetitore di segnale tra due punti, basandosi sul lavoro svolto in questa tesi, che faccia credere ai dispositivi di essere a stretto contatto quando in realtà li separa una distanza maggiore; questo dimostrerebbe che un dispositivo Bluetooth Low Energy è vulnera-

[1] Internet of Things, ovvero l'internet delle cose.

[2] Senza chiavi, permette l'apertura e l'avvio dell'automobile semplicemente tenendo in tasca le chiavi.

bile ad attacchi di tipo Relay, e forzerebbe i costruttori a risolvere queste vulnerabilità e quindi a migliorare la sicurezza del protocollo.

2 Sistemi esistenti

Sul mercato esistono già prodotti di sniffing completi, funzionanti e pronti per essere usati; il problema di questi dispositivi è che sono costosi e per la maggior parte non Open Source. La principale limitazione di ciò è che il codice su cui si basano non può essere manipolato a piacimento secondo le proprie esigenze e quindi questi programmi possono essere unicamente usati per lo scopo per cui sono stati creati. Se uno sviluppatore in possesso di uno di questi dispositivi a Closed Source volesse cambiare un leggero aspetto di un programma di Sniffing preconfezionato dovrebbe riscriversi completamente il codice da zero, con un enorme dispendio di tempo e risorse per simulare il comportamento di un applicativo già esistente.

2.1 Ubertooth One

Ubertooth One è un progetto Open Source di sviluppo su reti Wireless, capace di comunicare utilizzando il protocollo trasmissivo BLE. Sulla loro pagina GitHub [9] è possibile reperire, oltre ai vari progetti sviluppati appositamente, anche il disegno delle componentistiche Hardware, per poter assemblarlo e costruirlo privatamente o addirittura modificarlo secondo le proprie necessità.

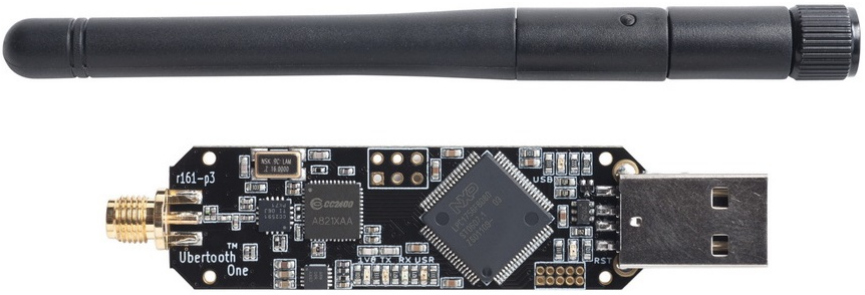


Figura 1: UberTooth One

Si hanno ha disposizione vari progetti creati per questo dispositivo, tra i tanti va citato 'Bluetooth Captures in PCAP'[10] che permette di ascoltate, in una sorta di modalità promiscua, tutte le comunicazioni bluetooth su un certo canale, oppure tramite l'ascolto di una CONNECT_ REQ. seguire una connessione. Permette ulteriormente di interfacciarsi con l'applicazione

WIRESHARK[3] per visualizzare in tempo reale il traffico BLE, applicare filtri su indirizzi o potenza del segnale ed inviare pacchetti personalizzati.

L'aspetto negativo di questa soluzione sta nel costo, difatti ad oggi si trova in commercio ad un prezzo che si aggira attorno ai 150€ + spese di consegna [8].

2.2 Bluefruit LE sniffer

Il Bluefruit LE monta al suo interno il chip della Nordic nRF51822 che viene venduto già programmato con un software che trasforma il dispositivo in uno sniffer di traffico Bluetooth Low Energy. I pacchetti raccolti vengono mandati in automatico a Wireshark dove possono essere visualizzati con un'opportuna struttura che facilita la comprensione delle varie parti del pacchetto, senza far riferimento al manuale Bluetooth Core.

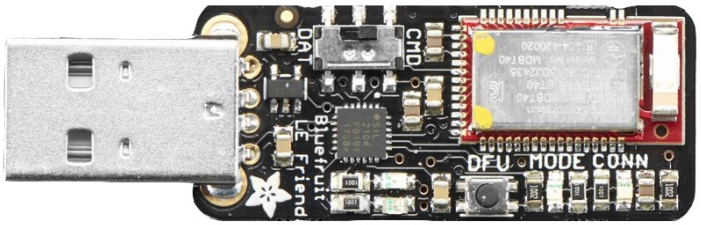


Figura 2: Bluefruit LE Sniffer - Bluetooth Low Energy (BLE 4.0) - nRF51822

Le limitazioni di questa soluzione sono molteplici; il software che viene installato per gestire il dispositivo come sniffer non è open source; il tool che permette l'interfacciamento tra il dispositivo e wireshark esiste solo per l'ambiente windows. Il dispositivo così come è venduto non può essere programmato, necessita infatti di un programmatore esterno, come il J-Link venduto ad un costo che si aggira sui 15€. Il costo del dispositivo è di circa 25€ paragonabile al prezzo di acquisto di un RedBear Nano2.

[3]É un software per analisi di protocolli o pacchetti utilizzato per la soluzione di problemi di rete, per l'analisi e lo sviluppo di protocolli o di software di comunicazione.

2.3 nRF52 DK

La board di sviluppo nRF52 DK della Nordic Semiconductor è una periferica di sviluppo versatile e completa per testare applicazioni BLE, ANT e protocolli proprietari che usano le frequenze 2.4GHz. Integra 4 led e 4 bottoni che possono essere usati rispettivamente per ricevere informazioni sullo stato di funzionamento del dispositivo e per impartire comandi a quest'ultimo.

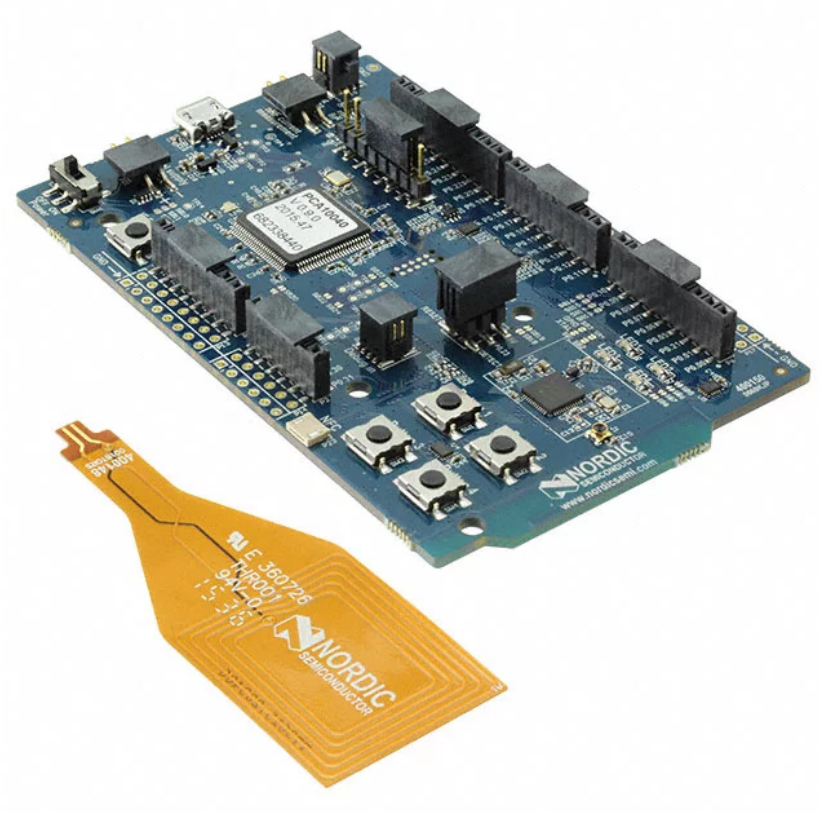


Figura 3: Board di sviluppo NRF52_DK creata dalla società Nordic Semiconductor.

Essendo ufficialmente supportata dalla stessa Nordic, lo sviluppatore ha a disposizione molti tool per lo sviluppo e il testing di applicazioni. Esiste anche un software di sniffing preconfezionato che permette di svolgere tutte le operazioni di sniffing direttamente da wireshark, tra cui la possibilità di seguire in maniera del tutto automatizzata una connessione e vedere in tempo reale la composizione e il significato di tutti i pacchetti scambiati. Sfortunatamente questo software esiste solo per windows e non è open source, quindi non è utilizzabile per lo scopo di questa tesi. Oltretutto il costo di questa

board di sviluppo è di circa 40€ più spese di consegna, più alto di quello di un Nano2.

2.4 USRP EttusB210

Il B210 fornisce una singola scheda integrata su una piattaforma USRP[4] con una copertura continua di frequenza da 70 Mhz a 6 GHz. Progettato per la sperimentazione a basso costo, combina il ricetrasmettitore di conversione diretta RFC AD9361 che fornisce fino a 56 MHz di larghezza di banda in tempo reale, un FPGA[5], Spartan6 XC6SLX150 open-source e riprogrammabile, ed una connettività ad alta velocità utilizzando USB 3.0 con canale di alimentazione dedicato. Il B210 consente un facile e veloce sviluppo con il toolkit di sviluppo software GNURadio, consentendo così di sperimentare con un'ampia gamma di segnali e bande di frequenza, inclusi trasmissioni FM, TV, cellulari, Wi-fi e Bluetooth.

Figura 4: USRP Ettus B210, Ettus Research una società della compagnia National Instrument.

La massima banda trasmissiva campionabile in tempo reale con questo dispositivo è di 56 MHz a 61,44 MS/s[6] inferiore alla banda trasmissiva del Bluetooth Low Energy. Il costo di questa board è molto elevato rispetto alle

[4]Universal Software Radio Peripheral, componente della Ettus Research di tipo Software-defined radio (SDR)

[5]Field Programmable Gate Array, circuito integrato le cui funzionalità sono programmabili via software

[6]Mega Sample al secondo, unità di misura della frequenza di campionamento.

precedenti soluzioni, essendo di circa 1.250€ . Per questo progetto disponiamo comunque di due di questi dispositivi, essendo stati acquistati dall'università in precedenza, che riescono a coprire gli 80 MHz di banda trasmissiva BLE. Si sono rilevati molto utili per capire ciò che realmente passa nell'etere a quelle frequenze. Il dispositivo campiona qualsiasi segnale nella banda trasmissiva, creando una gran quantità di dati che devono essere poi analizzati e filtrati con un software creato appositamente per estrarne i pacchetti BLE di nostro interesse.

2.5 Frontline BPA Low Energy

Progettato e commercializzato dalla società TELEDYNE LECROY, il Frontline BPA Low Energy Bluetooth protocol analyzer è un dispositivo creato appositamente per la funzione di sniffing di pacchetti BLE. Utilizzabile unicamente su sistemi Windows, viene venduto con un applicativo, non open source, da installare che permette di visualizzare tutto il traffico Bluetooth LE in tempo reale, salvare tutto ciò che è stato catturato ed analizzarlo nel dettaglio; con opportuni filtri risulta molto facile trovare il pacchetto di interesse tra gli svariati catturati e vederne nel dettaglio tutte le componenti con i loro valori. É interfacciabile anche con WireShark, permettendo quindi all'utente di usufruire delle potenzialità della piattaforma ed integrarlo in progetti già esistenti.

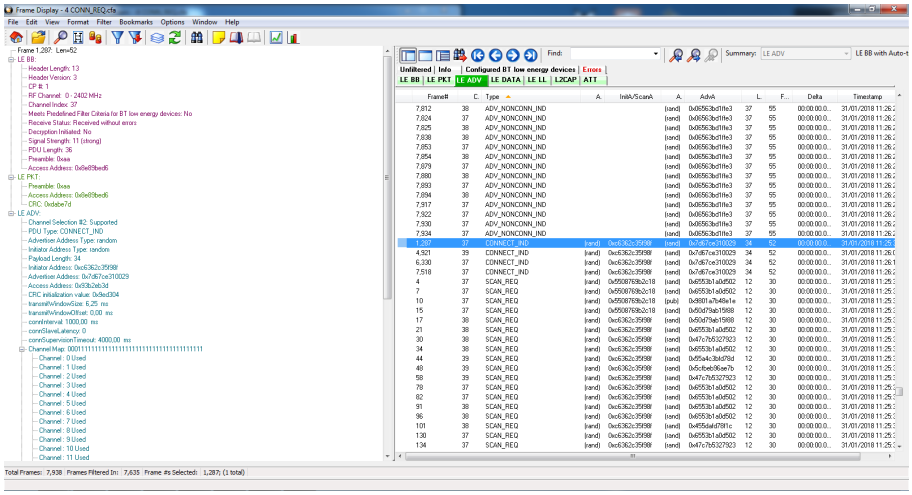


Figura 5: Schermata del software di analisi del traffico ble, distribuito con il dispositivo BPA di sniffing.

É stato utilizzato come supporto per lo sviluppo del progetto di tesi, in quanto è stato possibile averne uno a disposizione; si è rivelato utile per co-

noscere esattamente il valore dei vari campi, sopratutto di quelli composti da pochi bit, che il dispositivo nano2 tendeva ad inviare secondo un ordine che non era quello delle specifiche BLE. Conoscendo esattamente il nome del campo ed il relativo valore, fornito dal software di analisi utilizzato con lo sniffer BPA, è stato quindi possibile risalire all'esatta posizione dello stesso all'interno di ciò che veniva catturato dal dispositivo della RedBear, riuscendo a mappare l'esatta disposizione di tutti i campi del pacchetto. Sfortunatamente i documenti che mette a disposizione la società che ha creato il Nano2 sono scarsamente dettagliati e privi di molte informazioni essenziali, si è perciò reso necessario appoggiarsi su strumenti più professionali per capire come funzionasse esattamente e quindi per poter procedere con lo sviluppo di uno Sniffer a basso costo. Il dispositivo è disponibile sul mercato ad un costo decisamente elevato, che si aggira attorno ai 900€ .

