Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria dell'Informazione

Studio ed implementazione di architetture per l'IoT basate su rete mesh IPv6

Tesi di Dottorato di:
Lorenzo Palma

Tutor:
Prof. Paola Pierleoni

Coordinatore del Curriculum:
Prof. Francesco Piazza

XV ciclo - nuova serie

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum in Ingegneria dell'Informazione

Studio ed implementazione di architetture per l'IoT basate su rete mesh IPv6

Tesi di Dottorato di:
Lorenzo Palma

Tutor:
Prof. Paola Pierleoni

Coordinatore del Curriculum:
Prof. Francesco Piazza

XV ciclo - nuova serie

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Facoltà di Ingegneria
Via Brecce Bianche – 60131 Ancona (AN), Italy

Sommario

Questa tesi affronta lo studio e l'implementazione di architetture protocollari alla base dell'imponente sviluppo dell'Internet of Things. Si sono studiati ed implementati standard a tutti i livelli della pila protocollare, partendo dall'802.15.4 del basso livello, l'RDC per la gestione del risparmio energetico della radio, il 6LowPAN, il μIPv6 con protocollo di routing RPL ed UDP per il trasporto dei dati raccolti tramite applicativi http o CoAP. Si propone inoltre un protocollo per la gestione dell'aggiornamento del firmware over the air (FOTA) che consente di dare ulteriore flessibilità nello sviluppo e nella gestione delle potenziali installazioni. Sfruttando l'architettura sviluppata, si sono implementati innovativi sistemi di localizzazione indoor a basso costo, applicazioni per le Smart Cities, altre per l'agricoltura di precisione ed infine si è creato un ambiente di test Smart da esportare ed utilizzare per sistemi di Ambient Assisted Living integrati con strumenti per l'automazione domestica. Le performance dell'architettura sviluppata soddisfano a pieno i requisiti di scalabilità, interoperabilità, autoinstallazione ed automantenimento che una WSN per l'IoT deve garantire, inoltre i consumi dei dispositivi risultano anch'essi conformi alle aspettative di vita dell'ordine dei diversi mesi o anni che i sensori devono avere.

Abstract

This thesis deals with the study and implementation of protocol architectures at the base of the impressive development of the Internet of Things. The implemented protocols are standard on all levels of the protocol stack, starting from 802.15.4 the low level, the DRC for the power management of the radio, the 6LoWPAN, the μIPv6 with RPL routing protocol and UDP are designed and implemented for transporting data collected through http or COAP application. It is also proposed a protocol for the management of the firmware update over the air (FOTA) that allows to give further flexibility in the development and management of potential installations. Taking advantage of the architecture developed, I have implemented an innovative and low cost indoor location systems , applications for Smart Cities, a prototype for precision agricolture and finally a Smart environment test is created to test and prove the interoperation between Ambient Assisted Living systems with integrated tools for home automation. The performance of the developed architecture fully meet the scalability requirements, interoperability, auto-installation and self-maintenance that a WSN for the IoT must have. It also ensures a low power consumption of the devices that comply with the expected lifetime of several months or years that the sensors must have.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

In questa tesi approfondiremo diversi aspetti ma tutti profondamente legati al mondo delle WSN e al nuovo concetto di Internet of Things. Verrà data la descrizione generale dell'evoluzione delle Wireless Sensors Network verso il mondo dell'IoT, facendo anche un piccolo focus sui particolari ambiti applicativi.

Fino a qualche anno fa ogni produttore di componenti elettronici per WSN definiva anche un protocollo di comunicazione con il quale far comunicare i dispositivi delle reti, si finiva per avere migliaia di tipologie di WSN senza la possibilità di interconnessione tra le une e le altre, se non attraverso un nodo gateway appositamente progettato caso per caso. L'approccio dato in questa trattazione con l'utilizzo dell'IPv6 e dei protocolli comunemente utilizzati nell'Internet globale vuole finalmente creare un ambiente smart ed interoperabile dove possono coesistere diversi sensori accomunati dalla possibilità di connessione e visibilità diretta in internet grazie al proprio indirizzo IPv6 univoco.

Dato che i nodi delle WSN sono dei dispositivi embedded, l'obiettivo di poter implementare uno stack IP al loro interno risulta piuttosto ambizioso e non privo di difficoltà. A tale scopo si descrive l'utilizzo di un sistema operativo per dispositivi embedded in grado di aiutarci nello sviluppo dell'architettura e degli applicativi finali.

Data la vastità e la numerosità di dispositivi che possono comporre il mondo dell'IoT verrà proposta e descritta l'implementazione di un protocollo per l'aggiornamento firmware da remoto dei dispositivi installati.

Una volta descritti i vari protocolli implementati per la creazione della rete mesh si passerà alla simulazione tramite il Tool Cooja della rete mesh realizzata e si procederà con la descrizione dell'hardware scelto per lo sviluppo di applicazioni reali con la descrizione di queste ultime. Nel dettaglio parleremo di localizzazione indoor tramite la rete mesh IPv6 e analizzeremo i concetti relativi a tecniche di localizzazione indoor andando ad osservare studi e precedenti ricerche focalizzate su questo problema. Descriveremo lo sviluppo di un algoritmo per la stima della posizione di uno o più nodi mobili, implementato per lavorare con hardware a basso costo, garantendo comunque buone prestazioni. Seppur ancora in via prototipale, abbiamo realizzato delle prove in un ambiente indoor che più rispecchia un eventuale futuro impiego commerciale basato su questo studio.

Verranno poi descritte le applicazioni realizzate dando percezione della vastità di impiego del sistema realizzato. Fra queste troviamo un sistema che lavora nel mondo dell'efficientamento energetico oggi sperimentato in un comune toscano, un prototipo per l'agricoltura di precisione presentato all'EXPO 2015 ed un'installazione permanente presso il Dipartimento di Ingegneria dell'Informazione che propone un sistema di Ambient Assisted Living integrato con sistemi per il monitoraggio e l'automazione domestica. L'installazione consta di 40 nodi di cui 10 mobili ed è stata utilizzata per realizzare statistiche e test sulla rete mesh in condizioni reali.

Infine sarà descritto brevemente l'applicativo web realizzato ed operante con le API di Google Maps per la gestione delle fasi di installazione e monitoraggio della rete mesh di sensori IPv6 installata.

Capitolo 1

Dalle Wireless Sensor Network all'Internet of Things

In questo capitolo verranno descritti i concetti di Wireless Sensor Network ed analizzati i componenti essenziali con cui equipaggiare un sensore affinché questo possa considerarsi uno Smart Object e quindi entrare a far parte del mondo dell'Internet of Things. Sarà, inoltre, descritto l'obiettivo del presente lavoro di Tesi.

1.1 Il concetto di Wireless Sensor Network

Contrariamente a quanto si possa credere il 98% dei microprocessori non si trova nei pc-desktop o nei notebook, ma in dispositivi che ogni giorno utilizziamo per espletare le nostre normali attività domestiche, lavorative, di spostamento, di controllo e molti altri aspetti relativi alla vita quotidiana. Lo sviluppo che ha coinvolto in maniera prorompente molti campi della scienza e della tecnologia ci consente oggi di avere dispositivi esageratamente miniaturizzati che hanno, fino a pochi anni fa inimmaginabili, capacità di calcolo e d'interazione con l'ambiente circostante.

Approfondiamo il concetto di Wireless Sensor Network (WSN), concentrando l'attenzione sui singoli dispositivi che le costituiscono. Come ci lascia intendere il nome stesso si ha a che fare con reti di sensori dotati di connettività wireless e, poiché molteplici sono i campi di applicazione e di installazione risulta fondamentale la coesistenza e la minor invasività possibile rispetto agli oggetti di vita quotidiana. Da qui ne scaturisce una specifica fondamentale legata alle dimensioni che devono essere le più piccole possibili [1]. Sono elencate di seguito le principali caratteristiche di una WSN:

- elevato numero di nodi, spesso nell'ordine delle migliaia;
- flusso asimmetrico delle informazioni, la principale quantità di dati viaggia dai nodi sensore verso un nodo concentratore che di solito coordina anche la rete;

- le comunicazioni sono pilotate da eventi o da code;
- basso bitrate nell'ordine di 100kbps-250kbps;
- ogni nodo ha una quantità limitata di energia ed in molte applicazioni è impossibile ricaricare o sostituire la batteria;
- tipicamente le topologie di rete sono poco flessibili (punto-punto o stella);
- collegamenti ridondati in modo da migliorare la fault-tolerance;
- nodi caratterizzati da basso costo, scarso peso e piccole dimensioni;
- utilizzo eccessivo di comunicazioni broadcast invece che di comunicazioni point-to-point;
- i nodi sono distinti da ID univoci solo a livello locale;
- la sicurezza sia a livello fisico sia a livello superiore è molto limitata rispetto alle reti wireless convenzionali.

Ciascun nodo sensore appartenente ad una WSN è composto di un'unità di sensing che è in grado di trasdurre una grandezza fisica in una grandezza elettrica, da qui essa dovrà essere convertita da analogica in digitale tramite un Analog to Digital Converter, vi sarà poi un processore in grado di realizzare delle semplici elaborazioni sul dato ottenuto e che gestirà le operazioni di comunicazione, un transceiver radio doterà il dispositivo della connettività wireless necessaria per la trasmissione a distanza dell'informazione campionata.

Ogni dispositivo dovrà essere munito di un'unità di alimentazione, di solito costituita da una batteria o da un dispositivo di Energy-harvesting, dipenderà poi dalla particolare applicazione verso cui è destinato. Dato che questi nodi nella maggior parte delle applicazioni sono alimentati a batteria, è opportuno osservare come un aspetto fondamentale, oltre alla ricerca estrema di miniaturizzazione, sia il basso consumo energetico dei componenti elettronici che li costituiscono. Esso sarà contenuto se i componenti attivi presentano caratteristiche che ne consentano il funzionamento anche in modalità Low Power e permettano la gestione dello stato di Sleep, se il codice caricato nel processore è stato opportunamente ottimizzato e se il transceiver radio viene gestito in maniera tale da limitare il numero di trasmissioni necessarie e la commutazione dallo stato di ricezione a quello di sleep nei periodi temporali in cui si ha assenza di traffico. Riassumiamo gli aspetti chiave da considerare nella trattazione, nello studio e nella progettazione delle WSN:

- consumo di potenza: i nodi di una WSN sono, nella maggior parte dei casi, alimentati a batteria o hanno comunque scarsa energia; inoltre per molte applicazioni sarà impensabile andare a sostituire o ricaricare la batteria;

- dimensione fisica: è un aspetto importante perchè va a influenzare direttamente l'aspetto del nodo e, soprattutto, ne determina i possibili scenari applicativi;

- costi: sono essenziali e determinano sia la scelta dei componenti hardware che lo sviluppo del software residente. È significativamente importante che il costo del singolo nodo sia particolarmente contenuto affinché si possa ottenere una massiccia diffusione con milioni o miliardi di oggetti connessi.

Gli aspetti sopra descritti permettono di comprendere come la progettazione di un nodo per WSN sia al centro dell'attenzione di moltissimi ricercatori e aziende in quanto fortemente condizionata da tali fattori. Il contenimento della dimensione dei dispositivi lascia ampi spazi d'impiego per il prodotto finale, ma ciò equivale a dire che in poco spazio si devono ospitare tantissimi componenti. Determinante è il grado di miniaturizzazione che, spesso, determina scarsità di risorse di calcolo e di memoria. Quest'ultimo fatto andrà ad interferire fortemente anche con la progettazione del firmware dove l'ottimizzazione delle risorse di calcolo a disposizione e la gestione della scarsa quantità di energia di cui si dispone rappresentano una continua sfida per i progettisti.

1.2 Le WSN si estendono verso l'Internet of Things con la nascita degli Smart Objects

L'espressione Internet of Things (IoT) indica il collegamento ad internet di oggetti, luoghi, persone, animali, concreti e fisicamente presenti nel mondo reale. La definizione di Internet delle Cose è attribuita a Kevin Ashton che la usò per primo durante una presentazione presso Procter & Gamble nel 1999 [2].

I sensori associati ad un oggetto possono identificarlo univocamente e raccogliere informazioni in tempo reale su parametri del suo ambiente come: temperatura, localizzazione, pressione, rumore, luce, umidità. Questo nuovo scenario tecnologico unito al progresso delle tecnologie wireless, come le WSN, offre l'opportunità di sviluppare una serie di applicazioni innovative in cui gli oggetti che ci circondano e fanno parte della nostra routine quotidiana sono dotati di intelligenza artificiale e, pertanto, in grado di prendere decisioni autonome in funzione dell'ambiente e della sua evoluzione e mutazione nel tempo e nello spazio. Ecco che il concetto di Smart Object prende forma ed entra in maniera prepotente nella nostra normalità.

Se consideriamo la quantità e la varietà degli oggetti da gestire possiamo capire i problemi da risolvere per la raccolta e l'instradamento dei dati. Le

reti di Smart Object costituiscono l'infrastruttura per la raccolta, fusione e aggregazione dei dati immagazzinati nei nodi sensori. Un aspetto molto interessante delle WSN per l'IoT sviluppate negli ultimi anni è la possibilità di implementare un meccanismo multi-hop per il routing su reti di vasta area.

Fra i principali vantaggi delle nuove reti abbiamo la versatilità, ovvero sono in grado di supportare applicazioni molto differenti tra loro. Per questo sono particolarmente adatte a diversi campi applicativi, fra cui il controllo della salute del cittadino, la sicurezza, il controllo ambientale, l'efficientamento energetico, oltre a quelli in ambito militare dove, notoriamente, si ha un elevato contenuto di innovazione. Altri vantaggi rispetto ai tradizionali sistemi di misurazione sono la pervasività del monitoraggio e un'intrinseca tolleranza ai guasti.

Si è già sottolineato più volte come le WSN possano avere dimensioni gigantesche in termini di numero di nodi, la dimensione della rete ha un diretto impatto sulla progettazione del protocollo di rete. Essa ne condiziona direttamente le prestazioni sia in termini di quantità di dati trasportati che di velocità, inoltre influenza direttamente anche il tempo di vita della rete. Nell'ottica di una WSN caratterizzata da nodi alimentati a batteria, un protocollo di rete che prevede molti cicli di trasmissione impatta negativamente sul tempo di vita dei singoli nodi e quindi dell'installazione stessa. Per mantenere la rete si devono conservare delle informazioni di routing che inevitabilmente occupano risorse di memoria. Consegue che un protocollo di routing per una WSN deve fare i conti con la scarsità di energia e di memoria a disposizione per la conservazione delle informazioni di routing, quindi deve essere forzatamente orientato a limitare il numero di trasmissioni e la quantità di dati da mantenere in memoria per la gestione della rete. Fra i settori in cui si sta avendo una crescita esponenziale nell'utilizzo di Smart Object si hanno:

- Applicazioni militari:
  - monitoraggio delle forze, della strumentazione e degli armamenti;
  - sorveglianza del campo;
  - riconoscimento delle forze nemiche;
  - bersagli;
  - riconoscimento e rilevamento di attacchi nucleari, chimici e biologici.
- Applicazioni ambientali:
  - Monitoraggio della fauna selvatica;
  - Rilevamento d'incendi boschivi;
  - Rilevamento di inondazioni o alluvioni;
  - Ricerca geofisica e meteorologica;

  - Studio dell'inquinamento.