3 Bluetooth Low Energy

3.1 Introduzione

Il Bluetooth Low Energy (BLE) è una tecnologia wireless introdotta nello standard 4.0, disegnata e commercializzata dal SIG[7]; ideata per trasmissioni di dati di breve dimensione, essa si differenzia dal Bluetooth classico per un inferiore consumo di energia, mantenendo la stessa portata trasmissiva.

3.2 Banda Trasmissiva

BLE opera alla stessa banda di frequenza del bluetooth classico, 2,400 - 2,4835 GHz, ma utilizza un numero inferiore di canali, 40 canali da 2 MHz l'uno. Nel canale trasmissivo i dati sono trasmessi con una modulazione GFSK.[8] I bitrate trasmissivi sono limitati ai valori 125 kbit/s, 1 Mbit/s o 2 Mbit/s.

3.2.1 Mapping tra canali e frequenze

Il protocollo BLE utilizza 40 canali fisici differenti per la trasmissione, con una numerazione che va da 0 a 39. Ogni canale ha una frequenza centrata in $2402 + (k \cdot 2)$ MHz, dove k assume il valore del canale considerato. Questi 40 canali si dividono in:

- 3 canali 0, 12 e 39 centrati nelle frequenze 2402 MHz, 2426 MHz e 2480 MHz ed utilizzati per l'Advertise.
- i restanti 37 canali sono utilizzati per la trasmissione di dati che vengono utilizzati solo a connessione avvenuta per lo scambio di informazioni.

La distribuzione di questi canali non è casuale: essi difatti sono i canali utilizzati da un dispositivo per far conoscere la propria presenza ai vicini, è quindi fondamentale che almeno uno di essi sia disponibile per la trasmissione, quindi privo di disturbi generati da altri sistemi che operano nella stessa banda trasmissiva. Sono stati distribuiti lontani tra di loro cercando di utilizzare tutta la banda disponibile per massimizzare la possibilità di avere un canale libero per la trasmissione. Un altro motivo della scelta è stato quello di poter facilitare la co-esistenza con la tecnologia Wifi, che utilizza la stessa banda per trasmettere, centrata principalmente nei canali Wifi 1, 6 e 11.

[7] Bluetooth Special Interest Group: organizzazione che regolamenta e definisce gli standard per la trasmissione dati tramite la tecnologia Bluetooth.

[8] Gaussin Frequency Shift Keying, limita la larghezza dello spettro trasmissivo tramite un filtro di tipo Gaussiano.

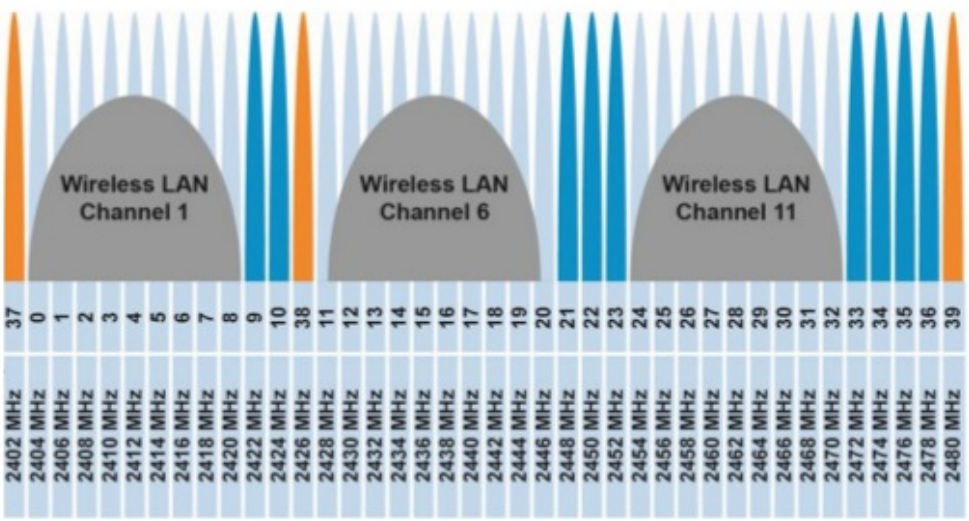
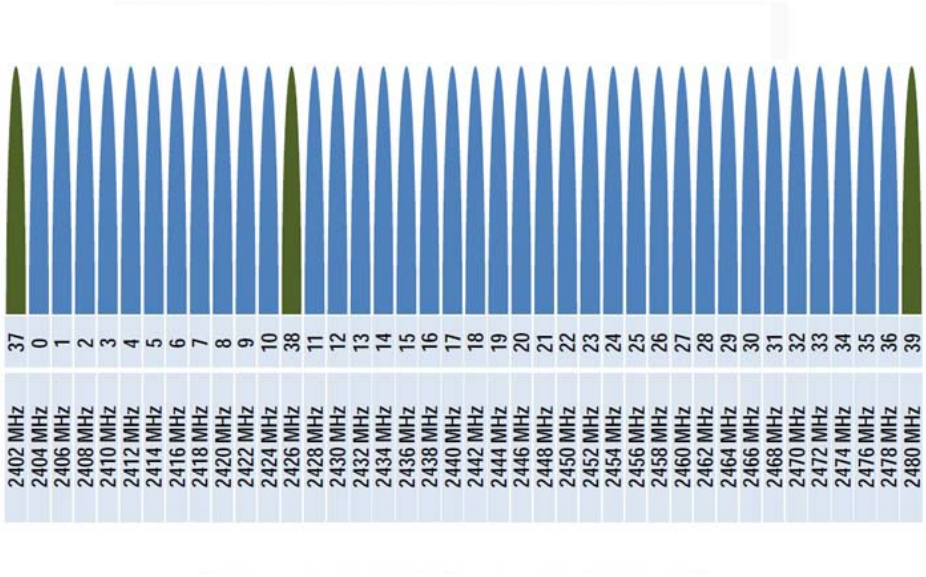


Figura 6: sovrapposizione alla banda BLE dei 3 canali usati principalmente nella comunicazione wifi; si noti come i 3 canali di advertise non rientrino in nessuno di questi 3 canali.

Per una miglior leggibilità ed un più facile utilizzo, si è scelto di mappare i canali reali con dei corrispondenti indici fittizi che vedono nelle numerazioni da 0 a 36 i canali data e 37, 38, 39 per i canali di Advertise;

- canale fisico 0, di Advertise a frequenza 2402 MHz, mappato come canale 37.
- canale fisico 12, di Advertise a frequenza 2426 MHz, mappato come canale 38.
- canale fisico 39, di Advertise a frequenza 2480 MHz, mappato come canale 39.
- canale fisico 1, di scambio dati a frequenza 2404 MHz, mappato come canale 0.
- ...
- canale fisico 38, di scambio dati a frequenza 2478 MHz, mappato come canale 36.

Figura 7: Distribuzione dei canali nella banda trasmissiva BLE, con il riferimento all'indice del canale mappato e la relativa frequenza centrale.



3.2.2 Adaptive Frequency Hopping

Bluetooth Low Energy utilizza il *Frequency Hopping*, una tecnica trasmissiva che consiste nel cambiare canale trasmissivo per ogni pacchetto inviato durante una trasmissione; l'utilizzo di questa tecnica permette di ridurre i problemi di interferenza con gli altri canali ed allo stesso tempo di aumentare la sicurezza delle trasmissioni. Esso è definito adattivo perché permette, in fase di connessione, di andare ad escludere determinati canali perché considerati troppo disturbati. L'algoritmo che regola questo cambio di canali è determinato da una formula che calcola il successivo canale in cui trasmettere, avendo come parametro l'ultimo canale usato e il numero di canali da saltare

$$Ch = (Ch_{-1} + HopIncrement) mod 37$$

Dove:

Ch: è il prossimo canale in cui trasmettere.

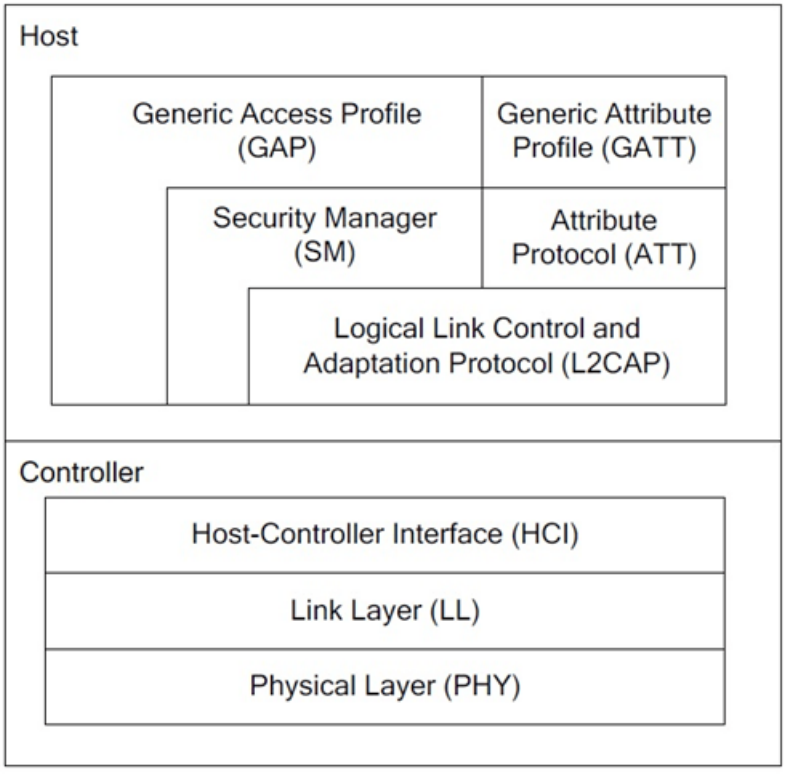
Ch_{-1}: è l'ultimo canale in cui si è trasmesso.

HopIncrement : costante che indica il numero di canali da saltare, che viene scambiato dai dispositivi in fase di connessione.

3.3 Stack protocollare

Storicamente lo stack Bluetooth è diviso in 2 componenti fondamentali: il *Controller* e l'*Host*. La parte di Controller è quella che si occupa di eseguire le componenti di basso livello dello stack necessarie per gestire i pacchetti scambiati a livello fisico e le loro temporizzazioni; la parte di Host comprende le componenti di alto livello tra cui profili e API[9]. La parte di Host, diversamente da quella di controllo, astrae dall'hardware ed è caratterizzata da una gestione meno rigida delle temporizzazioni.

L'*HCI*[10], l'interfaccia tra Host e Controller si occupa di mettere in comunicazione queste due componenti fondamentali, utilizzando canali comunicativi quali UART o USB. Se i due componenti sono montati su uno stesso chip, come nel caso dei SoC[11], allora l'interfaccia HCI è opzionale e può essere omessa.

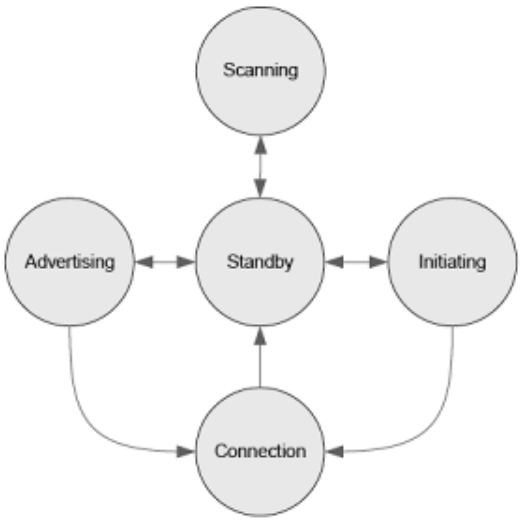


[9]Application Program Interface: insieme di procedure utili a svolgere uno specifico compito

[10]Host Controller Interface

[11]System on a Chip: sistema completamente contenuto su un solo Chip

- Physical Layer (PHY): è il livello fisico, ovvero l'etere; lavora nella banda di frequenza 2,4 GHz ISM[12] Si compone di 40 canali da 2 MHz ognuno suddivisi in 3 per advertising e 37 per lo scambio dati.
- Link Layer (LL): si occupa della gestione della sequenza e della temporizzazione dei pacchetti scambiati. Dialoga con i nodi vicini e scambia informazioni quali i parametri di connessione e controllo di flusso. É una macchina a stati che si compone di 5 stati: Standby, Advertising, Scanning, Initiating, Connection.



  - Advertising: il dispositivo rende noto agli ascoltatori della propria presenza, indicando la disponibilità ad una connessione ed inviando alcune informazioni utili.
  - Scanning: il dispositivo è in ascolto di tutte le informazioni trasmesse sui canali di advertise.
  - Initiating: un dispositivo in ascolto individua un Advertise di suo interesse ed indica la sua volontà di connettersi.
  - Connection: due o più dispositivi sono connessi.
  - Standby: il dispositivo è in uno stato di attesa caratterizzato da un basso consumo energetico.

  Lo stato di Scanning può essere attivo (richiede informazioni aggiuntive) o passivo; Lo stato Connection anch'esso si divide in 2 sottostati: Central e Peripheral.

[12]Industrial, Scientific and Medical: frequenze riservate alle applicazioni di radiocomunicazioni non commerciali, ovvero per uso industriale, scientifico e medico.

- Logical Link Control and Adaptation Protocol (L2CAP): fornisce servizi sui dati ai livelli superiori, come ad esempio il Security Manager Protocol o l'Attribute Protocol. É responsabile inoltre della segmentazione e ricostruzione dei pacchetti da e verso i livelli inferiori.

- Security Manager (SM): è responsabile del pairing dei dispositivi e della distribuzione delle chiavi crittografiche; BLE utilizza lo standard AES-128 bit per la cifratura dei dati ed il sistema di pairing per la distribuzione delle chiavi.

- Attribute Protocol (ATT): gestisce la distribuzione delle informazioni riguardanti le coppie attributo-valore presenti in un device peripheral (Server)

- Generic Access Profile (GAP): controlla *advertising* e *connection* . Divide i dispositivi bluetooth low energy in:

  - Peripheral: normalmente dispositivi dotati di una limitata capacità di calcolo, come ad esempio un sensore di temperatura.
  - Central: dispositivo che gestisce la rete bluetooth e che richiede una maggiore capacità di calcolo; mentre un dispositivo periferico può avere una connessione con un solo dispositivo centrale, un centrale può gestire più dispositivi periferici andando a creare una Piconet[13]

- Generic Attribute Profile (GATT): entra in gioco a connessione avvenuta, definisce il modo in cui 2 dispositivi BLE scambiano dati utilizzando i concetti di *Services* e *Characteristics*.

  - GATT server: è un peripheral device, che tramite il protocollo ATT permette al central di conoscere i dati che ha memorizzato in strutture di tipo service-characteristic.
  - GATT client: è il central device, che gestisce le comunicazioni e che richiede ai server periferici i dati raccolti.

3.4 Formato dei pacchetti

Il Link Layer (LL) ha un unico formato per tutti i pacchetti scambiati, che quindi è uguale sia per i pacchetti di tipo Advertise, sia per i pacchetti di scambio di dati. In figura 8 è mostrato il formato dei pacchetti

[13]Rete Bluetooth composta da massimo otto dispositivi in relazione master-slave e fino a 255 dispositivi in modalità inattiva o parcheggio.

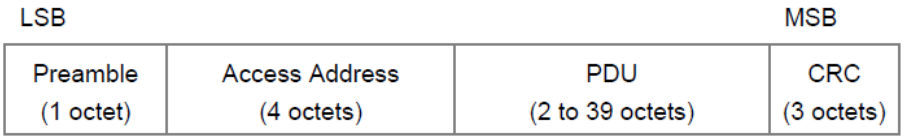


Figura 8: formato dei pacchetti Link Layer

- **Preamble**: composto da 8 bit, è utile al ricevitore del pacchetto per sincronizzarsi sulla frequenza portante, e per impostare l'*Automatic Gain Control* (AGC). I pacchetti di Advertise hanno come preambolo 10101010*b*. Per i pacchetti data il valore del preambolo è determinato dal LSB[14] dell'Access Address; se è 1 il preambolo è 01010101*b*, se è 0 il preambolo vale 10101010*b*

- **Access Address**: Campo composto da 4 Byte; per i pacchetti di Advertise assume un valore costante pari a 0x8E89BED6. Per le trasmissioni di dati l'Access Address deve essere diverso per ogni connessione; dovrebbe essere un valore casuale (con alcune restrizioni) di 32 bit generato dal dispositivo che inizia la connessione ed inviato al secondo dispositivo durante la connessione tramite il pacchetto di Connection Request.

- **PDU**: Composta da un numero variabile da 2 a 39 byte, è l'acronimo di Protocol Data Unit, ovvero l'unità di informazione scambiata. A seconda che si abbia un pacchetto di Advertise o un pacchetto dati, la PDU assume una struttura differente. I vari tipi di PDU sono trattati nel capitolo 3.5

- **CRC**: Alla fine di ogni pacchetto Link Layer c'è un CRC,Il cyclic redundancy check (ovvero controllo a ridondanza ciclica), di 24 bit. Esso serve per rilevare eventuali errori di trasmissione, quindi per sapere se il pacchetto ricevuto è corrotto. Il CRC viene calcolato sul campo PDU, partendo dal LSB; il polinomio per il calcolo è:

  $$x^24 + x^10 + x^9 + x^6 + x^4 + x^3 + x + 1$$

  Se il calcolo del CRC viene fatto su un PDU di tipo Advertise, deve essere inizializzato con il valore 0x555555; Se viene fatto su un pacchetto dati, deve essere inizializzato ad un valore scambiato in fase di comunicazione (di 3 Byte), contenuto nella Connection Request. In posizione

[14]Less Significant Bit, il bit meno significativo.

0 va inserito il bit meno significativo del valore di inizializzazione, ma il CRC viene inviato con il bit più significativo (bit 23) in posizione iniziale.

3.5 PDU dei canali di Advertise

Il PDU di un pacchetto di Advertise si compone di un header di 16 bit ed un payload con una lunghezza variabile dai 6 ai 37 Byte.

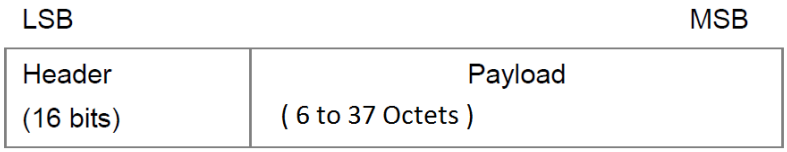


Figura 9: Advertising PDU

L'Header di questo PDU si compone di vari campi, come mostrato in figura 10

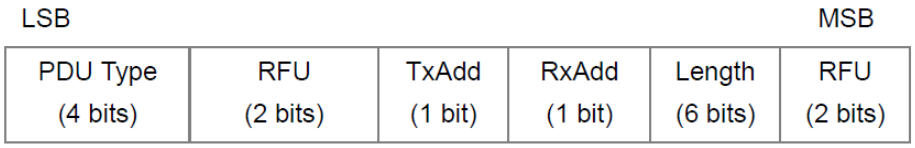


Figura 10: Advertising PDU Header

Il campo PDU type verrà approfondito in seguito, assieme al significato dei campi TxAdd e RxAdd; Il campo length, di 6 bit quindi con un valore massimo di 63, ma settabile dalle specifiche ad un valore massimo di 37, indica i Byte di dimensione del Payload.

3.5.1 PDU type

PDU Type $b_3b_2b_1b_0$	Packet Name
0000	ADV_IND
0001	ADV_DIRECT_IND
0010	ADV_NONCONN_IND
0011	SCAN_REQ
0100	SCAN_RSP
0101	CONNECT_REQ
0110	ADV_SCAN_IND
0111-1111	Reserved

Figura 11: Codifiche dei possibili tipi di PDU di un pacchetto di Advertise

I vari tipi di PDU di un canale di Advertise si raggruppano in 3 insiemi distinti: Advertising PDU, Scanning PDU e Initiating PDU.

3.5.2 Advertising PDU

Questi PDU sono inviati da un dispositivo in stato di Advertise e ricevuti da un sistema in stato di scanning.

- ADV_ IND: evento di advertising che notifica la disponibilità ad una connessione indiretta, ovvero senza specifiche sull'indirizzo da cui ricevere la richiesta di connessione. Il payload di questo tipo di PDU è mostrato in figura 12, il campo TxAdd dell'header se settato a 1 indica che l'indirizzo del campo AdvA del payload è casuale, se settato a 0 esso è pubblico.

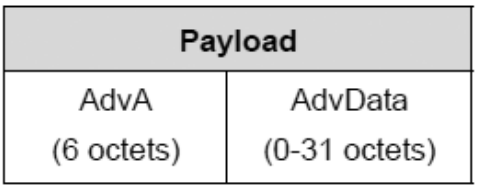


Figura 12: Payload di un PDU di tipo ADV_ IND

Si compone dei campi AdvA di 6 byte, che indica l'indirizzo, pubblico o privato, dell'advertiser, ovvero di chi sta inviando il pacchetto, e il campo AdvData che contiene eventuali informazioni aggiuntive ed ha una lunghezza massima di 31 Byte.

- ADV_ DIRECT_ IND: evento di connessione diretta, in cui nel payload, mostrato in figura 13 è indicato l'indirizzo da cui ci si aspetta una connessione. Il campo TxAdd dell'header se settato a 1 indica che l'indirizzo del campo AdvA del payload è casuale, se settato a 0 esso è pubblico; il campo RxAdd dell'header se settato a 1 indica che l'indirizzo del campo InitA del payload è casuale, se settato a 0 esso è pubblico.

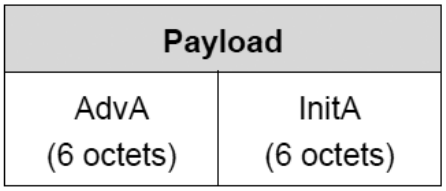


Figura 13: Payload di un PDU di tipo ADV_ DIRECT_ IND

Il campo AdvA contiene l'indirizzo dell'Advertiser, sia esso pubblico o privato, ed il campo InitA indica l'indirizzo da cui ci si aspetta la connessione. Il payload di questo tipo non contiene nessuna informazione aggiuntiva.

- ADV_ NONCONN_ IND: evento di non connessione; utilizzato per applicazioni di tipo BEACON, ovvero per condividere informazioni a chiunque è in ascolto. Nel payload, mostrato in figura 14 è indicato l'indirizzo dell'Advertiser. Il campo TxAdd dell'header se settato a 1 indica che l'indirizzo del campo AdvA del payload è casuale, se settato a 0 esso è pubblico;

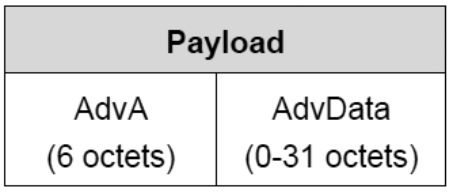


Figura 14: Payload di un PDU di tipo ADV_ NONCONN_ IND

Il campo AdvA contiene l'indirizzo dell'Advertiser, sia esso pubblico o privato, ed il campo AdvData che contiene le informazioni da condividere ed ha una lunghezza massima di 31 Byte.

- ADV_ SCAN_ IND: evento di connessione indiretta in cui si informa che si hanno ulteriori informazioni da condividere; chi è interessato a queste informazioni manderà all'Advertiser un PDU di tipo SCAN_ REQ, e l'advertiser risponderà con un PDU di tipo SCSN_ RSP contenente le informazioni aggiuntive. Nel payload, mostrato in figura 15 è indicato l'indirizzo dell'Advertiser. Il campo TxAdd dell'header se settato a 1 indica che l'indirizzo del campo AdvA del payload è casuale, se settato a 0 esso è pubblico;

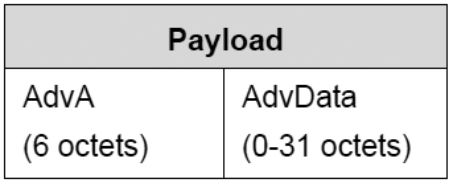


Figura 15: Payload di un PDU di tipo ADV_ SCAN_ IND

Il campo AdvA contiene l'indirizzo dell'Advertiser, sia esso pubblico o privato, ed il campo AdvData che contiene le eventuali informazioni da condividere ed ha una lunghezza massima di 31 Byte.

3.5.3 Scanning PDU

- SCAN_ REQ: dopo aver ricevuto un pacchetto di Advertise, uno scanner in stato attivo può richiedere eventuali informazioni aggiuntive con questo tipo di PDU. Il campo TxAdd dell'header se settato a 1 indica che l'indirizzo del campo AdvA del payload, figura 16 è casuale, se settato a 0 esso è pubblico; il campo RxAdd dell'header se settato a 1

indica che l'indirizzo del campo InitA del payload è casuale, se settato a 0 esso è pubblico.

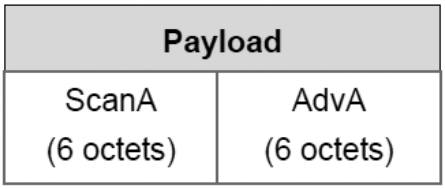


Figura 16: Payload di un PDU di tipo SCAN_ REQ

Nel campo ScanA viene indicato l'indirizzo del device che genera questo pacchetto di SCAN_ REQ, mentre nel campo AdvA viene indicato l'indirizzo del dispositivo in stato di Advertise a cui è destinato questo pacchetto.

- SCAN_ RSP: dopo aver ricevuto un pacchetto di tipo SCAN_ REQ destinato a lui, l'Advertiser può decidere di adempire alla richiesta inviando questo tipo di PDU con le informazioni aggiuntive che vuole fornire. Il campo TxAdd dell'header se settato a 1 indica che l'indirizzo del campo AdvA del payload, figura 17 è casuale, se settato a 0 esso è pubblico;

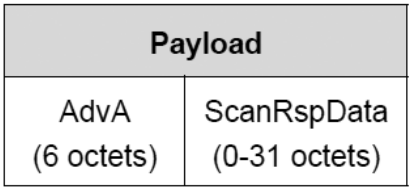


Figura 17: Payload di un PDU di tipo SCAN_ RSP

Nel campo AdvA viene indicato l'indirizzo del device che genera questo pacchetto di SCAN_ RSP, mentre nel campo ScanRspData viene inserita l'eventuale informazione aggiuntiva da condividere, con lunghezza massima di 31 Byte.

3.5.4 Initiating PDU

CONNECT_ REQ: inviata da un dispositivo in stato di scanning, consente a quest'ultimo di effettuare una connessione con il dispositivo in stato di Advertising che ha dichiarato la sua disponibilità a connettersi.