- Agricoltura di precisione e allevamento:
  - monitoraggio del livello dei pesticidi nell'acqua da bere;
  - irrigazione;
  - campionamento del terreno.

- Applicazioni in ambito medicale e assistenziale:
  - monitoraggio di parametri fisiologici umani;
  - tracciamento dei dottori e dei pazienti all'interno di un ospedale;
  - gestione dei farmaci negli ospedali;
  - assistenza agli anziani.

- Applicazioni domestiche:
  - smart home;
  - home automation.

- Sorveglianza:
  - sicurezza nei centri commerciali, nei parcheggi coperti e in strutture simili;
  - sorveglianza domestica.

- Smart Cities:
  - manutenzione condizionata d'infrastrutture civili come ponti, dighe, ecc. . . ;
  - controllo dello stress dei materiali;
  - controllo del traffico stradale;
  - tracciamento e rilevamento veicoli;
  - controllo degli ingorghi;
  - definizione dei percorsi.

- Produzione e logistica:
  - controllo e guida dei robot negli impianti automatici;
  - controllo dei processi industriali e automazioni;
  - diagnosi delle macchine e della strumentazione industriale;
  - gestione dell'inventario.

- Soccorsi in seguito a calamità naturali

- Esplorazioni spaziali

Quelle elencate sopra sono solo alcune delle possibili applicazioni delle WSN che stanno dando vita all'IoT, l'applicabilità delle stesse ad altri ambiti/usi è limitata solo dalla fantasia delle persone.

Fino a qualche anno fa le WSN erano pensate come strutture chiuse, ovvero costituite da uno svariato numero di sensori disposti su di un determinato sensing field dove si era interessati a monitorare determinate grandezze fisiche. I nodi sensore collezionano informazioni dal campo e le inviano verso un nodo collettore [3]; il nodo collettore ha risorse per elaborare e inviare tali informazioni a una rete di diversa natura come la rete internet. Sono esempi di questo modo di concepire le reti wireless i sistemi basati su protocolli molto conosciuti come ad esempio ZigBee [4] e Bluetooth [5], che oggi stanno anch'essi evolvendo verso soluzioni orientate al protocollo IP [6] ed ai protocolli internet in generale.

L'evoluzione di diverse tecnologie ha portato ad avere una più ampia veduta; se le WSN non fossero più strutture chiuse ma interconnesse tra loro o addirittura al World Wide Web? Ecco allora che nasce il concetto di Smart Object e di IoT.

Oggi si è più soliti parlare di IoT che di WSN, vediamo le differenze sostanziali tra i due concetti. L'IoT va visto come un'evoluzione delle WSN e non solo. Nelle WSN trovavamo dei soli nodi sensore, oggi abbiamo reti di Smart Object e ogni Smart Object è raggiungibile, e può raggiungere, il World Wide Web poiché ogni dispositivo sarà dotato di un indirizzo IP. Nelle WSN la connettività al WWW era delegata al nodo concentratore/sink che fungeva da gateway verso internet. La differenza sostanziale tra un nodo di una WSN e uno Smart Object del mondo IoT è che nel primo caso si hanno dei nodi specializzati nel sensing di grandezze fisiche, dotati di modeste capacità di calcolo e di connettività wireless, nel secondo caso si hanno nodi in grado anche di realizzare attuazioni e controlli, inoltre va sottolineato come uno Smart Object non nasca con il vincolo di dover funzionare solo sopra meccanismi di comunicazione wireless, ma può funzionare benissimo anche con meccanismi di comunicazione wired.

1.3 Hardware e software per l'IoT

L'hardware tipico che compone uno Smart Object è caratterizzato dai seguenti componenti elettronici:

- dispositivo di comunicazione: tipicamente è un transceiver radio più antenna o un connettore per collegamenti wired;
- microcontrollore: rappresenta il core del nodo sensore in quanto esegue il firmware dello Smart Object determinandone il comportamento;

- set di sensori e attuatori: sono gli strumenti attraverso i quali lo Smart Object interagisce con l'ambiente circostante;
- unità di alimentazione: è, forse, la parte più importante per il semplice motivo che l'elettronica che costituisce lo Smart Object ha bisogno di energia per funzionare.

In merito al dispositivo di comunicazione, che abbiamo detto essere un transceiver wireless, si può dire che tra tutti è il componente che consuma più energia in assoluto. Si può sottolineare come anche la sola fase di ascolto del canale sia dispendiosa quasi allo stesso modo della fase di trasmissione, quindi il modulo radio va gestito nella maniera più intelligente possibile. Il transceiver più banale è, semplicemente, in grado di trasmettere e ricevere singoli bit, vi sono poi transceiver più complessi che impacchettano l'informazione, quindi ricevono e trasmettono gruppi di bit e sono anche in grado di realizzare una cifratura/decifratura dell'informazione scambiata in modo da garantire un certo grado di sicurezza alla comunicazione instaurata. In merito al microcontrollore possiamo affermare che esso rappresenti la parte intelligente dell'oggetto. Un microcontrollore tipico per applicazioni IoT ha poche centinaia di kbyte di memoria e lavora a poche decine di MHz. Dispone inoltre di due tipologie di memoria ovvero ROM e RAM. Nella memoria ROM è conservato il programma principale da eseguire, nella RAM trovano posto le variabili temporanee per l'esecuzione del programma e i buffer di memoria per la gestione del traffico radio. Nei moderni microcontrollori più che di ROM si parla di FLASH, di Data-EEPROM e di SRAM invece che di RAM. Sostanzialmente la FLASH e la EEPROM sono la stessa cosa, solo che la FLASH è comunemente utilizzata per memorizzare il programma principale, può essere cancellata/riprogrammata, in linea teorica, infinte volte on-circuit e permette l'accesso a blocchi logici. La Data-EEPROM è essa stessa cancellabile/riprogrammabile on-circuit, permette un accesso bit a bit o per byte. La SRAM è una memoria volatile ad accesso casuale e particolarmente rapido.

Il microcontrollore è dotato oltre che di microprocessore e di memoria anche di timers e dispositivi hardware utili per l'interconnessione con sensori, attuatori, e dispositivi di comunicazione. La interfacce più comuni presenti in qualsiasi microcontrollore sono la USART (Universal Synchronous/Asynchronous Receiver/Trasmitter) per la comunicazione con porte seriali, molte USART possono essere configurate come SPI (Serial Peripheral Interface) bus per la comunicazione con sensori e attuatori, l'I2C (Inter Integrated Circuit), è un sistema di comunicazione seriale bifilare utilizzato tra circuiti integrati. Il classico bus I2C è composto da almeno un master ed uno slave. La situazione più frequente vede un singolo master e più slave (tipicamente sensori); possono tuttavia essere usate architetture multimaster e multislave in sistemi più complessi, l'USB

(Universal Serial Bus), è un'interfaccia standard industriale di comunicazione seriale sviluppata a metà degli anni novanta per standardizzare in un unico protocollo di comunicazione cavi e connettori da utilizzare per la connessione, la comunicazione e l'alimentazione tra computer e periferiche elettroniche.

In merito ai sensori e agli attuatori con cui può essere equipaggiato uno Smart Object possiamo dire ben poco, per il semplice motivo che dipende dalla particolare applicazione cui è destinato l'oggetto considerato. Riguardo all'unità di alimentazione possiamo sottolineare come questa sia essenziale per il funzionamento dell'intero oggetto. Nella stragrande maggioranza delle applicazioni delle WSN l'unità di alimentazione è rappresentata dalla batteria, che oltre a fornire energia deve essere poco ingombrante. Le ridotte dimensioni fanno sì che si abbia il grosso inconveniente di poter immagazzinare poca energia. Ad oggi si utilizzano per lo più batterie al Litio organizzate in celle. È impensabile sostituirle o ricaricarle su di una miriade di oggetti interconnessi nel mondo dell'IoT, quindi l'unico modo per aggirare questo problema è far in modo che la durata sia il più lunga possibile. Una delle soluzioni sviluppate per far fronte alla richiesta di energia consiste nell'equipaggiare i nodi con sistemi di energy harvesting. Tali dispositivi sono così in grado di ottenere energia utile direttamente dall'ambiente circostante, ad esempio dai campi elettromagnetici, dalle vibrazioni, dalla radiazione solare, dal movimento, ecc...

Per quanto riguarda il software presente su di uno Smart Object possiamo solo rilevare come debba essere altamente orientato ai principali compiti che deve svolgere, ottimizzato in modo da ridurre il numero di cicli macchina per compiere l'elaborazione e allo stesso tempo deve avere un ridotto footprint in memoria, viste le limitate risorse a disposizione. Data la massiccia, recente, diffusione di oggetti connessi e viste le sempre maggiori funzionalità richieste a questi ultimi, i recenti studi ed implementazioni sono ricorsi all'utilizzo di Sistemi Operativi (SO) per dispositivi embedded [7], [8]. I sistemi operativi per Smart Object sono particolarmente leggeri ed attenti al consumo di risorse. L'utilizzo di un sistema operativo in queste situazioni può essere utile perché lo Smart Object è chiamato a svolgere molti compiti, quindi il Sistema Operativo può gestire al meglio le risorse di cui dispone il microcontrollore e allo stesso tempo può permettere un'esecuzione sistematica di certi compiti. Il software che gira su di uno Smart Object deve implementare anche il protocollo di rete, utilizzato dal nodo per comunicare con gli altri attori della rete, tale protocollo è sviluppato in una struttura a strati e siccome ogni layer è stacked sopra ad un altro, è comune chiamare il protocollo di comunicazione con il nome di stack di rete. Nelle capitolo 3 approfondiremo un particolare sistema operativo embedded utilizzato per la realizzazione del lavoro oggetto di questa Tesi.

1.4 Obiettivi della ricerca e del lavoro svolto

In questa Tesi vengono affrontate le problematiche relative alla scelta ed all'implementazione di protocolli alla base dello sviluppo dell'IoT. Si descrive come ogni dispositivo può entrare a far parte del mondo connesso e si propone e giustifica la scelta del protocollo IPv6 come mezzo di unificazione e strumento per garantire l'interoperabilità della miriade di Smart Object presenti oggi nel mondo. Si analizza il SO Contiki e si descrivono una serie di applicazioni sviluppate e testate anche nel mondo industriale capaci di sfruttare la rete mesh IPv6 implementata nel triennio di Dottorato.

Capitolo 2

Lo stack protocollare implementato e la rete mesh IPv6 nel mondo dell'IoT

In questo capitolo sono trattati e brevemente descritti i protocolli scelti per la realizzazione del lavoro di Dottorato e quindi della rete mesh di sensori per l'IoT. Vengono descritte e giustificate le scelte effettuate per ciascun livello della pila protocollare, da quello fisico a quello applicativo, fino alla modalità di interazione con il web.

2.1 802.15.4

Nell'ambito delle comunicazioni wireless e più precisamente nelle wireless a corto raggio è stato definito dall'IEEE (Institute of Electrical and Electronic Engineers) uno standard chiamato 802.15.4. Questo standard è pensato per comunicazioni wireless di basso costo, bassa velocità e basso consumo energetico. È adatto a reti WPAN (Wireless Personal Area Network) a basso bit rate costituite da dispositivi alimentati tramite batterie che non possono essere sostituite frequentemente, come sono i sensori e i nodi di rete per l'IoT [9]. Le sue principali caratteristiche sono:

- la capacità di coprire un raggio di azione di poche decine di metri offrendo fino ad un bit rate massimo di 250 Kbps;
- offrire una bassa potenza in trasmissione con basso consumo energetico;
- permettere ai livelli superiori la presenza di livelli di rete che possono effettuare routing;
- infine offrire la possibilità di utilizzare ACK e quindi di poter effettuare ritrasmissioni dei dati in caso di mancata ricezione o di errori di trasmissione.

Lo standard 802.15.4 definisce il livello 1 (livello fisico) e il livello 2 (livello MAC). I dispositivi implementati sulla base delle specifiche IEEE 802.15.4, possono avere due diverse modalità di funzionamento: Reduced Function Device (RFD) e Full Function Device (FFD). FFD ha funzioni da coordinatore, può funzionare con qualsiasi topologia di rete e si fa carico delle connessioni con gli altri dispositivi. L'RFD è utilizzato unicamente nelle topologie a stella, non può diventare coordinatore della rete e può colloquiare solamente con esso, pertanto non risulta utilizzabile per il lavoro oggetto di questa tesi.

La possibilità offerta dal protocollo ai dispositivi, di poter creare un collegamento con altri dispositivi adiacenti, dà la possibilità di poter costruire reti di grandi dimensione e molto complesse, passando da strutture a stella a topologie mesh formate da cluster di dispositivi.

2.1.1 Livello fisico

I servizi che il livello fisico offre intervengono sull'accensione e sullo spegnimento del modulo radio, questa opzione permette al sistema il risparmio d'energia. A livello 1 viene anche effettuata la scelta del canale di comunicazione migliore, calcolando una stima sull'occupazione del canale e sul rapporto segnale/rumore SNR, diminuendo così la probabilità di errori in trasmissione.

Infine modula e demodula i segnali in ricezione e in trasmissione. Il livello fisico opera nella Banda ISM utilizzando tre possibili bande libere:

- 868-868.6 Mhz: banda utilizzata nella maggior parte dei paesi europei; dispone di un solo canale;
- 902-928 Mhz : banda utilizzata nel continente oceanico e nel continente americano, offre 10 canali disponibili;
- 2400-2483.5 : banda utilizzabile in quasi tutto il globo con un data-rate massimo di 250 Kbps e 16 canali a disposizione.

Nel nostro caso si utilizza la banda a 868MHz.

La modulazione del segnale più diffusa è il DSSS utilizzando le tecniche BPSK o QPSK, ma anche FSK e GFSK, anche se recentemente sono state introdotte nuove modulazioni come la PSSS. Il datagram a livello fisico è formato dai seguenti campi: all'inizio si trova il preambolo, formato da 32 bit, con finalità di sincronizzazione tra i nodi, successivamente viene utilizzato un ottetto (11100101) prefissato che funziona da indicatore di inizio pacchetto. Segue un campo physical header che indica la lunghezza del payload (Phisical Service Data Units) che può avere una lunghezza massima di 127 bytes.