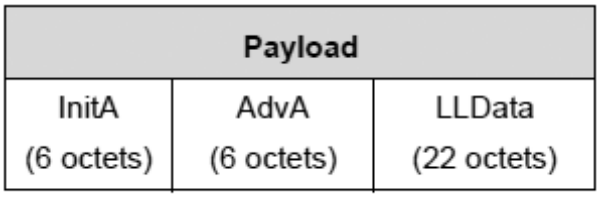


Figura 18: Payload di un PDU di tipo CONNECT_ REQ

Il payload è mostrato in figura 18 e si compone di 3 campi principali:

- InitA (6 Byte): l'indirizzo del dispositivo che ha avviato la connessione, ovvero chi ha mandato il pacchetto di CONN_ REQ;
- AdvA (6 Byte): l'indirizzo dell'Advertiser, quello a cui vuole connettersi chi ha iniziato la connessione;
- LLData (22 Byte): è il cuore del pacchetto; qui si trovano tutte le informazioni sui parametri da usare durante la connessione.

Una trattazione riservata va fatta al campo LLData, mostrato il figura 19 che si compone di 10 campi.

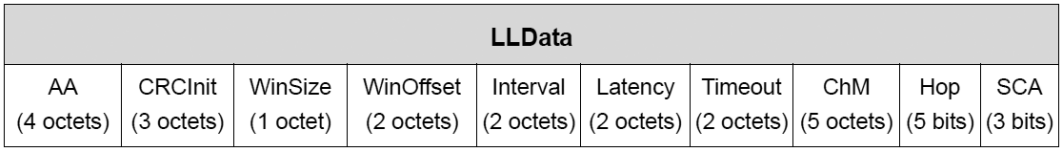


Figura 19: Campi di una struttura LLData di un pacchetto PDU di tipo CONNECT_ REQ

Nel dettaglio i campi sono:

- AA (4 Byte): il campo contiene l'Access Address (specificato nel capitolo 3.4) utilizzato per tutta la durata della connessione.
- CRCInit (3 Byte): contiene il valore a cui inizializzare il calcolo del CRC per la connessione a livello Link Layer. É un valore casuale, generato dal dispositivo che avvia la connessione.
- WinSize: determina il valore di $transmitWindowSize = WinSize \cdot 1.25$ ms, spiegato in dettaglio nel paragrafo 3.7.

- WinOffset: determina il valore di $transmitWindowOffset = WinOffset \cdot 1.25$ ms, spiegato in dettaglio nel paragrafo 3.7.
- Interval: determina il valore di $connInterval = Interval \cdot 1.25$ ms, spiegato in dettaglio nel paragrafo 3.7.
- Latency: indica il numero di volte che lo slave può non rispondere nella finestra di *connInterval* pur mantenendo valida la connessione (*connSlaveLatency*).
- Timeout: determina il valore di $connSupervisionTimeout = Timeout \cdot 10$ ms, spiegato in dettaglio nel paragrafo 3.7.1.
- ChM (5 byte): indica quali tra i canali destinati allo scambio di informazioni durante una connessione (vedi capitolo 3.2.1) devono essere usati. Ogni canale è rappresentato da un bit, il cui valore 1 indica che verrà utilizzato, mentre il valore 0 che non lo sarà. I bit 37, 38 e 39, che indicano i canali di Advertise, non sono utilizzati (RFU).
- Hop(5 bit): il campo hop indica il valore di HopIncrement (Capitolo 3.2.2) che verrà utilizzato dall'algoritmo per determinare di volta in volta il canale su cui trasmettere.
- SCA(3 bit): determina l'accuratezza dello *Sleep Clock*.

3.6 PDU dei canali Data

Il PDU che viene inviato nei canali di scambio dati, ovvero dal canale 0 al canale 36, ha la struttura mostrata in figura 20 e si compone di un header da 16 bit, payload e un campo MIC (*Message Integrity Check*) opzionale.

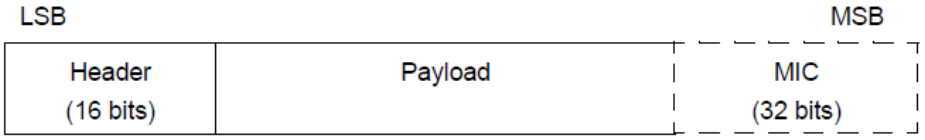


Figura 20: PDU di un pacchetto Data

L'header di questo pacchetto ha 5 campi, mostrati in figura 21, analizzati nel dettaglio di seguito:

Header						
LLID (2 bits)	NESN (1 bit)	SN (1 bit)	MD (1 bit)	RFU (3 bits)	Length (5 bits)	RFU (3 bits)

Figura 21: Header di un PDU di tipo Data

- LLID: da informazioni sul tipo di pacchetto, che può essere di *LL Data* o *LL Control*
  - 00b: non utilizzato.
  - 01b: PDU di tipo LL Data, può essere la continuazione di un messaggio L2CAP oppure un PDU vuoto, con campo length posto a 0, usato per il controllo.
  - 10b: PDU di tipo LL Data, può essere un messaggio L2CAP completo o il suo inizio.
  - 11b: PDU di tipo LL Control.
- NESN (1 bit): Next Expected Sequence Number, usato come acknowledge, ovvero per indicare che l'ultimo pacchetto ricevuto è stato ricevuto correttamente.
- SN (1 bit): Sequence Number, usato per conoscere se il pacchetto che si sta inviando è un nuovo pacchetto o la ritrasmissione di uno precedente.
- MD (1 bit): More Data, indica se il pacchetto è la continuazione di uno precedente o contiene nuove informazioni.
- Length (5 bit): come per l'header di un pacchetto di Advertise, indica la lunghezza del payload ed eventualmente quella del campo MIC se presente; il payload ha una lunghezza massima di 27 Byte, il MIC è fisso a 4 Byte, quindi il campo length varia da 0 a 31 Byte.

3.7 Stato di connessione

Il Link Layer entra in uno stato di connessione quando l'initiator invia un pacchetto di tipo CONNECT_ REQ (Capitolo 3.5.4); dopo aver ricevuto il pacchetto la connessione viene considerata creata ma non stabilita. Dopo che viene inviato il primo pacchetto dati la connessione si considera stabilita. Durante una connessione i due dispositivi assumono due ruoli ben distinti:

- Chi ha inviato il pacchetto di CONNECT_ REQ assumerà il ruolo di *Master* in posizione di dispositivo Centrale
- L'altro partecipante alla connessione sarà lo *Slave* ed assumerà la posizione di dispositivo periferico.

Una connessione si compone di più eventi di connessione; Master e Slave possono scambiarsi dati sono all'interno di un evento, il quale deve contenere almeno un pacchetto inviato dal Master. Lo slave deve sempre rispondere quando riceve un pacchetto dal Master, anche nel caso in cui il pacchetto abbia un CRC non valido; se il master non riceve un pacchetto in risposta dallo slave, deve chiudere la connessione. Ulteriormente la connessione può essere chiusa volontariamente sia dallo Slave che dal Master.

La temporizzazione di una connessione è scandita da due parametri fondamentali: connection event interval (*connInterval*) e slave latency (*connSlaveLatency*. L'instante dell'inizio di un evento di connessione è chiamato anchor point; in questo istante il master deve cominciare a trasmettere un pacchetto. Il connInterval deve essere un multiplo di 1,25 ms con un valore da 7,5 ms a 4 secondi. Il connSlaveLatency indica il numero di eventi di connessione che lo slave può ignorare pur mantenendo attiva la connessione; è un valore intero che va da 0 a 500 ed è calcolato come *((connSupervisionTimeout/connInterval) - 1)*

Sia Master che Slave devono avere un contatore a 16 bit (*connEventCounter*), settato a 0 al primo pacchetto inviato durante il primo evento di connessione, inviato dal master; viene incrementato ciclicamente ad ogni evento di connessione.

3.7.1 Supervision Timeout

Una connessione può terminare anche per fattori fisici riguardanti i dispositivi, come ad esempio una distanza troppo elevata, interferenze od una mancanza di potenza. Per determinare tali eventi, sia Master che Slave devono utilizzare un timer chiamato TLLConnSupervision. Se in fase di connessione questo timer raggiunge il valore di $6 \cdot connInterval$ prima che la connessione sia stabilita, essa si considera fallita. A connessione stabilita, il parametro connection supervisor timeout (*connSupervisionTimeout*) definisce il tempo massimo che deve passare tra la ricezione di due PDU di tipo data successive; Deve essere un multiplo di 10 ms con un valore compreso tra 10 ms e 32 s calcolato come $(1 + connSlaveLatency) \cdot connInterval$. Se durante una connessione, il timer raggiunge il valore di connSupervisionTimeout, la connessione deve considerarsi persa.

3.7.2 Transmit Window

In figura 22 è mostrato un evento di connessione a livello Link Layer, in cui sono visualizzati sia i pacchetti di Advertise con la relativa CONNECT_ REQ, sia i primi pacchetti di dati scambiati. Il dispositivo Master può settare l'Anchor Point a suo piacimento, per gestire eventuali altre connessioni a cui partecipa; l'Anchor Point viene settato quando il Master invia il primo pacchetto dati nella transmitWindow. Essa inizia a 1,25 ms + *transmitWindowOffset* dopo aver ricevuto il PDU della CONNECT_ REQ; il primo pacchetto non deve essere inviato più tardi di 1.25 ms + transmitWindowOffset + transmitWindowSize.

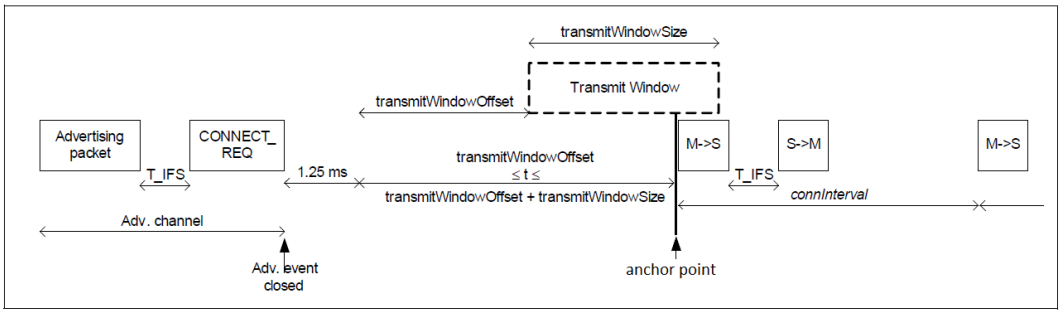


Figura 22: Schema di temporizzazione di un evento di connessione.

Il parametro *transmitWindowSize* è un multiplo di 1,25 ms e varia da 1,25 ms a meno di 10 ms e meno di (connInterval - 1.25 ms). Il *transmitWindowOffset* è sempre un multiplo di 1,25 e va da 0 a connInterval. L'intervallo di tempo tra due pacchetti consecutivi sullo stesso canale, come ad esempio la richiesta del Master e la risposta dello Slave, è chiamato Inter Frame Space; è definito come il tempo trascorso dalla fine dell'invio dell'ultimo bit del primo pacchetto all'inizio dell'invio del primo bit del secondo pacchetto. Abbreviato con l'etichetta T_ IFS ha un valore di 150 μs

3.8 Bit Ordering

L'ordine dei bit nei campi dei vari pacchetti seguono le specifiche del formato Little Endian, ovvero vengono scritti prima i Byte meno significativi ed in ordine fino ai più significativi. Si applicano inoltre le seguenti regole:

- Il bit meno significativo viene indicato con b0
- Il bit meno significativo è il primo a essere inviato nell'etere

- Nelle varie illustrazioni, l'LSB è mostrato sulla sinistra

Eventuali valori binari in ordine classico, quindi con il bit più significativo a sinistra, saranno indicati con l'appendice 'b', come ad esempio $01010101b$ che è un possibile valore per il preambolo.

Campi composti da più di un Byte, ad eccezione dei campi CRC e MIC, devono essere trasmessi con il Byte meno significativo prima ed ogni Byte con il proprio bit meno significativo iniziale. Per il campo CRC e MIC sono scritti con il byte più significativo a sinistra.

3.9 Device Address

I dispositivi sono unicamente identificati dal Device Address; è un indirizzo composto da 6 Byte e può essere pubblico o casuale. L'indirizzo di tipo pubblico, deve essere creato in accordo con lo standard IEEE 802-2001; un indirizzo di questo tipo è così composto:

- company assigned: campo assegnato dal costruttore dell'hardware, compone i 24 bit meno significativi dell'indirizzo.
- company id: identifica univocamente il costruttore dell'hardware, compone i 24 bit più significativi dell'indirizzo.

L'indirizzo di tipo casuale si divide nei 2 campi:

- Hash: occupa i 24 bit meno significativi, è ottenuto come funziona Hash a partire dai 3 Byte più significativi.
- Random: generato casualmente, compone i 24 bit più significativi dell'indirizzo.

4 RedBear Nano2

Il dispositivo usato per implementare la funzione di sniffer è il Nano2 della società RedBear. Non più grande di una moneta (10mm x 18mm), ha un costo che si aggira attorno ai 30$.

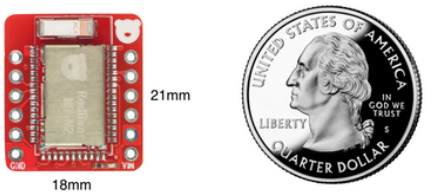


Figura 23: RedBear Nano2, comparato con una moneta da 1/4 di dollaro.

Il nome in codice del modello è 'MB-N2'ed integra al suo interno il chip della Nordic nRF52832, un'antenna integrata ed un totale di 32 pin di I/O[15] che offrono i servizi di UART, SPI, ADC e PWM.

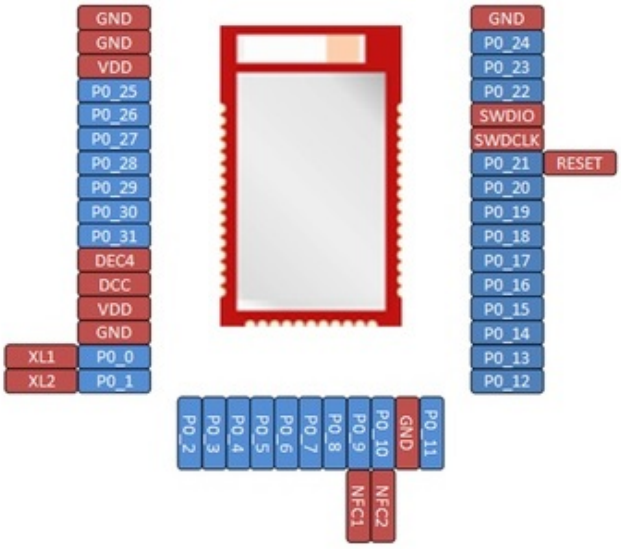


Figura 24: Modulo SoC Nordic nRF52832 integrato nel RedBear Nano2.

[15]Input, Output: utilizzabili a piacimento come periferiche di ingresso o uscita

Il chip della Nordic Semiconductor nRF52832 Soc ha le seguenti caratteristiche

- Processore ARM Cortex-M4F a 32 bit e 64 MHz.
- Bluetooth 4.2 certificato e compatibile con le specifiche 5.0 .
- NFC.
- 64 KB di ram.
- 512 KB di memoria Flash.
- FPU, unità di calcolo in virgola mobile.
- DSP, processore di segnale digitale.

Non disponendo di un'interfaccia standard per la comunicazione con un PC, il Nano2 necessita di una board di supporto sia per essere programmato, sia per comunicare informazioni a quest'ultimo. La board che viene fornita assieme al Nano2 è chiamata DAPLink e disponibile nella versione 1.5 . Essa monta un processore ARM Cortex-M3 MCU ed è utilizzata oltre che per la programmazione, anche per il debug dei progetti su Nano2

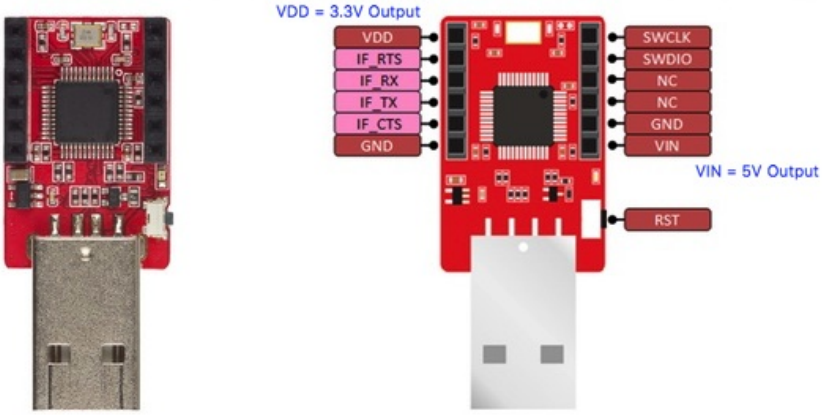


Figura 25: Board DAPLink, un progetto open source del team ARM mbed.

5 Ambiente di sviluppo

5.1 Introduzione

É stato scelto di sviluppare l'intero progetto di tesi in ambiente UNIX, usando come sistema operativo Linux, nello specifico la distribuzione di Ubuntu 17.04 a 64 bit denominata Zesty Zapus. Questa decisione è stata presa per integrare al meglio i progetti ed il codice già esistente e per creare una base di sviluppo futura che sia Open Source[16] e liberamente utilizzabile senza alcun vincolo di licenza. Non è stata comunque una decisione immediata perché il RedBear Nano2 integra un Soc[17] progettato dalla società Nordic Semiconductor, chiamato "nRF52832", la quale mette a disposizione vari Tool, funzionalità e IDE di sviluppo per questo chip ideati e creati per funzionare unicamente in ambiente Windows; fortunatamente esistono vari progetti e guide abbastanza dettagliate reperibili online, alcune delle quali supportate direttamente dalla Nordic stessa, che aiutano a settare ed integrare il materiale già esistente anche in ambiente UNIX. Rimangono comunque delle forti limitazioni nello sviluppo degli applicativi per Nano2 su Linux, primo fra tutti la mancanza della possibilità di un ambiente di Debug che permetta l'analisi istruzione per istruzione del codice in esecuzione sul dispositivo, con la relativa possibilità di visualizzazione dei valori delle varie variabili durante l'esecuzione; questa limitazione è stata in parte sopperita utilizzando l'interfaccia UART, utilizzandola per inviare stringhe contenenti informazioni su quali parti del codice fossero o meno state realmente eseguite ed il valore di alcune variabili di interesse nelle varie fasi dell'esecuzione dell'applicativo sviluppato. Questa incompatibilità ha rallentato il processo di sviluppo su Linux ma non lo ha reso impossibile, ne tanto meno ne ha limitato le potenzialità.

5.2 Installazione GNU toolchain

Prima di iniziare a scrivere codice, è necessario installare dei componenti software che permettano di compilare in linguaggio macchina ARM il codice da noi prodotto. Quello che ci serve è un compilatore chiamato *GNU toolchain for ARM Cortex-M* scaricabile gratuitamente dal sito ufficiale di 'arm.developer'. [5] Una volta scaricato ed estratto il pacchetto, bisognerà aggiungere al PATH di sistema la seguente directory:

[16] Qualità di un sistema che consiste nell'essere di libero uso e riutilizzabile senza alcun costo, per ampliarne funzionalità e caratteristiche

[17] Sistem on a Chip, è un particolare circuito integrato che in un solo chip contiene un sistema completo

```
<directory estratta>/gcc-arm-none-eabi-6-2017-q2-update/bin
```

Per compiere questa operazione in Ubuntu, basterà aprire una finestra del terminare BASH ed eseguire il comando:

```
echo 'export PATH=\$PATH:<directory
↪  estratta>/gcc-arm-none-eabi-6-2017-q2-update/bin' >>
↪  ~/.profile
```

ovvero aggiungere al file .profile il percorso al compilatore, il quale sarà aggiunto poi in automatico alla variabile di sistema PATH; questo permetterà di invocare i comandi di compilazione da qualunque punto del sistema, senza dover richiamare il percorso al compilatore. É possibile verificare se il procedimento è stato eseguito con successo eseguendo il seguente comando al terminale:

```
arm-none-eabi-gcc --version
```

che deve restituire la versione del compilatore installata.

Un altro componente essenziale da aver installato è il compilatore *GNU make*, che normalmente è già presente sulla maggior parte delle distribuzioni UNIX. Se così non fosse, per quanto riguarda Ubuntu, si installa semplicemente eseguendo il comando:

```
sudo apt-get install build-essential checkinstall
```

5.3 Configurazione SDK

L'SDK[18], ovvero l'insieme di tutte le funzioni di libreria necessarie per creare applicazioni da eseguire sul Nano2, è liberamente scaricabile dal sito della Nordic Semiconductor [6] ; La versione utilizzata per questo progetto è la 14.1.0 che è creata per essere utilizzata sul Chip nRF52832. La struttura di questa SDK è molto rigida, tutti i progetti contenuti al suo interno fanno riferimento ai vari file contenuti nell'SDK con un percorso assoluto; è meglio quindi non modificarne la struttura per non avere problemi di compilazione in futuro. Scaricata ed estratta in una directory a piacimento, va impostato il percorso alla toolchain ARM, installata nel capitolo 5.2, andandolo a impostare nel file makefile.posix che si trova nella cartella:

```
<SDK>/components/toolchain/gcc
```

[18]Software Development Kit

dove <SDK>è la root dell'SDK estratto.

Aprendo il file con un semplice editor di testo, cambiare i 3 campi riportati come segue:

```
GNU_INSTALL_ROOT := .../gcc-arm-none-eabi-6-2017-q2-update/bin/
GNU_VERSION := 6.3.1
GNU_PREFIX := arm-none-eabi
```

Nella SDK si trovano anche molti progetti di esempio, da cui è essenziale partire per sviluppare un nuovo programma; ogni progetto di esempio contiene svariati file sorgente da compilare e linkare. Per rendere semplice la compilazione è fornito un Makefile, ovvero un file che contiene tutte le istruzioni per la compilazione del progetto e che integra regole di dipendenza e regole di interpretazione, utilizzate per conoscere quali sono i file utilizzati nel progetto, con il relativo percorso, garantendo una compilazione esente da errori e solo di quei file sorgente che sono stati modificati dall'ultima compilazione.

Da questo punto è possibile compilare un progetto di esempio direttamente da terminale, per verificare che i passi compiuti fin'ora siano stati eseguiti correttamente; portandosi nella cartella del progetto 'blinky'e successivamente nella sotto cartella *armgcc*; qui invocando il comando *make* si avvierà la compilazione che se avrà successo creerà una sotto cartella chiamata _build contenete il file nrf52832_xxaa.hex, che è il nostro programma compilato in linguaggio macchina ARM che dovrà essere poi scritto sulla memoria del Nano2 per essere eseguito. Per scrivere il file sulla memoria del Nano2 basterà copiarlo all'interno della periferica usb che viene rilevata dal pc quando lo si connette tramite la board DAPLink v1.5 .

5.3.1 Configurazione piedinatura Nano2

Il dispositivo della società RedBear chiamato Nano2 che viene utilizzato in questa tesi, non è uno dei dispositivi ufficialmente supportati dalla SDK; Bisogna quindi prendere una delle varie board che vengono ufficialmente supportate e cambiarne la piedinatura, ovvero l'associazione tra una specifica periferica, come il led, o una specifica funzionalità, ad esempio UART, e il relativo pin sul chip. É stato scelto di prendere la board di sviluppo PCA10040 ed il relativo file di configurazione pca10040 e adattarne la piedinatura con quella dichiarata dalla Redbear, mostrata in figura 26

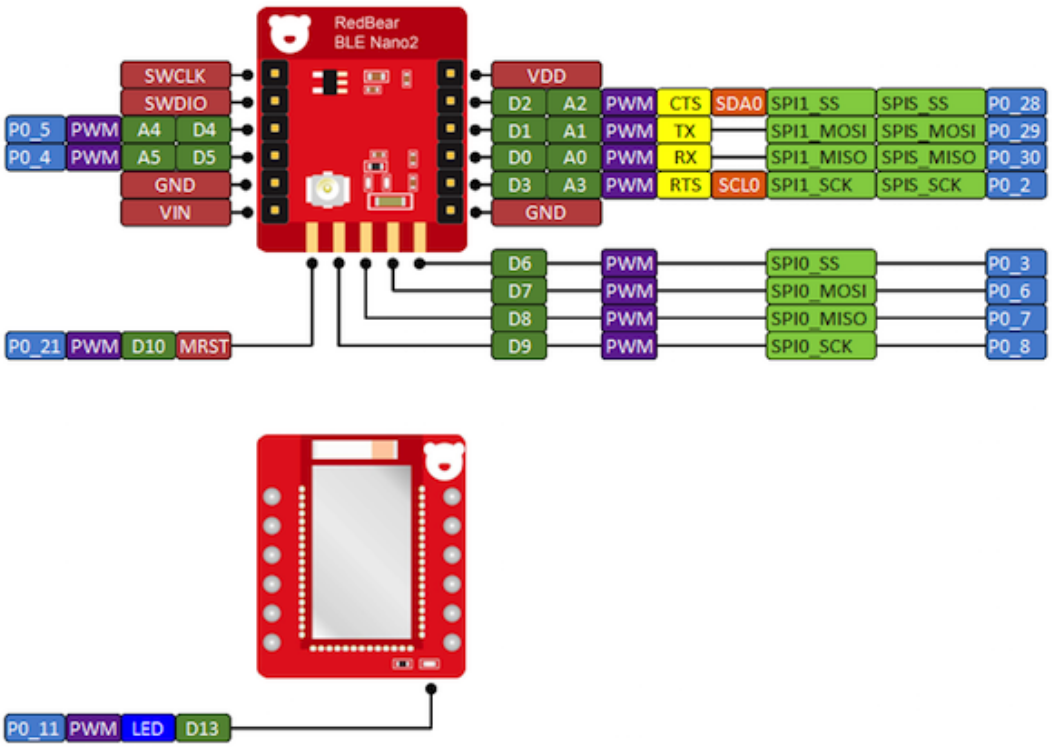


Figura 26: Piedinatura del modulo Nano2

ad esempio, qui si ha a disposizione solo 1 led, connesso al pin 11, mentre sulla board PCA10040 se ne avevano 4. Bisognerà cambiare quindi il numero totale di led da 4 a 1 ed il pin dell'unico led con il valore 11. Dopo aver modificato correttamente questo file, tutte le modifiche saranno automaticamente aggiornate per tutti gli esempi contenuti nell'SDK, andando a selezionare per ogni progetto il makefile contenuto nella cartella che riporta il nome della board PCA10040. Per l'esempio del lampeggio del led, il percorso per raggiungere il makefile corretto sarà quindi:

`<SDK>/examples/peripheral/PCA10040/blank/armgcc/`

5.3.2 Softdevice

Progetti più complessi di un semplice lampeggio di un led, come tutti i progetti che permettono lavorare con il BLE, necessitano che assieme al codice del programma venga scritto sul dispositivo dell'altro codice preconfezionato; questo ulteriore codice, chiamato *Softdevice* non è altro che un insieme di funzioni di libreria che vengono fornite già compilate e che sono utilizzabili

dal nostro codice per svolgere le operazioni più complesse utilizzando poche righe di codice. Esistono varie versioni del softdevice, diverse per funzionalità e compatibilità con i chip; per tutti le applicazioni sviluppate è stata usata la versione S132. Se si sta sviluppando un progetto che utilizza tali funzioni, prima di andare a scrivere il file .hex sul Nano2 bisognerà unirlo con il softdevice, che viene fornito nella SDK (¡SDK¿/components/softdevice/s132/hex), utilizzando una utility scaricabile dal sito della Nordic, chiamata 'mergehex', e disponibile per tutti i Sistemi Operativi. Per velocizzare il processo di unione dei due file .hex e successiva scrittura sul Nano2 è stato creato uno script di bash che automatizza tutto il processo:

```
#!/bin/bash

SOFTDEVICE=/home/utente/SoftDevices/s132_nrf52_5.0.0/s132_nrf52_5.0.0.hex
SORGENTE=nrf52832_xxaa.hex
OUTPUT=toflash.hex
USB_PROG=/media/utente/DAPLINK

mergehex -m ${SORGENTE} ${SOFTDEVICE} -o ${OUTPUT}

echo "Flashing to ${USB_PROG} ..."
cp ./${OUTPUT} ${USB_PROG}/{OUTPUT}
```

5.4 Installazione IDE

Anche se non strettamente necessario ai fini dello sviluppo, un ambiente grafico per la scrittura del codice (IDE) è consigliato, dato il gran numero di righe di codice che può contenere un applicativo. Eclipse, l'IDE di sviluppo utilizzato in questa tesi, è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma sviluppato in Java dalla *Eclipse Foundation*, distribuito liberamente e gratuitamente. Oltre all'ide di base, nella versione per sviluppatori C,C++ vanno aggiunti vari pacchetti per permettere lo sviluppo sui dispositivi integrati con processore ARM; per facilitare le operazioni di aggiunta e settaggio di questi pacchetti esistono delle versioni di Eclipse modificate che integrano tutti questi pacchetti; queste versioni modificate è scaricabile gratuitamente dal repository GitHub *gnu-mcu-eclipse* [7]. Scaricato ed estratto in una directory a piacimento sarà già pronto per essere utilizzato per sviluppare applicazioni per il Nano2.