2.1.2 Livello MAC

Il secondo livello (MAC) offre servizi quali la possibilità di creare una rete PAN, la trasmissione dei beacon e l'accesso al canale tramite il protocollo CSMA/CA. Questo livello supporta algoritmi di cifratura basati su AES-128 per la sicurezza dei dati, gestisce inoltre l'handshake cioè gli Acknowledge per la ritrasmissione dei dati in caso di mancata o erronea ricezione di questi ultimi. Infine calcola e verifica l'integrità della PDU (Protocol Data Unit, ovvero l'unità di informazione scambiata tra due entità). Inoltre può supportare reti fino ad un massimo di 65536 nodi utilizzando un indirizzamento fino a 16 bit.

Il livello MAC prevede una struttura chiamata superframe. Questa frame è costruito dal coordinatore della rete ed è contenuta tra due messaggi chiamati beacon. I beacon contengono informazioni che possono essere utilizzate per la sincronizzazione dei dispositivi, per l'identificazione della rete, per descrivere la struttura del superframe e la periodicità della loro spedizione. Il superframe è diviso in 16 slot temporali di uguale grandezza, dove il beacon frame è trasmesso nel primo e nell'ultimo slot di ognuna. Si possono avere due tipi di superframe: senza GTS (Guaranteed Time Slot) oppure con GTS. Quando viene inviato un superframe senza GTS, l'accesso al canale è regolarizzato dal protocollo CSMA/CA, come avviene nel nostro caso, dove ogni dispositivo deve competere con gli altri per assicurarsi l'accesso ad uno slot. Questo periodo è chiamato CAP (Contention Access Period). In altri tipi di applicazioni, invece, si necessita di costruire reti tali da garantire a tutti i nodi di poter trasmettere. È il caso del superframe con GTS, la quale dedica fino ad un massimo di sette slot temporali, detti CFP (Contention-FreePeriod), a determinati nodi. In questo periodo possono comunicare solamente i nodi prestabiliti e l'accesso al canale non è più CSMA/CA .

2.1.3 Modalità di trasferimento dati

La trasmissione dei dati può avvenire in tre modi differenti a seconda di chi spedisce i dati. La prima modalità si riferisce al trasferimento dati dal nodo al coordinatore, la seconda da coordinatore a nodo e la terza tra due nodi. Le prime due modalità possono essere utilizzate in tipologia di reti a stella, mentre per le reti mesh possono essere utilizzate tutte e tre le modalità.

È possibile utilizzare due meccanismi differenti per la trasmissione dei dati:

- trasmissioni basate sui beacon: Beacon-Enable;
- trasmissione senza l'utilizzo di beacon: Beacon-Disable.

Con una rete beacon-enabled, quando un nodo deve trasferire i dati ad un coordinatore, ascolta il beacon e si sincronizza al superframe. Il nodo trasmette il relativo pacchetto al coordinatore il quale, opzionalmente dopo aver ricevuto

i dati, trasmette un pacchetto di conferma dell'avvenuta ricezione al dispositivo (ACK).

In una rete beacon-disable, quando un dispositivo desidera trasferire dei dati al coordinatore, trasmette semplicemente utilizzando il protocollo di accesso al canale CSMA/CA. Anche in questo caso il coordinatore, dopo la ricezione dei dati, trasmette un ACK opzionale. La comunicazione diventa più articolata quando è il coordinatore che necessita di comunicare con un nodo. Se la rete è beacon-enabled, il coordinatore indica nel beacon la necessità di trasferire dati ad un dispositivo, il quale risponde attraverso una PDU chiamata MAC command. La MAC command ha il significato di conferma all'invio dei dati. Il coordinatore, dopo aver ricevuto la MAC command, spedisce in successione una PDU ACK di conferma e successivamente i dati. Il nodo a trasferimento completato invia un ACK. Se la rete è beacon-disabled, il trasferimento dati da coordinatore a nodo avviene allo stesso modo, con la sola differenza che il coordinatore memorizza i dati finché non è il nodo stesso a stabilire la connessione. In questa modalità si ricorda che ogni accesso è fatto attraverso il CSMA/CA. Per quanto riguarda i tipi di frame a livello MAC l'IEEE 802.15.4 ne definisce quattro tipi:

- Beacon Frame. Sono utilizzati per delimitare un superframe e per coordinare la sincronizzazione tra nodo e coordinatore;
- Data Frame. Questo frame ha un payload massimo di 104 byte. Il suo compito principale è la trasmissione dei dati ma fornisce altri servizi come la tracciatura dei pacchetti ed il controllo sull'integrità della PDU;
- ACK Frame. Il compito dell'ACK frame è quello di fornire un feedback attivo dal ricevitore della corretta ricezione di un pacchetto;
- MAC Command Frame. Esso consente meccanismi per il controllo e la configurazione dei vari nodi come nel caso di comunicazioni tra coordinatore e nodo.

2.2 Radio Duty Cycling

Il livello Radio Duty Cycling (RDC) si occupa di ottimizzare la gestione della radio per conseguire l'ottimizzazione delle risorse energetiche, è necessario per l'alimentazione a batteria dei nodi della rete. La gestione dell'alimentazione è determinante per i nodi sensori wireless per ottenere un elevato tempo di vita della rete. Per raggiungere questo obiettivo il solo hardware radio a basso consumo energetico non è sufficiente. Esistono transceiver radio a bassa potenza ma il loro impatto risulta comunque troppo elevato per garantire lunghi tempi di vita dei nodi. Ad esempio, il ricetrasmettitore Spirit1, prodotto dalla

STMicroelectronics, e venduto come dispositivo low energy richiede circa 60 milliwatt di potenza solo quando si è in ascolto per il traffico radio. Il suo consumo di energia durante la trasmissione dei dati via radio è leggermente superiore. Con un assorbimento di potenza di 60 milliwatt un nodo sensore esaurirebbe le batterie nel giro di pochi giorni.

Per ottenere una lunga durata delle risorse energetiche, il ricetrasmettitore radio deve restare spento quanto più tempo possibile. Ma di contro quando la radio è spenta, il nodo non è in grado di inviare o ricevere messaggi. Ecco che si rende indispensabile una gestione tale da permettere ai nodi di ricevere e trasmettere messaggi, che però mantenga il transceiver spento fra ricezione e trasmissione ottimizzandone l'utilizzo.

Lo scopo di un protocollo di duty cycling è proprio quello di conseguire il risparmio energetico mantenendo la radio spenta e fornendo abbastanza punti d'incontro per due nodi cosí da consentire la comunicazione tra di loro. Ci sono diversi modi di fornire punti di incontro in un protocollo RDC. Se i nodi sono tempo-sincronizzati, il protocollo può impostare e mantenere un programma di quando questi possono comunicare. Il protocollo dovrebbe accendere la radio in un tempo noto e permettere ai vicini di trasmettere i pacchetti in questo dato momento. Senza la sincronizzazione dell'ora programmata, i nodi devono utilizzare altri mezzi per fornire punti di incontro. Nella comunità dei ricercatori che lavorano nell'ambito delle reti di sensori, i protocolli RDC sono spesso chiamati protocolli MAC perché è proprio a questo livello che sono implementati i meccanismi di controllo dell'energia consumata dal transceiver. Nel nostro caso si è diviso il livello MAC da quello di gestione del consumo della radio e si è trasferito in un proprio livello, chiamato proprio RDC.

2.2.1 Ascolto del canale a basso consumo

Il mantenimento della radio in ascolto a bassa potenza è una tecnica fondamentale dei protocolli RDC in cui i ricevitori eseguono una periodica interrogazione del mezzo fisico per rilevare eventuali attività. Se c'è traffico nel canale, la radio viene mantenuta in ascolto per un tempo più lungo in modo tale da permettere l'invio e la ricezione dei pacchetti dati del nodo che deve inviarne. Prima di inviare un dato, il mittente invia un numero di pacchetti di segnalazione (Strobe) con l'intento di notificare al ricevitore che un nodo è in possesso di informazioni da inviare. Dato che tutti i vicini, periodicamente, effettuano il campionamento del mezzo radio, questi intercetteranno i pacchetti di Strobe e potranno così mantenere la loro radio accesa in attesa del pacchetto dati. Il periodo di strobe è definito come il più lungo periodo di sonno dei vicini in modo che questi possano essere svegliati dai pacchetti di segnalazione.

Ci sono diverse varianti di ascolto a basso consumo. Alcune non inviano pacchetti di segnalazione per svegliare i loro vicini, ma inviano un tono di sveglia di lunghezza definita tale da poter essere intercettato dagli altri nodi. Questo tono serve allo stesso scopo dei pacchetti di strobe, ma non può contenere alcuna informazione. Il grosso limite di tale tecnica è quello di non consentire l'inserimento dell'indirizzo del ricevitore come avviene per i pacchetti di segnalazione. In questo modo i vicini possono decidere rapidamente se rimanere svegli in attesa del pacchetto dati, laddove l'indirizzo indichi che tale dato è diretto al nodo stesso, oppure tornare con la radio spenta.

B-MAC [10] è un primo esempio di un protocollo di ascolto a basso consumo. B-MAC, che è stato inizialmente sviluppato per ricetrasmettitori radio che non implementavano automaticamente la pacchettizzazione dei byte in uscita e non utilizzavano i pacchetti di strobe, ma un tono di sveglia fra un invio ed il successivo.

WiseMAC [11] è simile a B-MAC, ma ottimizza ulteriormente la trasmissione consentendo a tutti i nodi di registrare il periodo di campionamento della radio dei loro vicini. Quando si invia un pacchetto a un dato vicino, il mittente attende che il vicino sia noto prima di testare il mezzo radio per poi inviare il suo tono di sveglia. Così facendo il tono di sveglia verrà inviata poco prima che il ricevitore si svegli, risparmiando ulteriormente energia.

X-MAC [12] è stato il primo protocollo di ascolto a basso consumo che è stato sviluppato per una radio che implementa la pacchettizzazione dei byte in invio, e il primo protocollo ad utilizzare lo strobe al posto di un tono di sveglia. In X-MAC, i pacchetti di segnalazione contengono l'indirizzo del destinatario cosicché gli altri vicini possano astenersi dal mantenere la loro radio accesa quando captano la segnalazione per un altro nodo. Inoltre, X-MAC ottimizza ulteriormente il meccanismo aggiungendo un pacchetto strobe di riconoscimento. Questo pacchetto viene inviato da un ricevitore in risposta ad uno strobe e contiene il suo indirizzo. Il mittente, dopo aver catturato il pacchetto di strobe, invia subito il proprio pacchetto dati.

2.2.2 Segnalazione a basso consumo

Il sondaggio del canale a bassa potenza è un metodo di sincronizzazione delle trasmissioni che inverte i ruoli rispetto all'ascolto a basso consumo. Invece di avere ricevitori che campionano periodicamente il mezzo radio, i ricevitori trasmettono periodicamente un segnale di test nel mezzo radio. Per inviare un pacchetto, il mittente accende la sua radio e ascolta se vi è o meno la presenza di un pacchetto di test dal vicino che deve ricevere il pacchetto di dati. Quando il pacchetto di presenza arriva, il mittente invia immediatamente il suo pacchetto di dati. Il ricevitore, che mantiene la sua radio accesa per un breve periodo

dopo l'invio del dato di presenza, sarà così in grado di ricevere il pacchetto di dati.

Sono stati pubblicati due diversi protocolli per la segnalazione della presenza in ascolto. Il primo, chiamato semplicemente Low Power Probing (LPP) [13], utilizza il meccanismo che consente di svegliarsi ad una sola sezione della rete. Il protocollo RI-MAC [14] è sviluppato come un protocollo MAC completamente funzionale basato sulla stessa idea dell'LPP. Sebbene i due protocolli siano stati pubblicati uno dopo l'altro, sono comunque stati sviluppati in modo indipendente.

2.2.3 Il protocollo RDC utilizzato

Contiki fornisce tre protocolli per gestire il Cycling radio a livello MAC: il protocollo ContikiMAC, Contiki X-MAC e Contiki LPP. ContikiMAC [15] è un protocollo basato sull'ascolto a bassa potenza con un'efficienza in termini di consumo energetico molto elevata. Il Contiki X-MAC si basa sul protocollo X-MAC originale, ma con un insieme significativo di miglioramenti ed estensioni. Il Contiki LPP si basa sulle stesse idee come l'originale LPP e RI-MAC, ma con una serie di modifiche.

In entrambi i protocolli X-MAC e LPP, i nodi si svegliano periodicamente in previsione di un potenziale pacchetto in arrivo. Il periodo di wake-up è fisso, ma i nodi non sono sincronizzati. Pertanto, vi è uno sfasamento tra tutti i nodi adiacenti. Utilizzando le informazioni per la sincronizzazione fra vicini, è possibile risparmiare energia quando è necessario l'invio di pacchetti. Sia il Contiki X-MAC che il Contiki LPP fanno uso di questa fase di allineamento. Sia il Contiki X-MAC che il Contiki LPP forniscono una funzionalità di streaming in cui i pacchetti possono essere contrassegnati con un flag che indica se si tratta di un flusso. I pacchetti contrassegnati con il flag flusso fanno parte di un gruppo di dati con priorità superiore, e il livello MAC li tratta in modo diverso rispetto ad altri.

2.3 Il livello di adattamento 6LowPAN

6LoWPAN [16] è stato progettato per adeguare i pacchetti IPv6 al trasporto tramite IEEE 802.15.4 e rispondere alle seguenti esigenze, tipiche del mondo dell'IoT:

- i dispositivi hanno limitata energia e devono gestire il duty-cycle, mentre IP presuppone che i device siano sempre attivi;

- le tecnologie wireless, come IEEE 802.15.4, non supportano il multicast, molto importante per alcune caratteristiche di IPv6, l'alternativa del flooding consuma energia e banda;
- le applicazioni dei dispositivi wireless nell'IoT beneficiano del multihop mesh networking, difficilmente integrabile con le normali soluzioni di routing IP;
- le tecnologie radio utilizzate dai sensori wireless hanno una banda limitata e un frame di dimensioni ridotte, ad esempio IEEE 802.15.4 ha un frame di 127 byte, con payload disponibile nel range tra 72 e 116 byte mentre IPv6 prevede un frame di dimensioni di 1280 byte;
- i protocolli standard non sono ottimizzati per reti wireless a bassa potenza, nelle quali può verificarsi la caduta di un nodo per failure o per esaurimento dell'energia.

La creazione di uno standard per il trasporto di IPv6 nelle reti di sensori wireless ha permesso di ridurre al minimo i prerequisiti e la riprogettazione di protocolli come quelli utilizzati per il Neighbor Discovery (ND) consentendo di mantenere inalterate le caratteristiche peculiari di tali reti.

Alcuni concetti di 6LoWPAN sono strettamente legati alle funzione del link-layer, come ad esempio il PAN ID nella gestione dell'indirizzamento, ma il gruppo di lavoro sul 6LoWPAN ha voluto mantenere un livello di generalità, evitando di utilizzare tutte le caratteristiche peculiari del livello IEEE 802.15.4. Le ridotte funzionalità richieste al MAC layer sottostante permetteranno di applicare 6LoWPAN anche in futuro ad un range di nuove tecnologie.

L'adaptation-layer realizza tre compiti fondamentali:

- gestisce la frammentazione/deframmentazione dei pacchetti IP;
- realizza la compressione degli header;
- gestisce l'inoltro dei pacchetti quando si utilizza il multi-hoping a livello due (questa funzionalità nel nostro caso non ci interessa perché la gestione del routing avviene a livello tre).

Il protocollo 6LoWPAN definisce quello che viene chiamato encapsulation header stack che precede ogni datagramma IPv6. L'encapsulation header stack di 6LoWPAN è costituito da tre diversi header: Mesh addr header, Fragmentation header e IPv6 header compression; quando presenti compaiono nell'ordine riportato in Fig. 2.1.

La frammentazione del pacchetto IP viene richiesta all'adaptation-layer quando il pacchetto IP non entra nella MTU prevista da 802.15.4; in tal caso il pacchetto IP (datagramma) viene spezzato in più link-frame, tali link-frame

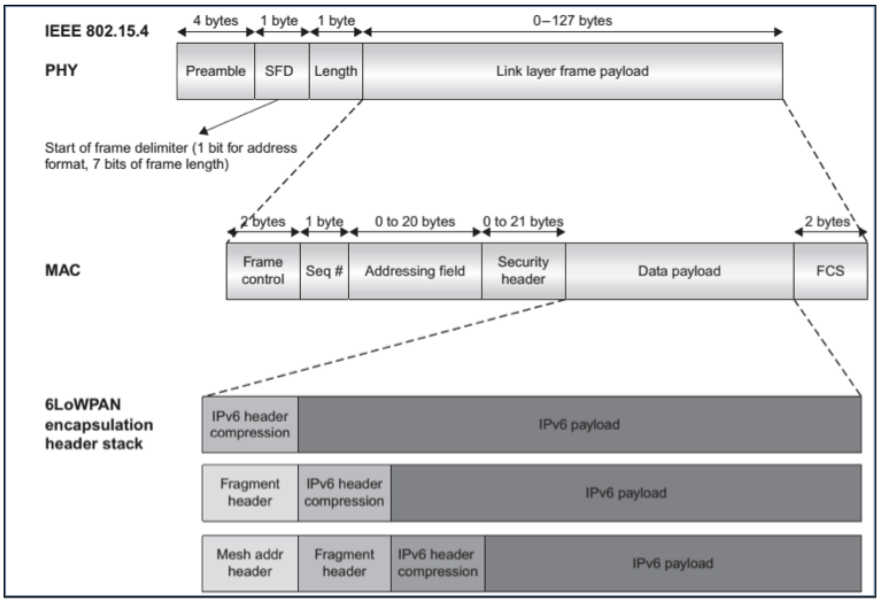

Figura 2.1: L'encapsulation header stack come previsto da 6LoWPAN; la parte Mesh addr introdotta da 6LoWPAN si riferisce al caso in cui il routing avvenga a livello Data-Link; nel nostro caso il routing avviene a livello Network

sono preceduti da particolari header il cui modello è riportato in Fig. 2.2. Il primo frammento non contiene il campo *datagram_offset*, mentre gli altri lo contengono tutti. Descriviamo molto rapidamente i campi del Fragmentation header di 6LoWPAN:

- *datagram_size*: campo di 11-bits, esprime la lunghezza del datagramma IPv6 originale. Tale campo è contenuto solo nel primo frammento dell'intero pacchetto;
- *datagram_tag*: viene utilizzato per indicare univocamente un datagramma, tale campo è uguale in tutti i frame che appartengono allo stesso datagramma IPv6;
- *datagram_offset*: è un campo ad 8-bit, è contenuto in tutti i frammenti data-link appartenenti allo stesso datagram IPv6, eccetto che nel primo frame. Serve ad indicare la posizione di un frame all'interno dello stesso datagram.

6LoWPAN prevede, inoltre, l'esistenza di un timer che viene avviato alla ricezione del primo frame di un determinato datagram, tale timer può avere

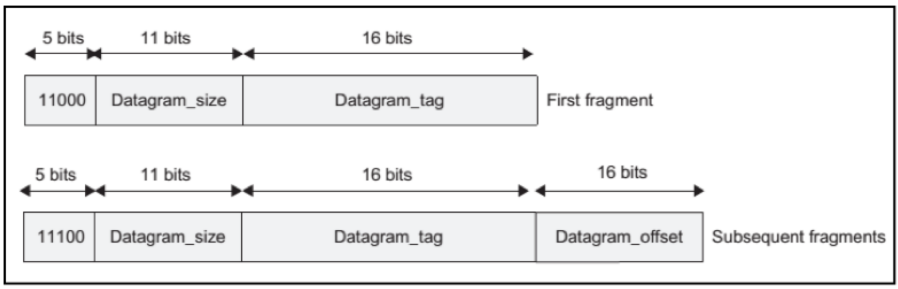

Figura 2.2: Struttura del Fragmentation Header del primo frammento e dei successivi

durata massima 60 secondi, se allo scadere del timer non si hanno tutti i frame il datagram viene scartato.

2.4 Il protocollo IPv6

Dalla nascita dell'internet globale ad oggi il protocollo di livello rete ha subito diverse rivisitazioni, dal 1981 ciò che lavora su miliardi di host è IPv4. A seguito di alcuni problemi che sono sorti con lo sviluppo esponenziale della rete internet, dopo anni in cui la rete internet globale è stata gestita in IPv4, ora esso si appresta a lasciare posto alla nuova versione IPv6 [17]. La transizione da IPv4 ad IPv6 sta gradualmente avvenendo, ma sarà particolarmente lenta e farà sì che IPv4 rimanga attivo ed utilizzato ancora per molti anni.

Evidenzieremo in poche righe le sostanziali novità introdotte da IPv6 e che quindi lo differenziano da IPv4. IPv6 nasce principalmente per risolvere il problema del limitato spazio indirizzi di cui dispone IPv4; tale spazio indirizzi è diventato limitato a seguito del massiccio sviluppo che ha avuto internet grazie anche alla nascita degli Smart Object ed all'avvento della nuova visione dell'IoT. Gli ideatori di IPv4 più di trent'anni fa non potevano nemmeno lontanamente immaginare una crescita di queste proporzioni.

Lo spazio di indirizzi in IPv4 è rappresentato da indirizzi IP a 32-bit, in IPv6 lo spazio indirizzi è rappresentato da indirizzi IP a 128-bit; giusto per comprendere le proporzioni IPv4 mette a disposizione circa quattro miliardi d'indirizzi, IPv6 ne mette a disposizione 2^{128}, in pratica infiniti. L'ampliamento dello spazio indirizzi è di sicuro il motivo principale per cui nasce IPv6, ma IPv6 ha ulteriori funzionalità che non erano previste in IPv4 o al massimo erano opzionali. IPv6 introduce capacità di autoconfigurazione della rete, aggiunge delle modifiche all'header del pacchetto IP, introduce dei meccanismi per l'integrità e la confidenzialità delle informazioni e dei sistemi di sicurezza.

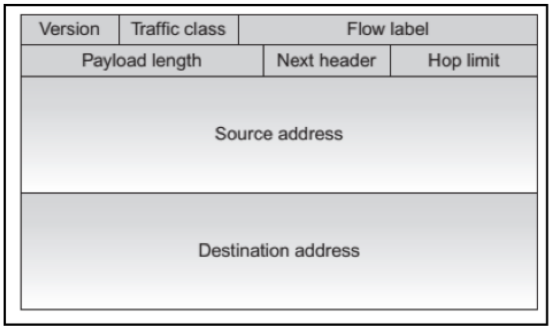

Figura 2.3: Struttura dell'header di un pacchetto IPv6

L'header del pacchetto IPv6 è complessivamente lungo quaranta bytes, contro i venti dell'header di un pacchetto IPv4. Osserviamo com'è fatto l'header di un pacchetto IPv6 (Fig. 2.3), elenchiamo qui di seguito i campi che lo costituiscono:

- version (4 bits): versione del protocollo IP=6;
- traffic class (8 bits): campo di 8-bit, indica la Class of Service (CoS) del pacchetto;
- tlow label (20 bits): è un campo utilizzato da un nodo sorgente per indicare una sequenza di pacchetti che richiedono una particolare attenzione da parte dei routers lungo il percorso che conduce alla destinazione finale (l'utilizzo di questa label è ancora sperimentale);
- payload length (16 bits): campo che indica la lunghezza del payload, escluso l'header del pacchetto. La lunghezza di eventuali extension headers utilizzati è inclusa nella lunghezza del payload;
- next header (8 bits): campo che identifica l'extension header che segue quello del pacchetto IPv6. Questo è un valido meccanismo per concatenare più extension header;
- hop limit (8 bits): questo campo è decrementato ogni volta che il pacchetto è inoltrato da un nodo, quando è zero il pacchetto viene scartato;
- source address: 128-bit, contiene l'indirizzo della sorgente del pacchetto IPv6;
- destination address: 128-bit, contiene l'indirizzo IPv6 di destinazione del pacchetto.

Come descritto sopra, il campo Next-Header contenuto nell'header del pacchetto IPv6 specifica il tipo di un eventuale Extension-Header. Serve a segnalare che il payload del pacchetto IPv6 considerato non inizia con dei dati utili ma i primi n-bit appartengono ad un certo extension-header. Tale meccanismo consente di concatenare un numero qualsiasi di extension-headers. Riportiamo un elenco dei possibili extension-headers previsti da IPv6:

- hop-by-hop options: è un header che deve essere processato da tutti i routers che il pacchetto incontra lungo il percorso sorgente-destinazione;
- routing: serve a indicare un certo numero di nodi che il pacchetto deve obbligatoriamente attraversare lungo il percorso sorgente-destinazione;
- fragment: IPv6 contempla una MTU di 1280 bytes ma nelle WSN 802.15.4 la MTU sono 127 bytes, ne consegue che sia necessaria la frammentazione del pacchetto IP. Questo campo serve a distinguere i frammenti di un determinato pacchetto IP e a ricomporli in modo opportuno;
- destination options: serve a inserire informazioni accessorie che saranno processate dal nodo destinazione;
- authentication;
- encapsulating security payload.

IPv6 classifica gli indirizzi in Unicast, Anycast e Multicast, non esiste il Broadcast di IPv4. Gli indirizziIPv6 Unicast si compongono di due parti:

- il prefisso di rete (primi64 bit);
- l'interface ID (ultimi64 bit): è rappresentato dal mac address o dall'identificatore EUI-64 che non è altro che un'evoluzione del precedente.

Esistono tre varianti d'indirizzo IPv6 Unicast: Link-Local, Site-Local (deprecated) e Unique Local Unicast. In merito all'indirizzo Link-Local IPv6 possiamo riassumere:

- è uno Scoped address: Scope (ambito) = link locale, può essere usato solo fra nodi dello stesso link, non può essere ruotato;
- fornisce a ogni nodo un indirizzo IPv6 per iniziare la comunicazione;
- è automaticamente configurato su ogni interfaccia, usa l'interface identifier (basato sul MAC address);
- presenta il seguente Formato: FE80:0:0:0: interface identifier (64bits).

L'indirizzo Unique Global Unicast è anch'esso limitato all'uso interno a una rete, non può essere ruotato su internet. Tale indirizzo è creato combinando un prefisso di rete (di lunghezza 7-bit), un Global-ID ottenuto mediante una particolare procedura, e concatenando il solito EUI-64 che identifica in modo univoco una certa interfaccia di rete. Questo meccanismo assicura una bassissima probabilità che un indirizzo IPv6 sia ripetuto. Riguardo al concetto di Anycast possiamo sottolineare che è limitato ai router, se si invia un pacchetto con indirizzo Anycast, soltanto un router di un determinato gruppo risponderà a quel messaggio in dipendenza dal protocollo di outing. Nel caso di pacchetto inviato con indirizzo multicast, tutti in nodi sullo stesso link che hanno quell'indirizzo multicast vengono coinvolti L'indirizzo multicast secondo IPv6 è così costruito: FF<flags><scope>::<group id>, dove:

- flag = 0 permanente / 1 temporaneo;
- scope: node (1), link (2), site (5), organization (8), global (E);
- group ID: identifica un gruppo multicast in un dato scope.

2.4.1 Il protocollo ICMPv6

Insieme alla nascita di IPv6 ha subito delle importanti modifiche anche il protocollo ICMP, utilizzato come strumento di diagnostica della rete. Associato ad IPv6 vi è la nuova versione di ICMPv6, osserviamo le novità introdotte da quest'ultimo. Ogni pacchetto ICMPv6 è così costituito:

- campo Type (8-bit): indica il tipo del messaggio;
- campo Code(8-bit): utilizzato per distinguere più in dettaglio la tipologia dei messaggi;
- campo di Checksum(16-bit): questo campo è stato introdotto perché nell'header del pacchetto IPv6 non vi è un campo di checksum;
- campo a lunghezza variabile.

ICMPv6 distingue i messaggi in due macro tipologie, ovvero messaggi di errore e messaggi informativi. Tra i messaggi informativi sono di significativa importanza i messaggi RA(Router Advertisement), RS(Router Solicitation), NA(Neighbor Advertisement) ed NS(Neighbor Solicitation). Tali messaggi sono utilizzati dal protocollo Neighbor Discovery che è un'assoluta novità introdotta da IPv6 e si presta bene al contesto delle reti di Smart-Objects.

2.4.2 Il protocollo μIPv6

Lo stack IP nei classici PC è parte integrante del sistema operativo. Vista la considerevole quantità di memoria di cui ha bisogno un tradizionale stack IP per funzionare, verrebbe da dire che nel contesto degli Smart-Objects non potrebbe essere utilizzato, data la scarsità di memoria di cui dispongono questi oggetti, ma in realtà esistono delle implementazioni leggere dello stack IP come μIP che andremo a vedere nel dettaglio e che è alla base del presente lavoro. μIP è quindi un'implementazione del tradizionale protocollo IP, realizzata per gli Smart-Objects o per dispositivi emebedded. La prima versione di μIP è nata nel 2001 e rilasciata con licenza Open. μIP implementa i tradizionali protocolli della suite IP di livello rete e trasporto come IP, ICMP, UDP e TCP; μP è il primo stack per dispositivi emebedded che implementa anche il protocollo TCP. TCP è implementato in uIP in modo piuttosto semplificato, ma è perfettamente compatibile con lo standard. μIP è nato per la versione quattro del protocollo IP, ma nel 2008 CISCO ha realizzato una versione perfettamente compatibile con le specifiche della versione IPv6. Possiamo dire che μIP fa tre cose fondamentali ed ha il seguente modo di operare (Fig. 2.4):

- appena un pacchetto viene ricevuto dal dispositivo di comunicazione viene chiamata in causa la funzionalità "input process" che si occupa di analizzare l'header del pacchetto IP. Tale funzione verifica il contenuto del pacchetto e lo inoltra ai livelli superiori;
- la funzionalità di "output process" viene richiamata da μIP quando l'applicazione ha prodotto dati da inviare. È μIP stesso che stabilisce quando dei dati in uscita possono essere passati al dispositivo di comunicazione. Prima di passare questi dati prodotti dall'applicazione la funzione di output aggiunge gli opportuni header al pacchetto;
- μIP ha una funzionalità che viene ripetuta periodicamente (periodic process) che ha lo scopo di verificare se qualche applicazione deve ritrasmettere dei dati o se qualche connessione deve essere chiusa;
- inoltro e routing dei pacchetti sono gestiti a parte.