5.4.1 Importare un esempio in Eclipse

Per inziare a sviluppare o testare uno dei vari esempi presenti nell'SDK utilizzando Eclipse, bisogna prima importarlo nell'ambiente di lavoro. Per fare ciò bisogna creare un nuovo progetto in Eclipse che si basi su un Makefile esistente.

`File -> Makefile Project with Existing Code`

andando poi a impostare come percorso la cartella che contiene il makefile del progetto; come toolchain bisognerà settare 'ARM Cross GCC'.

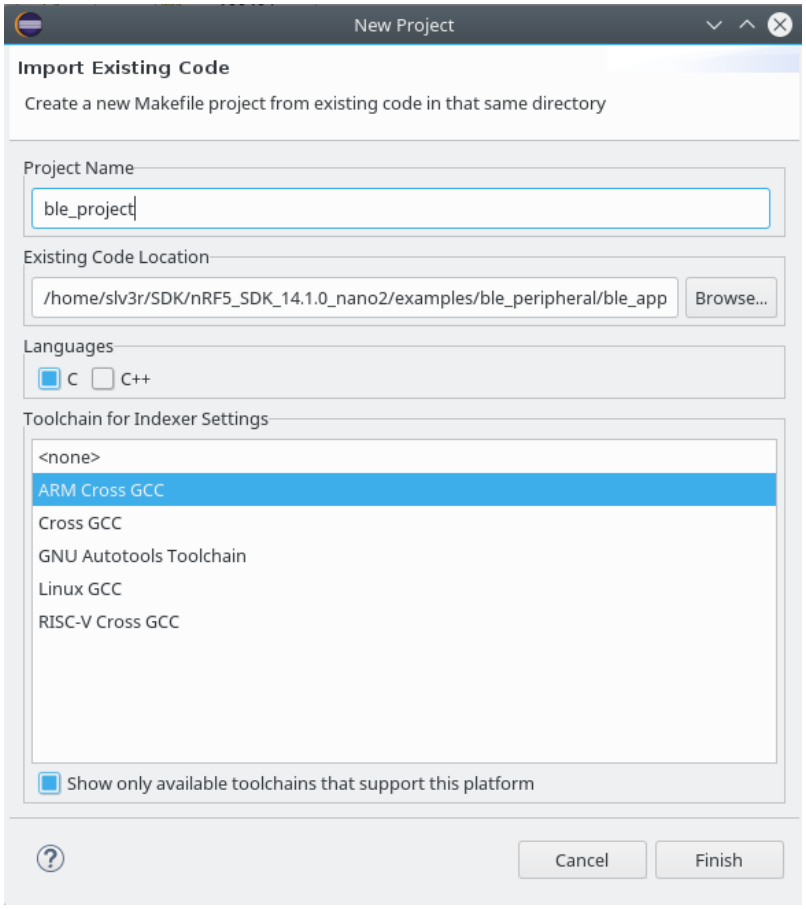


Figura 27: Creazione di un nuovo progetto da Makefile

Nella schermata dei progetti di eclipse apparirà il progetto appena creato, vedremo però solo il file Makefile; si deve importare manualmente il file main.c andando prima a creare una nuova cartella virtuale, tramite il percorso New->Folder e nella schermata selezionare Advanced e successivamente

Virtual Folder. Poi tramite tasto destro sulla cartella->import andare a selezionare il file main.c che è situato nella cartella principale del progetto.

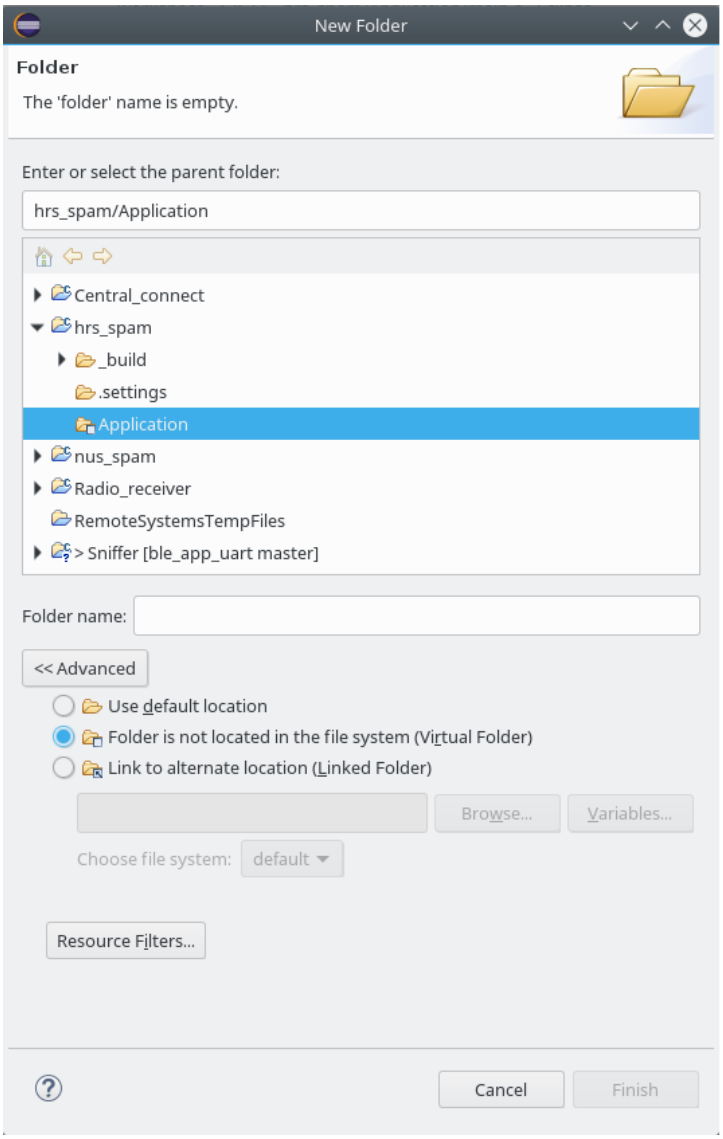


Figura 28: Creazione di una cartella virtuale nel progetto, che conterrà il file main.c

L'ultima operazione da compiere prima di poter compilare tramite Eclipse è settare il comando di compilazione; tramite tasto destro sulla cartella del progetto, visualizzata a lato sinistro, e selezionando Properties. Si aprirà una nuova schermata in cui si deve selezionare C/C++ Build ed impostare il campo Build Command come segue:

```
make VERBOSE=1
```

Questo, oltre ad invocare il comando corretto di compilazione, crea un output di tipo 'verbose'dal quale Eclipse ottiene le informazioni sui percorsi dei vari file da cui dipende il main, andando quindi ad eliminare tutti gli errori sintattici che erroneamente l'IDE evidenzia.

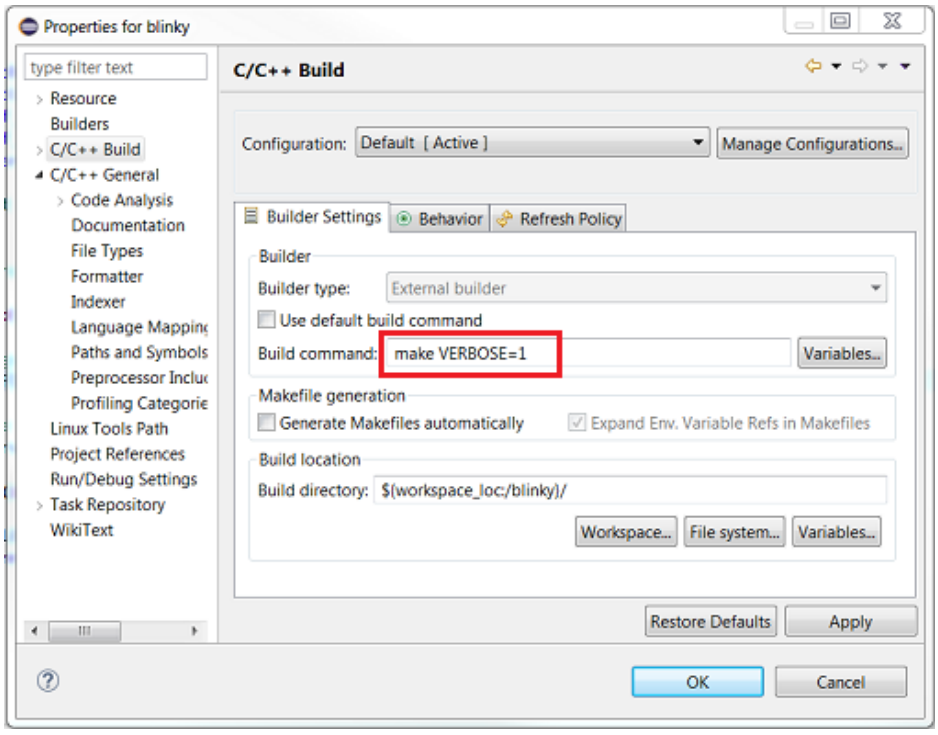


Figura 29: Impostazione del comando di compilazione su Eclipse

6 Progettazione sniffer

Il primo passo per lo sviluppo è stato prendere confidenza sia con l'ambiente di lavoro che con il dispositivo della RedBear. Testare un semplice progetto, come il lampeggio di un led, si è comunque rivelata un'operazione non banale. L'SDK che la Nordic fornisce contiene esempi modificati appositamente per le sue board di sviluppo, come la PCA10040 o la PCA10056, descritte nel capitolo 2.3, che come già accennato hanno una piedinatura e una dotazione di periferiche differente rispetto al nano2. Il primo passo è stato quindi quello di utilizzare una board supportata e modificarne la piedinatura delle periferiche connesse; il numero di led è stato ridotto a 1 solo connesso al pin 11 e sono stati rimossi i riferimenti a pulsanti connessi, perché non presenti sulla nostra periferica. É stato necessario modificare anche i collegamenti dei 4 pin relativi alla comunicazione UART come segue:

- CTS pin 28
- RTS pin 2
- TX pin 29
- RX pin 30

Tutte queste modifiche sono state apportate al file pca10040.h che si trova nell'sdk nella cartella /components/boards .

Eseguendo una nuova compilazione e provando a scrivere il file .hex generato continuava a non funzionare. Dopo varie prove si è capito che mancavano delle librerie da cui il progetto dipendeva; esse sono contenute nel SOFTDEVICE, un file .hex che viene fornito già compilato nell'SDK, nella cartella /components/softdevice/s132/hex. Per tutti i progetti creati si usa la versione s132 del softdevice che è la più completa e contiene tutte le librerie necessarie per far funzionare il nano2 sia come dispositivo di tipo central che peripheral.

6.1 UART

Per visualizzare i dati intercettati via etere è necessario poter comunicare informazioni ad un dispositivo dotato di output video; nel nostro caso è stato utilizzato un PC, che avendo una gran capacità di memorizzazione può contenere tutti i dati catturati, per essere visualizzati ed analizzati anche in un secondo momento. Per trasferire questi dati abbiamo utilizzato una funzionalità supportata dal nano2, la trasmissione usando UART. É l'acronimo

di Universal Asynchronous Receiver Transmitter, ovvero ricevitore trasmettitore asincrono/seriale, è un dispositivo hardware che converte flussi da un formato parallelo, tipicamente quelli usati all'interno di un processore, in formato seriale asincrono. Inviare dati tramire UART è un'operazione che richiede pochi settaggi e quindi poche righe di codice, utilizzando le apposite funzioni di libreria messe a disposizione nella SDK.

All'interno di un progetto, va prima inizializzato il modulo fornendo le impostazioni che desideriamo usare:

- Baudrate: il numero di simboli che vengono trasmessi in un secondo, determina la velocità di trasmissione.