La funzione input process (Fig. 2.5) inizia ad analizzare il pacchetto ricevuto a partire dall'header e ne va a verificare i diversi campi. Come prima cosa controlla che quello che ha ricevuto sia un pacchetto IP, verifica poi la versione del pacchetto e quindi comprende se è una versione che tale stack è in grado di gestire. Dopo va a controllare che la lunghezza riportata nel campo payload lenght dell'header coincida con la lunghezza del pacchetto presente nel buffer, se così non fosse il pacchetto verrebbe scartato. La ricostruzione di un pacchetto frammentato avviene in ordine al processing dell'extension-header.

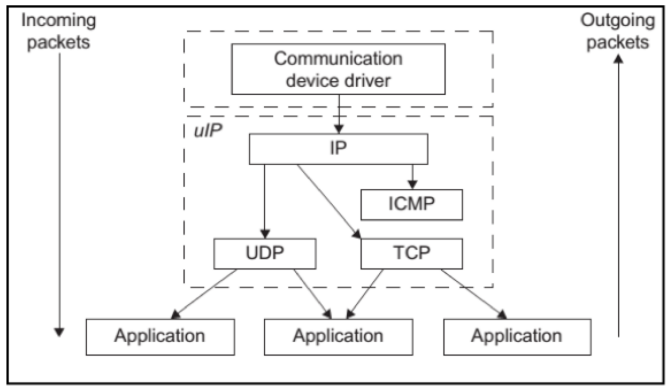

Figura 2.4: Principio di funzionamento dello stack μIP e flusso dei pacchetti in entrata e in uscita

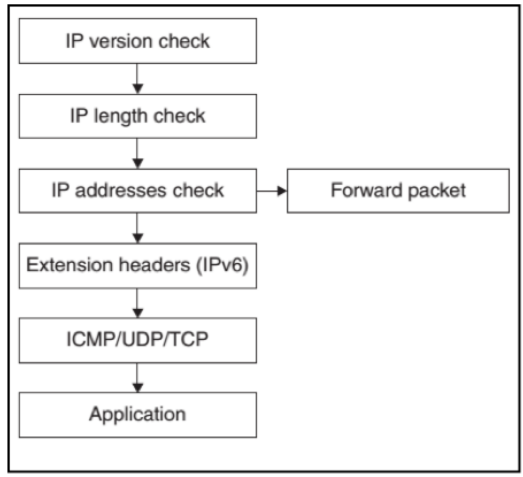

Figura 2.5: Diagramma di flusso dell'input process di μIP

A questo punto l'input process si porta ad analizzare il contenuto del campo Destination-Address dell'header del pacchetto IP, come prima cosa si preoccupa di verificare che l'indirizzo contenuto in tale campo sia un indirizzo IP valido altrimenti scarta il pacchetto, appurata la validità dell'indirizzo va a verificare che questo coincida con almeno un indirizzo locale. Se non si ha nessun matching con gli indirizzi locali il pacchetto viene passato all'unità di forwarding che si preoccuperà di inoltrare il pacchetto al vicino di next-hop o al diretto interessato.

Riguardo l'output process possiamo rilevare come tale processo venga avviato dallo stack stesso, vale a dire che un processo di livello superiore per inviare i dati deve attendere di essere richiamato da μIP. Quando l'applicazione è

richiamata da μIP può decidere se inviare i dati o no, nel caso in cui i dati siano inviati μIP si preoccupa di aggiungere gli opportuni header e quindi di far arrivare il pacchetto al driver del dispositivo di comunicazione che si occupa di inviarlo nel mezzo trasmissivo. Un protocollo di livello applicativo come TCP invia i dati solo quando è invocato da μIP, mentre un protocollo come UDP può inviare i dati quando vuole senza aspettare che venga richiamato da μIP.

μIP svolge tre compiti fondamentali e ognuno di questi è svolto da un processo dedicato, quello che ci rimane da descrivere è il periodic process. Il periodic process è una sorta di controllore che a istanti temporali specifici svolge delle particolari funzioni. Il periodic process può essere richiamato una o due volte al secondo, dipende dalla particolare implementazione di μIP, e si occupa di verificare che non vi siano connessioni in time-out, controlla il buffer dei pacchetti frammentati che sono in attesa di essere ricostruiti. Se un pacchetto rimane nel buffer dei pacchetti frammentanti per più di trenta secondi è scartato perché vuol dire che qualche frammento è andato perso. Tale processo periodico si preoccupa di controllare, se vi sono, le connessioni TCP attive e per ognuna di esse verifica se vi è una richiesta di ritrasmissione pendente e nel caso in cui vi sia richiama l'applicazione per autorizzarne la ritrasmissione dei dati. Per risparmiare memoria μIP sceglie di far ricreare il dato da rinviare direttamente all'applicazione interessata piuttosto che memorizzarlo in un buffer. Il processo periodico controlla anche le connessioni in time-out, cioè quelle connessioni che hanno inviato una serie di pacchetti e non hanno mai ricevuto una risposta, tali connessioni vengono chiuse.

μIP si occupa anche di realizzare il packet-forwarding quando l'indirizzo IP contenuto nel campo Destination-Address dell'header non matcha con nessuno degli indirizzi locali del nodo su cui è in esecuzione μIP. Il meccanismo di forwarding ha luogo se μIP è nella modalità router. Il meccanismo di forwarding chiede al modulo di routing di confrontare l'indirizzo di destinazione del pacchetto con gli indirizzi contenuti nelle tabelle di routing, il modulo di routing restituisce al modulo di forwarding l'indirizzo del next-hop neighbor cui deve essere inoltrato il pacchetto. Altra caratteristica interessante di μIP è il modo in cui gestisce la memoria, contrariamente ad un normale stack IP in μIP la gestione della memoria è relativamente semplice e inoltre viene utilizzato un unico buffer per i dati in entrata e in uscita allo scopo di risparmiare spazio. Se un nuovo pacchetto arriva mentre μIP ne sta processando uno, quello più recente viene mantenuto nel buffer del dispositivo di comunicazione. La lunghezza del buffer μIP è tale da consentire la memorizzazione di un pacchetto di massima dimensione.

Un'ultima osservazione va fatta riguardo alle API (Application Program Interface) fornite da μIP. Le API definiscono il modo in cui un programma applicativo interagisce con lo stack IP, nel caso di μIP si hanno delle API event-

driven mentre in uno stack IP tradizionale si hanno delle API multithreading. Il concetto delle API event-driven serve solo a ridurre la quantità di memoria necessaria, inoltre è stato dimostrato come delle API event-driven ben realizzate sono esattamente efficenti quanto delle API multi-threading. Per limitare la quantità di memoria necessaria per implementare μIP si fanno delle scelte di compromesso, ad esempio da μIP sono state rimosse tutte quelle funzionalità che in IP permettono di ottimizzare i servizi ma nonostante queste scelte μIP risulta sempre perfettamente compatibile con lo standard IP.

2.5 Routing Protocol for Low Power and Lossy Network

Un altro livello fondamentale da gestire, tenendo sempre presenti i limiti in termini di risorse di calcolo e di energia a disposizione in un nodo sensore per l'IoT è quello del protocollo di routing da utilizzare per l'instradamento dei pacchetti IP. Il routing all'interno delle classiche reti IP alimentate da rete viene gestito da protocolli come RIP [18] e OSPF [19], ma tali soluzioni sono consolidate, stabili ed utilizzabili in reti alimentate, mentre sono troppo onerose, sia dal lato computazionale che da quello energetico, per reti di Smart Object. Nelle WSN si opera con un mezzo trasmissivo che è molto variabile e inaffidabile, quindi è necessario che il protocollo di routing sia in grado di gestire questi tipi di problemi.

Routing Protocol for Low Power and Lossy Network (RPL) [20], è stato definito dalla IETF (Internet Enginering Task Force) tramite un apposito gruppo di lavoro creato in seguito alla crescente necessità legata all'imponente sviluppo delle WSN e degli Smart-Objects. WSN e Smart-Objects lavorano su collegamenti per lo più wireless a bassa potenza caratterizzati inoltre da elevata variabilità, quindi inaffidabili. Oltre alle caratteristiche del collegamento, nella definizione del protocollo, si è dovuto tener conto anche del fatto che i nodi che costituiscono le WSN sono tutti caratterizzati da scarsità di risorse di memoria, da traffico dati tipicamente piuttosto limitato e necessità di ottimizzare le risorse di energia che richiede la limitazione del traffico di controllo per l'instaurazione ed il mantenimento delle connessioni. Sottolineiamo che RPL è un protocollo di routing IPv6 basto su distanze vettoriali e non un classico protocollo link-state; questo aspetto permette di risparmiare molta memoria.

Vediamo qui di seguito come opera RPL. RPL costruisce un DODAG ovvero Destination Oriented Directed Acyclic Graph, ogni percorso nel grafo è costruito da un nodo della rete verso il nodo radice (comunemente chiamato root). Il grafo costruito non rappresenta una classica topologia ad albero, contrariamente ad essa nel DODAG di RPL ogni nodo presenta percorsi alternativi verso gli

altri nodi della rete e verso il root. Questa scelta è utile a ovviare al problema dell'inaffidabilità dei collegamenti wireless a bassa potenza.

RPL definisce tre nuove tipologie di messaggi di controllo ICMPv6 che utilizza per la costruzione del grafo. I nuovi messaggi definiti da RPL sono il DIO (DODAG Information Object), DAO (DODAG Advertisement Object) e DIS (DODAG Solicitation Object); descriveremo questi messaggi nel dettaglio in seguito. Nella rete che si vuol creare, vi deve essere uno o più nodi configurati dall'amministratore come root, è il root che inizia la procedura di costruzione del DODAG. Se un nodo "scopre" più nodi vicini appartenenti al DODAG esso utilizza diverse regole per scegliere se connettersi alla rete e a chi connettersi sulla base delle informazioni ricevute dai messaggi scambiati, queste regole prendono il nome di metriche. Non appena un nodo si è connesso al DODAG è disponibile un percorso di routing dal nodo verso il root. RPL utilizza la terminologia upward direction per indicare i percorsi che vanno dai nodi periferici, chiamati foglia, al root e downward direction per indicare i percorsi che vanno dal root ai nodi periferici. I percorsi nella "up" direction si determinano in automatico mano a mano che il DODAG si costruisce, mentre i percorsi nella "down" direction utilizzano i DAO message per essere determinati. Un nodo foglia invia un DAO per pubblicizzare la raggiungibilità del prefisso al root, tale messaggio ha la capacità di memorizzare i nodi attraversati così il root riceve un DAO che contiene delle informazioni aggregate sulla raggiungibilità di un certo prefisso. RPL introduce una particolare notazione per indicare i vari nodi nella rete, si farà spesso utilizzo della terminologia parent, child e sibiling. Il parent è il nodo genitore caratterizzato dall'avere un più basso valore di rank rispetto al nodo figlio considerato. Con la terminologia sibiling si intendono indicare dei nodi "fratelli", caratterizzati quindi da uno stesso valore di rank. Da questa descrizione si comprende come RPL fornisca un supporto al traffico MP2P (con le rotte indicate come "up"), al traffico P2MP (con le rotte indicate come "down") e al traffico P2P. Per capire come avviene il traffico P2P basta osservare questo esempio: supponiamo che un nodo A voglia inviare l'informazione a un nodo B che non è direttamente raggiungibile, ne consegue che A invierà l'informazione destinata a B al suo set di genitori. I genitori controllano che il nodo di destinazione sia contenuto tra i lori figli, in tal caso il pacchetto è inoltrato lungo una "down" direction, altrimenti è inoltrato nella "up" direction in modo che il pacchetto informativo giunga a un antenato comune, sarà a tal punto che il pacchetto sarà instradato lungo una "down" direction per raggiungere quindi la destinazione finale. Questo può continuare a ritroso fino a raggiungere il root se i nodi interessati nella comunicazione non hanno antenati comuni ai più bassi livelli del DODAG.

RPL utilizza la terminologia "floating" e "grounded" per indicare che il DODAG sia legato a un nodo root disconnesso da altre reti (floating) o sia legato a

un nodo root a sua volta connesso a un'altra rete (groundend) come può essere la rete internet. Una volta che il DODAG è stato costruito e le tabelle di routing sono state popolate, il protocollo di routing è pienamente operativo, nel caso in cui dei collegamenti o nodi vengano a mancare vi sono dei meccanismi di autoriparazione della rete locali e globali. Tali meccanismi prendono il nome di "local_repair" e "global_repair", agiscono secondo delle specifiche regole, la funzionalità di "local_repair" è quella di realizzare una riparazione locale della rete, coinvolgendo solo i nodi vicini al nodo che la invoca. La funzionalità di "global_repair" invece, viene impartita dal root quindi coinvolge tutta la rete, non può essere considerata un meccanismo di riparazione ma come un meccanismo di ottimizzazione della rete. RPL supporta anche il concetto di istanza, identificata mediante la parola chiave di RPLIstanceID. Il concetto di istanza permette di definire diverse topologie di routing sopra lo stesso grafo, attraverso l'impiego di diverse funzioni obbiettivo si vengono a costruire differenti topologie di routing, ognuna ottimizzata secondo la funzione obbiettivo utilizzate per costruirla.

Facciamo una piccola parentesi sulla struttura dei messaggi ICMPv6 definiti da RPL e utilizzati per la costruzione del DODAG. I messaggi DIO sono inviati dai nodi per pubblicizzare le informazioni sul DODAG e sulle sue caratteristiche. I DIO sono utilizzati per scoprire un DODAG, per la sua formazione e il suo mantenimento. Vediamo i principali campi di cui è composto un DIO message:

- grounded (G): indica se il DODAG è grounded o floating;
- destination Advertisement Trigger (T): è utilizzato per triggherare il refresh delle downward routes;
- destination Advertisement Stored (S): è utilizzato per indicare se un nodo genitore, diverso dal root, mantiene in memoria una tabella di routing con delle voci apprese dai messaggi DAO;
- destination advertisement supported (A): quando settato abilita la pubblicizzazione dei prefissi nel DODAG da parte del root;
- DODAGPreference (Prf): campo a 3-bit, per indicare le preferenze del root;
- DODAGSequenceNumber: è un numero che indica le iterazioni di costruzione del DODAG, numero che viene gestito solo dal root;
- RPLInstanceID: definisce l'istanza del DODAG, è utile nel caso in cui si vogliano definire diverse topologie di routing sopra lo stesso DODAG;

- Destination Advertisement Trigger Sequence Number (DTSN): è un intero a 8-bit impostato dal nodo che invia il DIO;
- DODAGID è un numero a 128-bit, corrisponde con l'indirizzo IPv6 del root e identifica univocamente il DODAG considerato;
- DODAG Rank: è il rango del nodo che invia il messaggio DIO. Il rango determina la posizione di un nodo nel DODAG, è utilizzato in primo luogo come meccanismo per il loop-avoidance. È un intero a 16-bit, si noti che un nodo non può avere rango inferiore dei suoi genitori all'interno di un determinato DODAG.

I messaggi DAO sono utilizzati per propagare informazioni di destinazione in salita lungo il DODAG, cioè dai nodi foglia al root. Tali informazioni servono a popolare le tabelle di routing e per fornire supporto al traffico (nella down direction) P2MP e P2P. Vediamo come è costituito un pacchetto DAO:

- DAO sequence: è un contatore incrementato dal nodo ogni volta che invia un nuovo messaggio DAO;
- RPLInstanceID: la topologia di routing appresa da un messaggio DIO;
- DAO rank: indica il rank del nodo cui appartiene il DAO;
- DAO lifetime: espresso in secondi, indica il tempo di vita di un certo prefisso;
- route tag: un intero a 8-bit che viene utilizzato per etichettare le rotte critiche;
- Destination Prefix: è un campo di lunghezza del prefisso che contiene il numero di bit validi del prefisso;
- Reverse Route Stack: contiene un numero che conteggia gli indirizzi IPv6 che all'interno della rete non memorizzano delle tabelle di routing.

L'altra tipologia di messaggio ICMPv6 introdotta da RPL è il DIS che è molto simile al messaggio IPv6 chiamato RS (Router Solicitation), è utilizzato per scoprire i DODAGs nelle vicinanze e sollecitare l'invio dei DIO da parte dei nodi vicini. Il messaggio DIS non ha ulteriore contenuto informativo. Dopo questa panoramica descrittiva vediamo come avviene la costruzione del DODAG. Il root inizia a inviare informazioni riguardo il DODAG attraverso i messaggi DIO, i nodi nelle immediate vicinanze ricevono e processano i DIO e attraverso delle regole definite dalla funzione obbiettivo decideranno se unirsi e dove unirsi al DODAG. Non appena un nodo si unisce alla rete ha un percorso verso il nodo root. Ogni nodo si calcola il proprio rank che definisce la posizione del nodo nel

grafo. Se un nodo è configurato come router inizierà ad inviare informazioni riguardo il grafo ai nodi vicini inviando messaggi DIO con le nuove informazioni. Se un nodo è configurato come foglia si limita a ricevere il DIO e si unisce alla rete. I nodi router vicini ripeteranno questa procedura realizzando una selezione dei genitori, aggiunta delle rotte e pubblicizzando le informazioni del grafo attraverso i messaggi DIO. Questo effetto a onda si propagherà fino ai nodi foglia dove termina. Attraverso questo processo ogni nodo ha una rotta per il routing verso i suoi genitori, i nodi foglia possono inviare un messaggio verso il root in qualsiasi parte del grafo essi si trovino semplicemente inoltrando il loro messaggio al genitore più immediato. Questo è il routing che va sotto il nome di MP2P dove ogni nodo può raggiungere il root, o anche definito UPWARD routing. Vi sarà la necessità di gestire un traffico in discesa cioè diretto dal root o dai nodi intermedi verso i nodi foglia, quindi dobbiamo avere informazioni di routing per il forwarding dei pacchetti nella down direction. Tali informazioni per la gestione del DONWARD routing sono ottenute dai messaggi DAO utilizzati per pubblicizzare la raggiungibilità di un prefisso verso i nodi foglia. Appena un nodo si unisce al grafo invia un messaggio DAO al set dei suoi genitori, ogni nodo che riceve un DAO processa l'informazione del prefisso e aggiunge una voce nella tabella di routing, questo processo continua fino a che l'informazione del prefisso giunge al root. Quando l'informazione di un determinato prefisso raggiunge il root, questo avrà a disposizione un percorso completo verso quel prefisso. Questa modalità di funzionamento prende il nome di storing-mode di RPL dove tutti i nodi intermedi memorizzano delle tabelle di routing. RPL ha anche un altro modo di funzionare chiamato non-storing-mode dove le informazioni di routing sono tutte contenute nel nodo root.

2.5.1 Metriche per la scelta dei parent

Le metriche di rete sono uno strumento utilizzato dai nodi per prendere decisioni di instradamento e vengono utilizzate per determinare quale percorso sia il migliore [21]. Esse si basano principalmente su informazioni come: lunghezza del percorso, larghezza di banda, carico, hop, costo del percorso, ritardi, unità massima di trasmissione (MTU), affidabilità e qualità del collegamento. Esistono numerose tipologie di metriche di rete, durante questa tesi ne sono state analizzate 4:

- ETX;
- Hop Count;
- Node Energy;
- Link Color.

ETX

La metrica ETX indica una stima delle trasmissioni attese prima che il pacchetto venga correttamente recapitato a destinazione, è una misura della qualità di un percorso tra due nodi in una rete wireless. L' ETX offre un valore (che può non essere un numero intero) calcolato secondo la formula 2.1:

$$ETX = \frac{1}{1 - e_{pt}} \tag{2.1}$$

Tale risultato è ampiamente variabile a causa delle caratteristiche del mezzo trasmissivo, e_{pt} è la probabilità di errore di pacchetto. L'oggetto ETX è formato da sub-oggetti ETX, di lunghezza fissa a 16 bit, e deve comprenderne almeno uno.

L'oggetto ETX può essere usato o come vincolo o come metrica di percorso, nel primo caso esso presenta un header comune che indica il valore relativo al vincolo che rappresenta il numero massimo di trasmissioni che un nodo aspetta di fare verso una destinazione per consegnare con successo un pacchetto. Nel secondo caso l'oggetto ETX è usato come un valore additivo della metrica dove il valore viene aggiornato lungo il percorso per definire la qualità del percorso stesso. Quando un nodo riceve il valore additivo aggregato dell'ETX di percorso (calcolato come la somma di tutti gli ETX dei nodi attraversati nel collegamento), se seleziona quel nodo come suo genitore preferito, aggiunge l'ETX del collegamento locale fra lui e il suo genitore preferito.

Hop Count

La metrica Hop Count (HP) serve a conoscere il numero di nodi che vengono attraversati per raggiungere una destinazione. L'HP è espresso come un numero intero.

L'oggetto Hop Count può essere utilizzato come vincolo o come metrica. Quando viene utilizzato come vincolo, la radice DAG indica il numero massimo di hop che un percorso può attraversare, una volta raggiunto tale numero, nessun altro nodo può aderire a tale percorso. Quando viene utilizzato come metrica, ogni nodo raggiunto semplicemente incrementa il campo Hop Count. Si noti che il primo nodo lungo un percorso dove è inserita una metrica HP deve necessariamente impostare il valore Hop Count a 1, in modo da far partire il conteggio.

Node Energy

In una rete, delle volte, è possibile che si voglia evitare di utilizzare come router un nodo con bassa energia, in tal caso la metrica, in particolare la Node Energy, può selezionare un percorso alternativo per una parte del traffico in

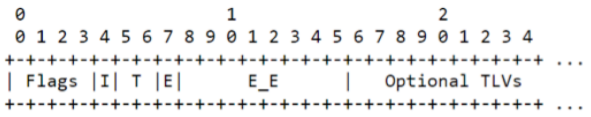

Figura 2.6: Oggetto della metrica Energy

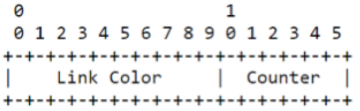

Figura 2.7: Formato dell'oggetto Link Color come metrica

modo da aumentare la durata di vita della rete. L'oggetto Node Energy (NE) viene utilizzato, infatti, per fornire informazioni relative all'energia dei nodi e può essere utilizzato oltre che come metrica anche come vincolo.

Nel sub-oggetto NE(Fig.2.6 il bit I è rilevante solo quando viene usato come vincolo ed, ad esempio, può essere selezionato al fine di far percorrere al collegamento solo i nodi alimentati dalla rete di alimentazione e di escludere quelli alimentati a batteria, in generale quando I è settato vuol dire che i nodi del tipo specificato nel bit T vanno inclusi nel collegamento, al contrario quando non è settato i nodi specificati vanno esclusi. Il bit T specifica il tipo di nodo: T=0 contrassegna un nodo alimentato dalla rete, T=1 un nodo a batteria, T=2 un nodo alimentato da energy scavenger (celle solari,energia termica, ecc...). Il bit E, quando è impostato, fornisce una percentuale stimata di energia residua sul nodo indicata nel campo E_E a 8 bit. Quando è azzerato, al contrario, la percentuale stimata di energia non viene rilevata. Quando il bit E è impostato, invece, per un vincolo, il campo E_E definisce una soglia per l'inserimento o l'esclusione di un nodo nel collegamento.

Link Color

L'oggetto Link Color (LC) è un registro a 10 bit utilizzato al fine di evitare collegamenti specifici e determinati tipi di traffico. L'oggetto LC può essere utilizzato sia come metrica che come vincolo, nel primo caso esso può essere soltanto registrato e può essere utilizzato da ogni nodo per selezionare il genitore basandosi su regole definite dall'utente, nel secondo caso specifica ai nodi quali collegamenti includere e quali escludere.

Quando l'oggetto Link Color è usato come metrica (Fig.2.7) il campo *Counter* tiene conto del numero dei collegamenti raggiunti che rispettano la specifica presente nel campo *Link Color*. Quando invece l'LC viene usato come vincolo (Fig.2.8 Il registro *Reserved* (5bit) è riservato e deve essere settato a 0 in tra-

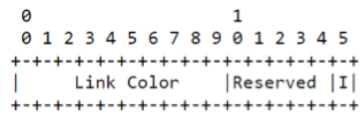

Figura 2.8: Formato dell'oggetto Link Color come vincolo

smissione ed ignorato in ricezione. Il bit *I* è rilevante solo quando l'LC viene utilizzato come un vincolo. Quando tale bit è impostato, questo indica che i collegamenti specificati nel campo *Link Color* devono essere inclusi. Quando è deselezionato, vuol dire che i collegamenti specificati nel medesimo campo devono essere esclusi. Almeno un sub-oggetto LC deve essere sempre presente all'interno di un oggetto LC.

2.6 TCP e reti di sensori wireless

TCP sta per Transfer Control Protocol, IP sta per Internet Protocol. In realtà il protocollo TCP/IP è una famiglia di protocolli, ognuno con una sua funzione particolare, TCP ed IP sono due protocolli importanti di questa famiglia. Alcuni di questi, come TCP, UDP, IP, ICMP e ARP sono di basso livello. Questo significa che essi lavorano vicino al livello fisico della rete infatti i loro compiti sono strettamente correlati con la trasmissione effettiva dei dati. La loro funzione è di fornire servizi ai protocolli superiori e alle applicazioni. Tutti i protocolli di livello superiore, sono specializzati nel compiere qualche servizio particolare. Fra i più comuni ed importanti abbiamo FTP, File Transfer Protocol, permette il trasferimento dei file, TELNET, network terminal protocol, permette di parlare con un computer remoto come se vi foste davanti, SMTP, Simple Mail Trasfer Protocol, serve a trasferire la posta elettronica, SNMP, Simple Network Management Protocol, per gestire e monitorare lo stato della rete.

I protocolli di livello superiore basano il loro lavoro sull'assunzione che i protocolli di basso livello siano in grado di comunicare in modo affidabile. TCP offre questa garanzia di affidabilità, ma il prezzo da pagare è la crescita di traffico di controllo senza contenuto informativo che causa un veloce esaurimento delle risorse energetiche, pertanto è stato implementato, in quando presente nello stack IP oggetto di questo lavoro, ma il suo utilizzo non è stato preso in considerazione viste le caratteristiche che non lo rendono idoneo a lavorare con nodi sensore wireless.

Innanzi tutto diciamo che TCP fornisce un servizio byte-stream. Questo termine indica che i dati da e per i livelli superiori vengono presentati e ricevuti come un unico flusso di byte, e non come pacchetti. Questo significa che è TCP,

e non i protocolli superiori (e quindi le applicazioni), a doversi preoccupare di preparare i pacchetti, con l'evidente vantaggio di avere una chiara distinzione dei ruoli: l'applicazione di alto livello pensa a preparare il messaggio, TCP pensa a come deve inviarlo. Un pacchetto TCP è detto segmento. Essenzialmente i compiti di TCP sono:

- suddividere i dati da spedire in tanti segmenti indipendenti e numerati;
- riassemblare i dati arrivati all'altro capo, presentandoli nuovamente come un flusso di byte;
- rispedire i datagrammi non arrivati o arrivati corrotti;
- riordinare i datagrammi se alcuni di essi non risultassero sequenziali;
- controllare il flusso attraverso il meccanismo delle finestre e dell'acknowledgement;
- multiplexing attraverso l'uso delle porte.