- Control Flow: indica se utilizzare o meno le due linee aggiuntive destinate alla comunicazione UART ovvero, CTS (Clear To Send) e RTS (Ready To Send) per gestire lo scambio di dati, utile per annullare i conflitti trasmissivi.

Normalmente i settaggi sono inseriti nella funzione *uart_init()* che viene eseguita all'avvio del dispositivo.

Vanno inseriti dei caratteri di controllo per differenziare la fine di un pacchetto con l'inizio del successivo, ed eventualmente indicare informazioni aggiuntive al solo pacchetto, come la lunghezza dello stesso. É quindi stato inserito il carattere denominato MARKER del valore di 0xE0 all'inizio di ogni dato inserito; i 2 Byte successivi indicano la lunghezza dell'intero pacchetto, quindi comprendono anche i caratteri di controllo. Il quarto Byte viene usato per indicare il canale il cui è stato catturato il pacchetto e dal 5° Byte in poi si avrà ciò che è stato catturato, così come viene rilevato dal Nano2.

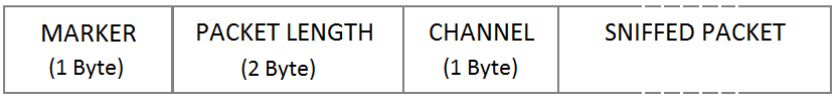


Figura 30: Pacchetto inviato tramite UART che contiene il pacchetto sniffato e i campi di controllo.

Nel caso in cui all'interno del pacchetto stesso ci siano Byte del valore di 0xE0, che hanno lo stesso valore del Marker, per evitare di confonderli si raddoppia il Byte. In questo modo se nel pacchetto ricevuto si incontrerà un Byte singolo del valore del Marker, allora quello sarà un vero Marker che indica l'inizio di una nuova trasmissione; negli altri casi sarà un Byte del pacchetto, si dovrà quindi scartare il Byte successivo, che non fa parte dei dati catturati dallo Sniffer.

6.2 NRF RADIO

Il cuore del progetto, ovvero l'intercettazione dei pacchetti, viene creato utilizzando la struttura NRF_ RADIO, che contiene tutti i possibili campi settabili per far funzionare il dispositivo Radio, contenuto all'interno del SoC nRF52832, nelle modalità che si desiderano.

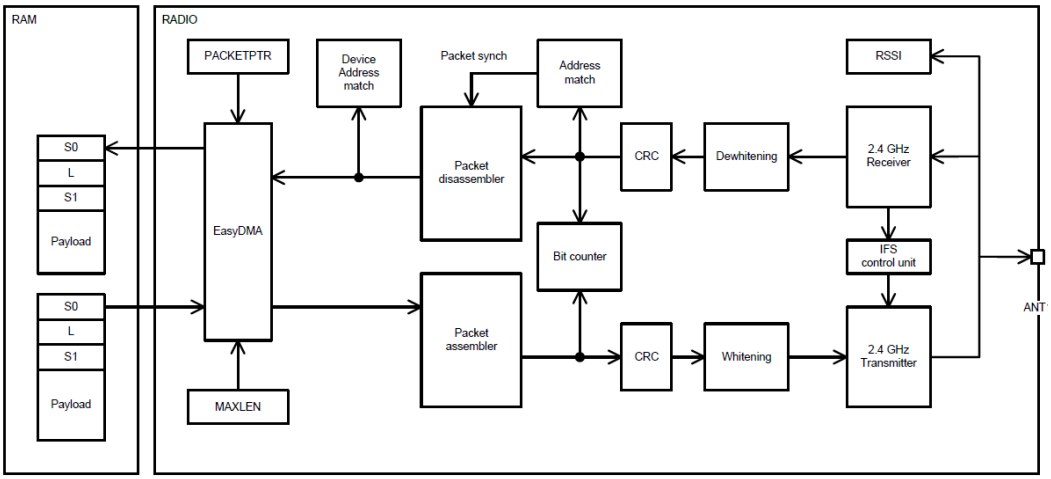


Figura 31: Diagramma a Blocchi del trasmettitore radio.

Il dispositivo Radio contiene un ricevitore e trasmettitore a 2.4 GHz, un assemblatore e disassemblatore di pacchetti automatico, un generatore e verificatore di CRC automatico che permettono in modo facile e veloce di ottenere il pacchetto ascoltato partendo da ciò che è stato ricevuto via radio. Include inoltre un Device Address Match, utile per quelle funzionalità che richiedono il filtraggio dei pacchetti in automatico, per ricevere solo quelli destinati o provenienti da un determinato indirizzo. Si è dimostrato molto utile anche il sistema RSSI[19], ovvero un sistema che calcola l'intensità del segnale ricevuto, per essere eventualmente filtrato se troppo lontano o debole. In ogni ambiente in cui si è andato ad attivare lo sniffer erano sempre presenti nell'etere pacchetti di altri dispositivi che andavano a sommarsi ai dispositivi usati per il test, rendendo più difficile l'individuazione a video dei pacchetti di interesse; tramite la ricezione della potenza del segnale è stato possibile escludere dall'invio e quindi dalla visualizzazione tutti quei pacchetti dei dispositivi lontani dallo sniffer, quindi di non interesse, garantendo la visualizzazione dei soli pacchetti provenienti dai dispositivi sotto test. Il sistema Radio integrato nel SoC della Nordic è pensato per essere utilizzato

[19]Received Signal Strength Indicator

con più protocolli trasmissivi, oltre al Bluetooth Low Energy; vanno quindi indicati, in fase di configurazione la lunghezza in Byte delle varie parti di un pacchetto BLE, con particolare specifica del campo Length del pacchetto. I campi di cui andare a settare la grandezza sono SO, LENGTH ed S1. Tramite il valore di PCNF0 della struttura NRF_ RADIO è possibile configurare la lunghezza dei campi suddetti.

```
NRF_RADIO->PCNF0 = (
        (((1UL) <<RADIO_PCNF0_S0LEN_Pos)& RADIO_PCNF0_S0LEN_Msk) |
        (((2UL) <<RADIO_PCNF0_S1LEN_Pos)& RADIO_PCNF0_S1LEN_Msk) |
        (((6UL) <<RADIO_PCNF0_LFLEN_Pos)& RADIO_PCNF0_LFLEN_Msk)
);
```

Dove la lunghezza di S0 è espressa in Byte e contiene il primo Byte dell'header, dove sono contenute le informazioni sul tipo di pacchetto e sul tipo degli eventuali indirizzi contenuti nel Payload (pubblici o privati. La lunghezza di S1 è espressa invece in bit; nel nostro caso è uguale a 2 perché per i pacchetti di tipo advertise nel secondo Byte dell'header 2 bit sono RFU e attribuiti quindi a questo campo: La lunghezza del campo length è espressa sempre in bit, 6 nel nostro caso e contiene appunto la lunghezza in Byte del payload.

Il valore del campo PCNF1 configura altri parametri, come la lunghezza massima che può assumere il Payload, eventuali Byte di espansione di quest'ultimo, lunghezza dell'indirizzo, il tipo di endian da utilizzare (introdotto nel capitolo 3.8)

```
NRF_RADIO->PCNF1 = (
        (((37UL) << RADIO_PCNF1_MAXLEN_Pos) & RADIO_PCNF1_MAXLEN_Msk)  |
        (((0UL) << RADIO_PCNF1_STATLEN_Pos) & RADIO_PCNF1_STATLEN_Msk) |
        (((3UL) << RADIO_PCNF1_BALEN_Pos) & RADIO_PCNF1_BALEN_Msk)     |
        (((RADIO_PCNF1_ENDIAN_Little) <<
        ↪  RADIO_PCNF1_ENDIAN_Pos&RADIO_PCNF1_ENDIAN_Msk) |
        (((1UL) << RADIO_PCNF1_WHITEEN_Pos) & RADIO_PCNF1_WHITEEN_Msk)
);
```

Da configurare all'interno della struttura NRF_ RADIO è anche il canale in cui essere in ascolto; esso influenza 2 campi:

```
NRF_RADIO->FREQUENCY = frequency_resolver();
NRF_RADIO->DATAWHITEIV = channel;
```

FREQUENCY è il campo in cui scrivere la frequenza di ascolto, indicata a scarti di 2 MHz e con prima frequenza utile il valore 0, ha quindi un campo di escursione da 0 a 80 MHz. Viene calcolata tramite la funzione

frequency_resolver() che si basa sull'indice del canale desiderato. DTAWHITEIV va settato semplicemente con l'indice del canale e serve per inizializzare l'algoritmo che andrà a rimuovere il whitening dai pacchetti ricevuti.

Tramite l'Access Address e il CRCInit che vengono forniti alla Radio si riceveranno solo i pacchetti che corrispondono a questi valori. Per i pacchetti di Advertising l'Access Address è fisso e vale 0x8E89BED6. Esso all'interno della struttura Radio è diviso in 2 parti: il PREFIX0 e BASE0; per impostare la Radio per la ricezione dei pacchetti advertise, l'indirizzo va diviso come segue:

```
NRF_RADIO->PREFIX0 = 0x8e;
NRF_RADIO->BASE0 = 0x89bed600;
```

Per i pacchetti di una connessione l'AA varia, ed è deciso dal device Central al momento della connessione ed inviato al device Peripheral nella CONNECT_REQ.

I parametri da impostare per quanto riguarda la gestione del CRC sono il CRCCNF che indica il numero di Byte di lunghezza del CRC, il CRCINIT che per i pacchetti di Advertise ha un valore fisso a 0x555555 ma che per una connessione varia in accordo con il valore contenuto nella CONNECT_REQ, ed il CRCPOLY, che per tutte le trasmissioni BLE è fisso al valore 0x65B.

```
NRF_RADIO->CRCCNF  = (RADIO_CRCCNF_LEN_Three << RADIO_CRCCNF_LEN_Pos) |
           (RADIO_CRCCNF_SKIPADDR_Skip << RADIO_CRCCNF_SKIPADDR_Pos);
NRF_RADIO->CRCINIT = 0x555555;
NRF_RADIO->CRCPOLY = 0x00065B;
```

Dopo aver correttamente impostato la Radio è possibile mettere in ascolto il dispositivo per la ricezione dei pacchetti. la ricezione viene descritta ed attuata tramite una macchina a stati; il dispositivo Radio viene impostato e assume determinati stati, e controllandoli è possibile conoscere quando è stato ricevuto il pacchetto. Il diagramma degli stati per la ricezione e l'invio di pacchetti è mostrato in figura 32.

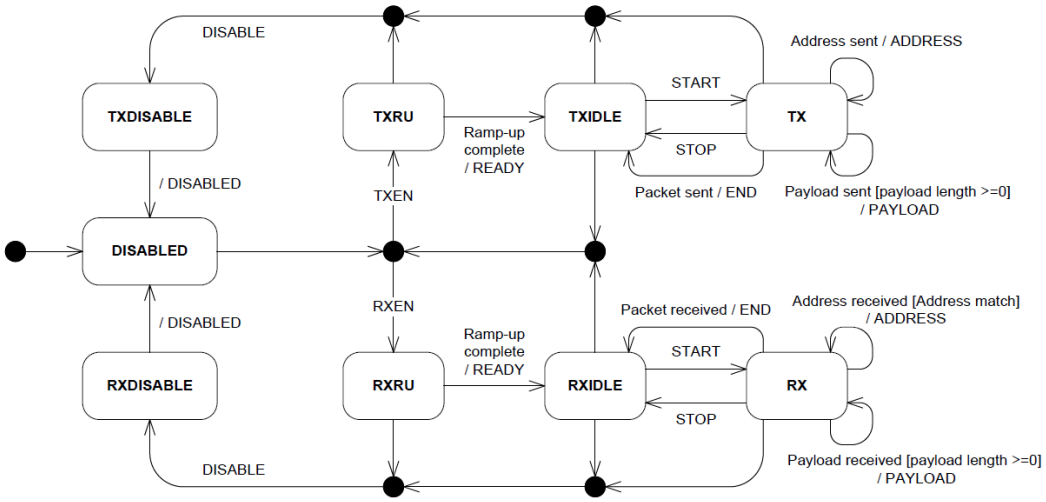


Figura 32: Diagramma degli stati per il dispositivo Radio che comprende le fasi di Invio e Ricezione.

Avviando la task per la ricezione, tramite il comando

```
NRF_RADIO-> TASKS_RXEN = 1U;
```

si imposta la Radio nello stato RXRU. Successivamente controllando quando il valore del campo EVENTS_ READY assume il valore 1, si può sapere se il dispositivo Radio è pronto per la ricezione. Avviando poi il task di Start, la radio andrà nello stato RX. Per conoscere quando un pacchetto è stato completamente ricevuto si deve controllare il valore del campo EVENTS_ END, se è 1 un pacchetto è stato ricevuto. Il controllo della correttezza del CRC non avviene in automatico ma va verificato controllando il valore del campo CRCSTATUS, se 1 il CRC calcolato corrisponde a quello inviato con il pacchetto e quindi il pacchetto è stato ricevuto senza errori. Ad ogni ricezione la Radio va disabilitata, per poter tornare allo stato iniziale e ricevere il pacchetto successivo.