Ad ogni segmento, TCP aggiunge il suo header. Nell'header sono contenuti i seguenti campi:

- source port/destination port: (16 bit + 16 bit) quando si inizia ad inviare o a ricevere dati, qui vengono memorizzati due numeri di porta, uno al quale inviare i dati, uno sul quale i dati verranno spediti da parte dell'altro dispositivo. Sull'altra macchina, i numeri di porta sono uguali ma scambiati. Per ogni comunicazione full-duplex vengono aperte due connessioni, una in trasmissione e una in ricezione, ognuna con il suo numero di porta. Normalmente, quando si stabilisce una comunicazione generica, TCP genera un numero di porta casuale. Bisogna comunque ricordare che molte applicazioni definiscono numeri di porte standard (detti well known services, ad esempio 21 per FTP, 23 per Telnet, 53 per il servizio DNS, 80 per il Web Server;
- sequence number: (32 bit) contiene il numero necessario per sapere quale sia l'ordine dei segmenti e per sapere se qualcuno è andato perduto. Diversamente da come si potrebbe pensare, non viene usato il numero di segmento, ma il numero del primo byte di quel segmento. Quindi se ogni segmento contiene 1500 byte, il sequence number sarà $0 \ldots 15001 \ldots 3000 \ldots$ e non $0 \ldots 1 \ldots 2 \ldots$;
- acknowledgment number: (32 bit) viene usato per segnalare la ricezione di tutti i dati fino al numero di byte specificato meno uno, e dovrebbe essere uguale al valore del prossimo Sequence number che sarà ricevuto;

- window: (16 bit) nei segmenti con il flag ACK, esprime in byte l'ampiezza della finestra che il computer è in grado di ricevere;
- data offset: (4 bit) è il numero di parole a 32-bit dell'header TCP. Indica dove iniziano i dati;
- reserved: (6 bit) riservato ad usi futuri.
- flags: (1 bit per ogni flag) è composto dai seguenti campi:
  - Flag URG: (urgent) se attivo indica che il campo Urgent Pointer è valido e deve essere letto;
  - Flag ACK: (acknowledgement) se attivo indica che il campo Acknowledgement number è valido e deve essere letto;
  - Flag PSH: (push) usato per indicare che il segmento deve essere inviato immediatamente. Utile nei programmi interattivi come Telnet, dove vogliamo che i comandi digitati con la tastiera siano inoltrati quanto prima. Senza il Flag PSH, per motivi di efficienza il TCP accumula tutti i dati in un buffer interno, la cui spedizione viene ritardata fino a che non si sia completamente riempito;
  - Flag RST: (reset) usato per reinizializzare completamente la connessione TCP corrente;
  - Flag SYN: (synchronize) usato quando viene stabilita una sessione, indica che il ricevente dovrà leggere il campo Sequence number e sincronizzare il proprio con esso;
  - Flag FIN: (finish) indica che non ci sono altri dati da trasmettere. La connessione rimane comunque aperta in ricezione.
- checksum: (16 bit) un numero che serve per sapere se il datagramma corrente contiene errori nel campo dati;
- urgent pointer: (16 bit) indica che il ricevente deve iniziare a leggere il campo dati a partire dal numero di byte specificato. Viene usato se si inviano comandi che danno inizio ad eventi asincroni urgenti. Ad esempio il comando Control-C in una sessione Telnet;
- options: (lunghezza variabile) contiene varie opzioni di TCP (Maximum Segment Size, Window Scale, Sack Permitted, Sack, Time Stamp);
- padding: (lunghezza variabile) una serie di 0 inserita perché la lunghezza dell'header TCP sia un multiplo di 32 bit;
- data: il nostro campo dati.

2.6.1 Il meccanismo delle finestre scorrevoli in TCP

Senza ulteriori approfondimenti si potrebbe pensare che il lavoro del TCP consista semplicemente nel mandare un segmento, aspettare l'acknowledgement per quel pacchetto, mandare un altro pacchetto. Se l'Ack non arriva entro un certo intervallo di tempo ritrasmettere il primo segmento. Anche se funziona, questo modo di procedere è estremamente inefficiente poiché sfrutta solo una piccola parte della larghezza di banda disponibile. Un sistema molto più efficiente ed ugualmente affidabile, è il meccanismo a finestre. Il funzionamento è il seguente:

- il mittente invia una finestra di pacchetti TCP, da 1 a n, ma senza aspettare l'Ack, inoltre fa partire un timer per ognuno di essi. Il numero n è detto ampiezza della finestra;
- il ricevente manda un Ack per ogni pacchetto;
- il mittente sposta la finestra in avanti di un pacchetto per ogni Ack ricevuto nell'ordine. Cioè quando arriva l'Ack del segmento 1, la finestra viene spostata in modo da coprire i pacchetti da 2 a n+1 e viene trasmesso il segmento n+1;

Consideriamo ora qualche caso particolare: Se il pacchetto 2 non arrivasse a destinazione, la finestra non verrebbe spostata oltre il pacchetto 1. Il destinatario manderebbe gli Ack dei pacchetti 3, 4, 5... ma tutti uguali, cioè settati al valore 1, dato che è questo l'ultimo pacchetto valido, ricevuto nell'ordine di consegna. Ad un certo punto il timer per 2 scade e il pacchetto viene ritrasmesso. A questo punto però sorge una domanda: dobbiamo ritrasmettere anche 3, 4, 5...' Purtroppo non possiamo saperlo. Se mandiamo solo 2, ma anche 3, 4, 5... sono andati persi dovremo aspettare che scadano i timer di tutti questi altri segmenti. Alternativamente possiamo rimandare tutta la finestra. È comunque chiaro che nessuna soluzione sia priva di inefficienze, perché l'informazione del campo Ack non è sufficientemente espressiva: non dice nulla del frame ricevuto, dice solo qual è l'ultimo frame valido ricevuto nell'ordine di consegna. Altra caso particolare: il pacchetto 2 viene ricevuto correttamente, ma è l'Ack che viene perso. Semplicemente, il mittente riceverà prima o poi un Ack con valore 3, questo indica che tutti i pacchetti fino al terzo sono arrivati a destinazione, quindi anche il secondo. Dopo l'Ack 3 il mittente può spostare la finestra in avanti di 2 passi in una volta. La finestra ora coprirà i pacchetti da 4 a n+3. Nella realtà, per identificare i segmenti si usa il Sequence number non il numero di pacchetto. Inoltre l'ampiezza della finestra è variabile da parte del ricevente durante la connessione grazie al campo Window.

2.6.2 Stabilire una connessione TCP: l'handshake a tre vie

Quando due dispositivi utilizzano TCP devono innanzitutto creare una sessione. La procedura attraverso la quale la sessione viene stabilita si chiama "three way handshacking", o handshacking a tre vie. Se il computer client PC1 vuole connettersi al computer server PC2, succede questo:

1. PC1 manda a PC2 un segmento TCP attivando il flag SYN;
2. PC2 risponde a PC1 con i flag SYN e ACK attivi;
3. PC1 risponde a sua volta con il flag ACK.

Al primo passo, PC1 attiva nel primo segmento il flag SYN per indicare a PC2 che il campo Sequence number è valido e che quindi deve essere letto. Il valore settato da PC1 in quel campo è detto Sequence number iniziale, o ISN (Initial Sequence Number). PC2 risponde attivando il flag SYN ed indicando un proprio ISN, inoltre attiva l'ACK indicando l'ISN+1 di PC1. Infine PC1 attiva l'ACK indicando l'ISN+1 di PC2. A questo punto la comunicazione è stabilita e PC1 può iniziare ad inviare gli altri segmenti.

Un problema di questa tecnica è la vulnerabilità ad un famoso tipo di DoS (Denial of Service): il SYN-Flooding. In sostanza il client (PC1) richiede decine e decine di connessioni al server vittima (PC2) inondandolo di pacchetti SYN (inondare = to flood). PC2 risponde con tanti SYN-ACK, a cui però PC1 volutamente non risponde (in teoria dovrebbe completare gli handshake con un ACK per ogni connessione). PC2 dopo un pò smette di aspettare, ma quando i SYN ricevuti sono troppi, rimane pressoché paralizzato. Dato che esiste un limite massimo al numero di connessioni che si possono stabilire su PC2, un eventuale utente PC3 che si collega a PC2 non verrebbe accettato perché tutti i canali disponibili sono impegnati ad aspettare ACK che non arrivano.

2.6.3 Chiudere una connessione TCP: il saluto a quattro vie

Quando si vuole terminare una connessione, si usa una procedura di terminazione a quattro vie. Un computer manda all'altro un segmento con il flag FIN attivo. Il flag FIN chiude la connessione in una sola direzione. All'altro capo verrà spedito tutto ciò che deve ancora essere spedito, dopodiché anche questo invierà un segmento con il flag FIN. Ora che la comunicazione è chiusa in entrambi i sensi, la sessione può considerarsi terminata.

2.6.4 Gli stati di una connessione TCP

Una sessione TCP può essere in diversi stati. Ad esempio per visualizzare lo stato di tutte le connessioni in corso su di un computer è possibile usare il comando: netstat -na Gli stati possibili sono i seguenti:

- LISTEN: in attesa che qualcuno richieda una connessione;
- SYN-SENT: durante la creazione di una connessione, indica che è stato inviato un segmento con flag SYN attivo e si sta aspettando il segmento di risposta (quello con i flag SYN/ACK);
- SYN-RECEIVED: durante la creazione di una connessione, indica che è stato ricevuto un segmento SYN, è stato inviato in risposta il SYN/ACK e che ora si sta attendendo l'ACK che completa l'handshake;
- ESTABLISHED: stato raggiunto dopo che l'handshake è stato completato con successo. La connessione è ora aperta e si possono trasferire dati;
- FIN-WAIT1: in attesa di una richiesta di terminazione della sessione da parte del computer remoto o di un acknowledgement della richiesta di terminazione della connessione precedentemente stabilita;
- FIN-WAIT2: in attesa di una richiesta di terminazione della sessione da parte del computer remoto;
- CLOSE-WAIT: in attesa di una richiesta di terminazione della sessione da parte del computer locale;
- CLOSING: in attesa dell'acknowledgement alla richiesta di terminazione da parte del computer remoto;
- LAST-ACK: in attesa dell'acknowledgement della richiesta di terminazione della connessione che è stata precedentemente inviata al computer remoto, include un acknowledgement della sua richiesta di terminazione della connessione;
- TIME-WAIT: in attesa che passi abbastanza tempo in modo da essere sicuri che il computer remoto abbia ricevuto l'acknowledgement della sua richiesta di terminazione della connessione;
- CLOSED: quando la connessione è del tutto terminata.

2.6.5 Il controllo delle congestioni di TCP

Per evitare congestioni indesiderate sulla rete, TCP dispone di alcuni algoritmi di controllo.

Slow start: quando due computer iniziano a comunicare, questo algoritmo impedisce al mittente di partire subito con una finestra di massima ampiezza (suggerita nel campo window del ricevente, detto anche advertised window). Questo meccanismo è molto utile quando la rete che unisce due computer non è una LAN diretta, ma una serie di collegamenti lenti (router e collegamenti

WAN). Quando si stabilisce la comunicazione, abbiamo una finestra nel TCP del mittente detta congestion window, cioè finestra di congestione. All'inizio è fatta di un solo segmento, poi cresce esponenzialmente (2, 4, 8, 16...) mano a mano che arrivano gli ACK dal computer ricevente, finché non si raggiunge il valore del campo window del ricevente. Spesso però, prima di raggiungere il valore indicato dall'advertised window, si arriva ad punto in cui alcuni pacchetti iniziano ad essere scartati dai collegamenti lenti intermedi. Questo indica che è opportuno fermare la crescita della finestra di congestione. Risulta importante non confondere la congestion window con l'advertised window, quest'ultima è usata solo dal ricevente per indicare quanti segmenti per finestra esso è in grado di accettare nel buffer di ricezione, mentre la congestion window è gestita dal mittente in base al numero di ACK giunti indietro.

Congestion Avoidance: Congestion Avoidance e Slow Start sono due algoritmi mutualmente esclusivi, cioè non possono mai avvenire contemporaneamente, tuttavia esiste un forte legame tra di loro. Quando avviene una congestione e vengono persi segmenti durante lo Slow Start, viene salvata in una speciale variabile ssthresh (che sta per slow start threshold) la metà del valore della congestion window per cui abbiamo perso segmenti. La finestra viene poi riportata ad uno, dopodiché, quando il valore della congestion window ricresce diventando uguale a ssthresh, la crescita della finestra diventa lineare e non più esponenziale. Quindi, se durante una slow start abbiamo una congestione per una congestion window pari a 32 byte, salviamo in ssthresh il valore 16, rimettiamo la congestion buffer a 1, poi rifacciamo una nuova slow start fino a 16 (crescita esponenziale: 1, 2, 4, 8, 16) dopodiché l'algoritmo di Congestion Avoidance interviene sulla velocità della crescita della congestion window (crescita lineare: 17, 18, 19, 20,...).

Fast Retrasmit: come dicevamo nel paragrafo delle finestre scorrevoli, quando un pacchetto viene perso, gli ACK dei segmenti successivi arrivati a destinazione riportano come Acknowledgement number il valore del segmento mancante nell'ordine di consegna. In condizioni normali dovremmo aspettare lo scadere del timer per il segmento andato perduto prima di poterlo ritrasmettere. Questo perché forse il segmento incriminato non è veramente andato perduto, ma è semplicemente molto lento. L'algoritmo di Fast Retrasmit permette di scavalcare il timer del pacchetto sospetto quando sono già arrivati 3 ACK riportanti lo stesso valore nell'Acknowledgement number. Supponiamo, ad esempio, che il mittente spedisca i segmenti:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

ma per un problema di trasmissione il segmento n.4 viene perso. Al ricevente arriveranno:

1, 2, 3, 5, 6, 7, 8, 9, 10

Gli ACK number spediti indietro al mittente saranno:

1, 2, 3, 3, 3, 3, 3, 3, 3

La conclusione è che il segmento 4 è probabilmente andato perduto e che sono già arrivati altri 6 pacchetti dopo che è arrivato il 3. Con il Fast Retrasmit invece di ricevere un sacco di ACK tutti uguali aspettando che il timer di 4 scada, ritrasmettiamo il segmento 4 (valoreACKduplicati+1) al terzo ACK uguale.

Fast Recovery: per migliorare l'efficienza del Fast Retrasmit, quando ritrasmettiamo un segmento che probabilmente è andato perduto, Fast Recovery fa in modo che il valore della congestion window non venga ridotto proprio del tutto, e che non venga effettuata una Slow Start, bensì una Congestion Avoidance. Il valore della congestion window infatti non viene riportato a 1, ma a sshtresh + 3, dove ssthresh è ancora settato alla metà della congestion window per cui è avvenuta la congestione, e 3 è il numero di ACK duplicati, questo per aumentare la congestion window del numero di segmenti che sono nel buffer di ricezione del destinatario, inoltre per ogni ulteriore ACK uguale incrementa ancora la congestion window di uno, questo per aumentare la congestion window del numero di segmenti partiti. Se permesso dal nuovo valore della congestion window, viene trasmesso un nuovo pacchetto. Quando arriva il successivo ACK relativo a nuovi dati, setta la congestion window a ssthresh (la metà). Così entriamo in Congestion Avoidance.

2.7 UDP

L'UDP (User Datagram Protocol) è un protocollo non orientato alla connessione come il TCP e proprio per questo non ha tutti i meccanismi di controllo di flusso che rendono la trasmissione molto affidabile. La mancanza di messaggi aggiuntivi per il controllo di flusso lo rende perfetto per le reti di sensori wireless alimentate a batteria ed è ampiamente diffuso nelle applicazioni per IoT. Ogni datagramma UDP viene mandato in un singolo datagramma IP. In sostanza l'unica cosa che un datagramma UDP aggiunge al protocollo IP è la capacità di usare le porte. L'esempio classico è quando bisogna contattare un server DNS. Una richiesta di questo genere è molto corta, non abbiamo bisogno di usare il TCP. L'UDP non si divide in datagrammi e non tiene conto di ciò che viene mandato o di ciò che non è arrivato, eventuali meccanismi di Acknowledge possono essere implementati a livello applicativo. L'header UDP è composto dai campi: porta sorgente/porta destinataria, equivalenti alle porte dell'header TCP, un campo Message Lenght che indica la lunghezza del datagramma incluso l'header, un campo di checksum di 16 bit ed infine il campo dati.