Ogni pacchetto ricevuto deve essere inviato alla UART; vanno prima aggiunti i caratteri di controllo e il separatore (Marker) dei pacchetti, ed è necessario controllare se il pacchetto catturato contenga il valore 0xE0, in quel caso va aggiunto un ulteriore Byte al pacchetto in successione a quest'ultimo per evitare che venga riconosciuto erroneamente come un Marker. Va quindi calcolata la lunghezza totale del pacchetto così modificato e inserita nei Byte 1 e 2 cosicché il ricevitore, dall'altro lato della comunicazione seriale, sappia quanto è lungo il pacchetto e riesca ad identificare eventuali errori di trasmissione se si vede arrivare un Marker prima che il pacchetto

inviato abbia raggiunto la lunghezza dichiarata. Sul dispositivo Nano2 esiste una funzione di libreria che permette di inviare un singolo Byte alla UART e conoscerne il momento dell'effettivo invio, così da evitare di terminare l'invio precedente e sovrascriverlo con uno nuovo; la funzione va quindi chiamata con un while che ne attende il completamento dell'invio:

```
while(NRF_SUCCESS != app_uart_put(ch));
```

6.3 Ricevitore seriale

Dall'altro lato della comunicazione UART è stato usato un PC con sistema operativo Linux, nello specifico Ubuntu 17.04, che ha il vantaggio di supportare nativamente le trasmissioni in seriale andando a creare uno speciale file, /dev/ttyACM0 , in cui si trovano scritti tutti i dati ricevuti da seriale. Leggendo il file è si ottengono i pacchetti inviati dal Nano2 che vanno poi elaborati, visualizzati a video e salvati su un file per analisi future. É stato creato un programma, scritto in linguaggio C, che permette di svolgere le funzioni sopra citate; essendo la trasmissione asincrona, esso deve innanzitutto aspettare la ricezione di un Marker singolo, che indica l'inizio di un nuovo pacchetto, andare a leggere i 2 Byte successivi e ottenerne la lunghezza, e mediante un ciclo for, andare a leggere tutti i Byte successivi e presentarli a video, salvandoli anche sempre su un file secondario. Per svolgere queste funzioni è stata progettata una macchina a stati, replicata poi in linguaggio C mediante il costrutto *Switch(state)*, mostrata in figura 33.

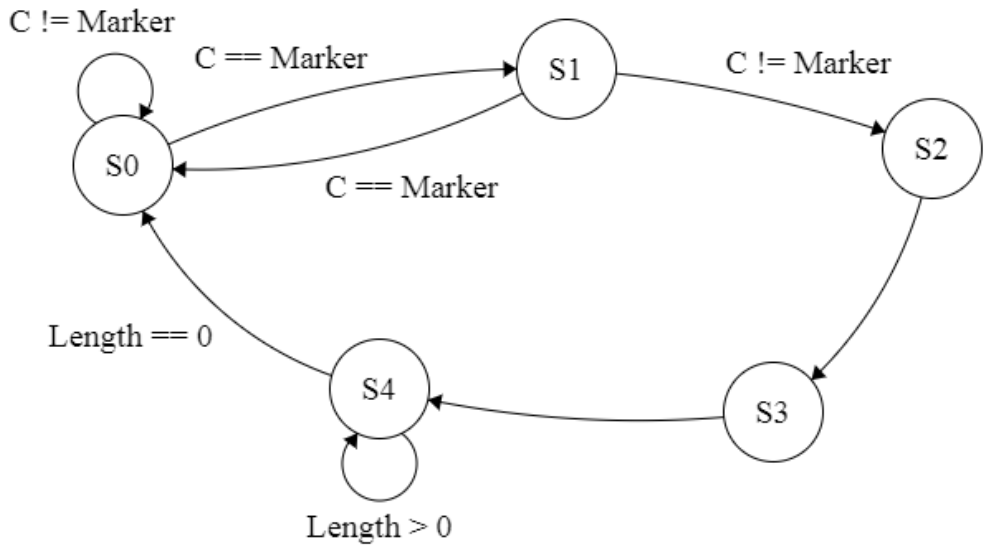


Figura 33: Diagramma della macchina a stati implementata in C per ricevere i dati dalla UART.

Lo stato 0 analizza tutti i Byte scritti nel file alla ricerca del Marker singolo, se lo trova avanza nello stato 1. Qui se il Byte successivo è anch'esso un Marker, siamo di fronte ad un falso Marker e si deve tornare nello stato 0 ad aspettare un nuovo Maker, se invece il Byte è diverso allora esso è il byte più significativo della lunghezza, che va letto e salvato in una variabile; si passa automaticamente nello stato 2 dove si legge il restante Byte della lunghezza avendo quindi ad ora la lunghezza del pacchetto da stampare. Nello stato 3 si va a leggere il 4° Byte del pacchetto che indica in che canale è stato catturato e si va a scrivere subito l'informazione; si procede direttamente nello stato 4 dove si andranno a leggere e stampare tutti i restanti Byte del pacchetto, andando di volta in volta a decrementare la variabile length finché non assume il valore 0 ed il pacchetto è stato interamente letto e si torna quindi nello stato iniziale 0.

6.3.1 Seguire una connessione

Per catturare i pacchetti di tipo Advertise è sufficiente sintonizzarsi su un canale di Advertise (37, 38 o 39) e tramite l'AA e CRCInit noti per tutti i pacchetti di Advertise si ricevono tutte le comunicazioni dei dispositivi che vogliono farsi individuare nell'area adiacente allo Sniffer. Un qualsiasi dispo-

sitivo che invia pacchetti di Advertise lo fa sempre sui i 3 canali dedicati; uno sniffer quindi è sufficiente che si metta in ascolto su uno dei 3 per poter catturare tutti i pacchetti di questo tipo. Il procedimento si fa più complicato quando si deve andare ad intercettare il pacchetto di CONNECT_ REQ, perchè viene inviato soltanto su 1 dei 3 canali, ed inviato una sola volta. Per ovviare a questo problema ci sono varie soluzioni; la prima, la più elementare ma che si adatta a tutti i possibili scenari, è fissarsi su un canale e forzare il dispositivo target a riconnettersi più volte finché non manderà la CONNECT_ REQ sul canale in cui siamo in ascolto. Una seconda soluzione è creare un dispositivo di Advertising a piacimento che faccia Advertise solo sul canale in cui siamo in ascolto; il dispositivo che vorrà connettersi con quest'ultimo dovrà obbligatoriamente, seguendo le specifiche del protocollo BLE, inviare la sua richiesta di connessione sul canale in cui siamo in ascolto. L'ultima soluzione, la più efficacie ma anche più complessa e costosa, è avere 3 sniffer, uno per ogni canale di Advertise che forniscono una sicurezza del 100% di riuscire ad intercettare il pacchetto di CONNECT_ REQ. Per seguire successivamente la connessione bisognerà ottenere alcuni valori dal pacchetto di connessione catturato; l'AA contenuto nei Byte [15],[16],[17],[18] CRCInit nei Byte [19],[20],[21] e il valore dell'Hop Increment contenuto negli ultimi 5 bit del pacchetto. Successivamente si deve impostare la struttura NRF_ RADIO con i valori estratti dal pacchetto di connessione per catturare unicamente i pacchetti della connessione; La parte più complessa è gestire il cambio di canale con le tempistiche esatte per riuscire a ricevere tutti i pacchetti della connessione. Dalle specifiche Low Energy è noto che l'intervallo di connessione, ovvero il tempo in cui la connessione avviene entro un determinato canale, è fisso e deciso in fase di connessione. Questo intervallo inizia nello stesso istante in cui il dispositivo Central invia il primo pacchetto; l'istante viene definito come Anchor Point, ovvero punto di ancoraggio per la connessione. Non esiste però una funzione di libreria della Nordic che permette di ottenere questo istante, ci si può però basare sull'istante in cui viene ricevuto l'indirizzo, che si può conoscere tramite funzioni di libreria, e togliere il tempo che è trascorso per ottenere l'istante iniziale in cui il pacchetto è stato trasmesso, dato che l'indirizzo viene trasmesso sempre un numero fissato di Byte dopo l'inizio del pacchetto e la velocità di trasmissione è nota e costante. Dopo aver fissato l'anchor point, utilizzando la gestione del channel hopping descritta nel capitolo 3.2.2, è possibile seguire la connessione ed ottenerne tutti i pacchetti.

6.4 Problematiche riscontrate

Il progetto dello Sniffer che riuscisse anche a seguire una connessione tra due dispositivi target ha presentato non pochi scogli; la prima grossa problematica è nata sulla cattura della CONNECT_ REQ. Questo pacchetto, fondamentale per poter ascoltare una connessione, nelle prime versioni dello Sniffer non riusciva ad essere mai catturato. Indipendentemente dai canale di Advertise in cui si era in ascolto o dal numero di tentativi di connessione che di provavano o dalla potenza trasmissiva usata, il pacchetto di connessione non compariva mai nell'elenco dei pacchetti catturati. La soluzione di questo problema è stata trovata analizzando con il dispositivo USRP i tempi che intercorrevano tra i vari pacchetti di Advertise e la richiesta di connessione; si è notato che essa avveniva con un tempo minore del millisecondo dopo un pacchetto di Advertise. Il problema è stato individuato nella lentezza della trasmissione del dispositivo UART che impegnava per più di un millisecondo la CPU e che quindi non riusciva a riattivare in tempo il dispositivo Radio per ricevere la CONNECT_ REQ. Inibendo l'invio dei pacchetti di Advertise catturati, ma inviando solamente quello relativo all'evento di connessione, è stata immediata la sua cattura.

Successivamente alla riuscita della cattura del pacchetto di connessione si è quindi provato a impostarci su un canale di connessione, in modo fisso, per vedere se si riusciva nella cattura di un pacchetto. Partendo dal presupposto che i valori dell'AA e del CRCInit che il Nano2 catturava non erano da usare nell'ordine in cui li inseriva nel pacchetto catturato ed usando come similitudine il comportamento che utilizzava per altri campi del pacchetto, si è pensato che il loro valore reale fosse una specularità Byte a Byte del valore reperibile dal pacchetto. Seppur con svariati tentativi di connessione e inibendo anche la verifica del CRC, concentrandosi quindi unicamente sulla correttezza dell'Access Address, non si è riuscito a catturare nessun pacchetto nei canali data. É stato grazie ad un uso verboso dei log che si è individuato il problema: il dispositivo poco dopo l'avvio andava in crash. Questo comportava il riavvio dello stesso con la conseguenza che tutti i dati estratti dal pacchetto di connessione andavano sempre persi e quindi era impossibile catturare un qualsiasi pacchetto nei canali data. Solo risolvendo la causa del crash, ovvero un Null Pointer sulla struttura UART, è stato possibile catturare il primo pacchetto nei canali data.

7 Sviluppi futuri

Con lo svolgimento di questo lavoro di Tesi si sono create le basi per la creazione di un dispositivo di Relay, un sistema in grado di catturare tutti i pacchetti inviati da un certo dispositivo e replicarli ad una distanza di molto maggiore della portata trasmissiva del protocollo BLE; questo sistema può venire usato per implementare l'attacco Man in the Middle, ovvero un attacco informatico in cui un dispositivo attaccante ritrasmette e/o altera la comunicazione tra due parti che credono di comunicare direttamente tra di loro; nell'ambito Bluetooth questo attacco permette di far credere a 2 dispositivi lontani di essere a breve distanza e di comunicare. La riuscita di questo tipo di attacco permette di svelare le vulnerabilità di tutti quei sistemi che utilizzano la vicinanza tra 2 dispositivi Bluetooth come metodo di accesso a dispositivi protetti da un sistema di sblocco, normalmente con password. Ad oggi sono sempre di più le applicazioni che utilizzano un sistema keyless per permettere l'accesso ai soli autorizzati, basti pensare ad una serratura della porta principale di un'abitazione, che viene automaticamente sbloccata quando qualcuno in possesso della chiave si trova nelle strette vicinanze della porta e apre semplicemente la maniglia; con un relay di questo tipo un potenziale malintenzionato potrebbe intercettare il proprietario della casa mentre è ad esempio al supermercato, avvicinarlo con un dispositivo di ascolto mentre un suo complice si trova davanti alla sua porta di casa e con il dispositivo di ritrasmissione aprire facilmente la porta di casa. L'utilizzo di un dispositivo a basso costo rende più facile ed economica la creazione di un relay bluetooth, che può essere reperito ed utilizzato da un numero maggiore di attaccanti; come citato nelle premesse è molto importante che il protocollo trasmissivo BLE sia sicuro ed adotti misure per impedire che attacchi di questo tipo possano funzionare.

Per la creazione di un relay funzionante deve ancora essere sviluppata la parte di trasmissione dei pacchetti catturati a lunga distanza, attraverso mezzi trasmissivi come la rete cellulare utilizzando il protocollo LTE. Per la creazione del dispositivo che ritrasmette i pacchetti catturati, è già stato creato del codice funzionante che permette la trasmissione di pacchetti completamente personalizzati, utilizzato ampiamente nelle fasi di test dello Sniffer.

8 Bibliografia

Riferimenti bibliografici

[1] Bluetooth SIG Working Groups. *Bluetooth® Core Specification v4.0* 30 June 2010.

[2] Introduction to Bluetooth Low Energy,
https://learn.adafruit.com/introduction-to-bluetooth-low-energy

[3] CRC and data whitening,
http://dmitry.gr/index.php?r=05.Projects&proj=11.%20Bluetooth%20LE%20fakery

[4] Bluetooth LE Stack Partition,
https://www.eetimes.com/document.asp?doc_id=1278966

[5] ARM GNU Toolchain,
https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads

[6] Nordic SDK per nRF5,
http://developer.nordicsemi.com/nRF5_SDK/nRF5_SDK_v14.x.x

[7] GNU MCU Eclipse IDE for C/C++ Developers Neon,
https://github.com/gnu-mcu-eclipse/org.eclipse.epp.packages/releases/

[8] Ubertooth One negozio online,
https://www.antratek.com/ubertooth-one

[9] Official GitHub page of UberTooth,
https://github.com/greatscottgadgets/ubertooth

[10] Software Open Source per sniffare pacchetti con ubertooth,
https://github.com/greatscottgadgets/ubertooth/Bluetooth-Captures-in-PCAP