2.8 I protocolli applicativi

Il livello Applicazione fornisce servizi all'utente. La comunicazione è realizzata per mezzo di una connessione logica: i livelli applicazione nei due lati della comunicazione agiscono come se esistesse un collegamento diretto tra due identità logiche attraverso il quale poter inviare e ricevere messaggi. Un programma che vuole comunicare con un altro programma, ha bisogno di un insieme di istruzioni per gestire i primi quattro livelli dello stack TCP/ IP (aprire la connessione, inviare/ricevere dati e chiudere la connessione). Un insieme di istruzioni di questo tipo viene chiamato API (Application Programming Interface) o socket . Il livello di applicazione è l'unico che fornisce servizi agli utenti di Internet, la sua flessibilità consente di aggiungere nuovi protocolli con estrema facilità, per cui oggi non è più possibile indicare il numero dei protocolli esistenti poiché ne vengono costantemente aggiunti di nuovi. È importante distinguere fra applicazioni della rete e protocolli dello strato dell'applicazione. Un protocollo dello strato dell'applicazione è solo un pezzo di un'applicazione della rete. Nel nostro caso, il Web è un'applicazione della rete che permette agli utenti di ottenere a richiesta dai server Web i dati rilevati dai sensori. L'applicazione Web è costituita da molti componenti, che comprendono gli standard per i formati dei documenti, i browser del Web, i server del Web e un protocollo dello strato dell'applicazione. Il protocollo dello strato dell'applicazione Web definisce come i messaggi vengono passati fra browser e server Web e come i processi delle applicazioni, che funzionano su differenti terminali, si scambiano i messaggi. In particolare:

- i tipi di messaggi scambiati, per esempio, messaggi di richiesta e messaggi di risposta;
- la sintassi dei vari tipi di messaggio, per esempio i campi del messaggio e come questi campi vengono caratterizzati;
- la semantica dei campi, cioè il significato dell'informazione nei campi;
- le regole per determinare quando e come un processo invia o risponde a messaggi.

Un protocollo dell'applicazione tipicamente ha due parti, un lato client e uno server. Il lato client all'interno di un terminale comunica con il lato server di un altro terminale. Nella presente trattazione saranno analizzati nel dettaglio due protocolli: l'HTTP (Hyper Text Transfer Protocol) ed il CoAP (Constrained Application Protocol). Il primo è molto comune e, forse il più utilizzato nel mondo di internet, il secondo nasce proprio per le reti di sensori e l'IoT, pertanto fondamentale nella presente trattazione. Entrambi utilizzano l'architettura REST (REpresentational State Transfer), definisce la nozione di

risorsa e mira a modellizzare tutte le interazioni client/server come uno scambio di rappresentazioni di risorse.

2.8.1 HTTP

L'HTTP costituisce il cuore del web. Nel caso dell'HTTP, un browser web implementa il lato client e un server web ne implementa il lato server. Come nel caso di quasi tutte le applicazioni, l'host che inizia la sessione è etichettato come client. Il client HTTP apre una connessione e manda un messaggio di richiesta a un server HTTP che rimanda un messaggio di risposta, contenente solitamente la risorsa richiesta. Da notare che il server invia gli oggetti richiesti ai client senza immagazzinare alcuna informazione di stato relativa al client. Se uno stesso client chiede lo stesso oggetto due volte nel giro di pochi secondi, il server lo rispedisce come se avesse completamente dimenticato ciò che ha appena fatto. Poiché un server HTTP non conserva le informazioni relative ai client, l'HTTP è detto protocollo senza stato (stateless protocol). Il formato dei messaggi di richiesta e di risposta è simile, entrambi consistono in:

- una linea iniziale;
- nessuna o più linee di intestazione;
- una linea vuota;
- un corpo del messaggio opzionale.

Una richiesta HTTP, che rappresenta il messaggio mandato da un client HTTP (browser) ad un server HTTP (server Web), comprende tre elementi di base:

- linea di richiesta;
- intestazioni http;
- corpo del messaggio.

Ogni richiesta HTTP incomincia sempre con una linea di richiesta. Questa linea di testo è composta da tre campi, separati da spazi:

- metodo;
- URI;
- versione HTTP.

Essa indica quindi il metodo che il client sta richiedendo, la risorsa alla quale applicare il metodo e la versione HTTP che si sta usando. Dopo che un server Web riceve una richiesta HTTP, intraprende le azioni necessarie a fornire la risorsa richiesta. Queste possono includere l'esecuzione di uno script CGI o altro. Fatto ciò, il server Web risponde al client con una risposta HTTP. Questa risposta è organizzata in modo simile ad una richiesta HTTP. L'unica differenza significativa è che le risposte cominciano con una linea di stato piuttosto che con una linea di richiesta. Quindi, come nel caso di una richiesta, una risposta HTTP comprende tre elementi di base:

- linea di stato;
- intestazioni HTTP;
- corpo del messaggio.

Ogni risposta HTTP incomincia sempre con una linea di stato. Essa è composta da tre campi, separati da spazi:

- versione http;
- codice di stato;
- messaggio di stato.

Essa include quindi la più alta versione che il server supporta, un codice di stato che indica il risultato della richiesta e un messaggio di stato corrispondente. A differenza di un messaggio di richiesta dove la prima linea è quella di richiesta, in un messaggio di risposta vi è la linea di stato. Tutte le linee successive sono le intestazioni del messaggio terminate da una linea vuota (CRLF). Le intestazioni HTTP includono informazioni di supporto che possono aiutare a spiegare in modo più chiaro la risposta del server Web. Ci sono tre tipi di intestazioni che possono apparire in una risposta:

- intestazioni generali;
- intestazioni della risposta;
- intestazioni del corpo dell'entità.

Il corpo dell'entità è il nucleo del messaggio. In esso c'è la risorsa richiesta e rimandata al client (questo è l'uso più comune del corpo del messaggio), o testo che spiega al client se si è verificato un errore per cui l'operazione richiesta non può essere effettuata con successo. In un messaggio di richiesta invece rappresenta la parte dove si trovano i dati che l'utente ha immesso (ad esempio in una form) o file di upload da mandare al server. Uno dei più

importanti attributi di una richiesta HTTP è il metodo di richiesta. Esso indica l'intenzione della richiesta del client. Sebbene molti metodi siano usati raramente nella pratica, sono tutti importanti per scopi diversi. In http 1.1 sono otto i metodi di richiesta:

- GET;
- POST;
- PUT;
- DELETE;
- HEAD;
- TRACE;
- OPTIONS;
- CONNECT.

La versione 1.0 dell'HTTP include due metodi, LINK e UNLINK, che non esistono nella versione 1.1. Questi metodi, che non erano tra l'altro largamente supportati da browser o server Web, permettevano a un client HTTP di modificare le informazioni su una risorsa esistente senza cambiare la risorsa stessa. Mentre http 0.9 supporta solo un metodo di richiesta, il più comune GET, http 1.0 specifica tre metodi (GET, HEAD e POST), sebbene altri quattro sono implementati da alcuni server e client. Il supporto per questi altri quattro metodi (PUT, DELETE, LINK e UNLINK) è inconsistente e per lo più indefinito, anche se ognuno di essi è brevemente citato nell'appendice D dell'RFC 1945.

2.8.2 CoAP

Uno degli obiettivi principali del protocollo CoAP è realizzare un protocollo WEB adatto alle esigenze di dispositivi con risorse limitate in termini computazionali ed energetiche. Il modo con cui si vuole realizzare questo protocollo non consiste in una semplice compressione del protocollo HTTP, quanto piuttosto nell'implementazione di un sottoinsieme delle funzionalità offerte dall'architettura REST in comune con HTTP ottimizzando queste nell'ottica delle applicazioni di comunicazione M2M [22]. Il lavoro di standardizzazione del protocollo è opera del Constrained RESTful Environments (CoRE) Working Group IETF. In più il protocollo CoAP per essere veramente più leggero si appoggia ad UDP anziché a TCP: deve avere quindi un sublayer tra il livello trasporto e il livello applicazione che si occupa di garantire un minimo di affidabilità della trasmissione del messaggio in modo da colmare le lacune di UDP.

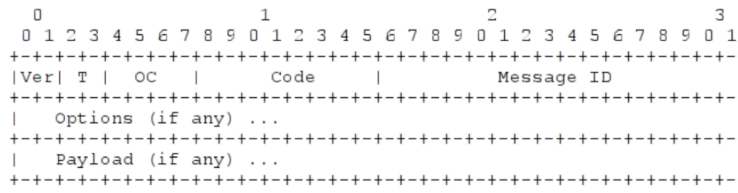

Figura 2.9: Struttura di un pacchetto CoAP

Caratteristiche

Le principali caratteristiche che il protocollo CoAP presenta sono:

- protocollo web per nodi di rete con risorse limitate adatto a comunicazioni M2M;
- trasporto su protocollo a datagramma come UDP con affidabilità opzionale;
- scambio asincrono di messaggi;
- basso overhead e bassa complessità di parsing dell'header;
- supporto a risorse URI e ad informazioni content-type del payload;
- realizzazione in modo semplice d'intermediari (proxy);
- possibilità di memorizzare in cache risposte per ridurre tempi di risposta ed occupazione di banda;
- traducibilità nel protocollo privo di stati HTTP con possibilità di realizzare proxy per garantire a nodi HTTP l'accesso a risorse CoAP e viceversa.

Pacchetto CoAP

Viene mostrata in Fig. 2.9 la struttura di un pacchetto CoAP.
Si noti che l'header del pacchetto è solamente 4 byte, al quale possono essere aggiunte delle opzioni:

- i primi due bit sono la versione del protocollo. Attualmente l'unico valore possibile è 01;
- altri due bit servono ad indicare il tipo di pacchetto che si manda. Il pacchetto può essere di 4 tipi: confirmable, non confirmable, ack e reset;
- i successivi 4 bit formano l'option count, cioè il numero di opzioni che il pacchetto possiede alla fine dell'header. Può essere anche 0;

- i successivi 8 bit formano il code. A seconda del suo valore può assumere diversi significati e funzioni. Quando il suo valore è 0, significa che il pacchetto è vuoto. I valori da 1 a 31 sono riservati per i tipi di richiesta, i valori da 64 a 191 per i codici di risposta;
- i rimanenti 16 bit formano il message ID. È un numero che serve ad associare un pacchetto al suo ack, e che serve a identificare eventuali pacchetti duplicati;
- seguono gli spazi per le opzioni e per il payload, entrambi di dimensioni variabili.

Tipi di messaggio

CoAP implementa delle funzioni che garantiscono un minimo di affidabilità della trasmissione. Per questo motivo ogni pacchetto può essere di 4 categorie, tale identificativo è contenuto nel campo type. Di seguito i possibili tipi di messaggi:

- confirmable (CON). Un pacchetto confirmable richiede che il suo destinatario mandi un ack che testimoni l'avvenuta ricezione. In caso di perdita del pacchetto o dell'ack, il messaggio viene ritrasmesso, duplicando ogni volta il tempo tra una trasmissione e la seguente, e fino ad un massimo fissato di ritrasmissioni. Come valori consigliati dallo standard, il tempo di attesa tra la prima trasmissione e la seguente è di 2 secondi, mentre il massimo numero di ritrasmissioni è 4 (nelle ritrasmissioni non si conta la prima trasmissione). Ovviamente in caso di perdita dell'ack, al destinatario arriveranno pacchetti duplicati: in questo caso il message ID fa da discriminante. L'ack deve avere lo stesso message ID del pacchetto che l'ha generato;
- non-confirmable (NON). A differenza del precedente, un pacchetto non-confirmable non richiede l'invio di un ack che testimoni l'avvenuta ricezione. Viene usato quindi per trasmissioni che non richiedono nessuna affidabilità come ad esempio nell'invio continuato nel tempo delle rilevazioni di un sensore;
- acknowledge (ACK). La funzione dell'ack è già stata spiegata parlando dei pacchetti confirmable. Non è stato detto che l'ack può essere pieno, quindi contenere la risposta all'interrogazione precedente (seguendo il meccanismo del piggy-backing), oppure vuoto, per implementare le trasmissioni deferred;
- reset (RST). Viene inviato un messaggio di reset in risposta ad un pacchetto che per qualche motivo il server non è riuscito a processare. Un

pacchetto di reset deve essere vuoto e come l'ack deve avere lo stesso message ID del pacchetto che lo ha generato.

Quando viene inviata una request, il server genera una response. Ma il protocollo deve tener conto che sta lavorando con dispositivi limitati, quindi potrebbe succedere che il server non abbia subito la risposta pronta. Quindi, a differenza dell'Internet tradizionale in cui si considera solamente il caso di risposte immediate, nel caso di dispositivi constrained si considera anche il caso di risposte di tipo deferred. Nel caso in cui la request sia di tipo CON, è comunque necessario inviare un ack, altrimenti il client continua a ritrasmettere. È in questo caso che si usa un empty ack, che comunica quindi al client che il server ha ricevuto la richiesta, e che invierà al più presto la relativa risposta. Quando sarà pronta, essa sarà una nuova CON, che richiede quindi che il client invii un empty ack per l'avvenuta ricezione. Questa nuova comunicazione dovrà avere un message ID diverso. A causa della possibilità di risposte deferred, può servire un metodo per associare una richiesta ad una risposta. Il message ID non lo può fare perché deve cambiare ad ogni singola trasmissione. Si usa quindi l'opzione Token. Ognuno dei pacchetti che appartengono allo stesso scambio di dati devono avere lo stesso token. Non è definita la sintassi del token, che può essere qualsiasi; anche la lunghezza può variare da 1 a 8 byte.

Richiesta e risposta

Ogni pacchetto CoAP può essere una richiesta o una risposta a seconda del valore scritto nel campo Code. Per ora CoAP supporta solo 4 tipi di richiesta, corrispondenti ai 4 tipi di richiesta basilari nel paradigma REST: GET (codice 1), PUT (codice 2), POST (codice 3), DELETE (codice 4). Le risposte definite sono molte di più, e sono raccolte nella Tab. 2.1. Si può notare anche in questo caso una forte corrispondenza con HTTP, ma in questo caso i codici di risposta sono nettamente divisi in due parti: la classe che occupa 3 bit e il dettaglio che occupa 5 bit. Nel caso in cui un proxy debba convertire da CoAP ad HTTP un codice CoAP non previsto in HTTP, nel pacchetto HTTP andrà messo il codice 502 (Bad Gateway).

Opzioni

La Tab. 2.2 mostra un elenco delle opzioni disponibili per i pacchetti CoAP. Ogni opzione possiede un codice che la identifica, un formato, una lunghezza, e alcune un valore di default. Sono divise in due categorie: critical ed elective. Mentre quando un'opzione elective non viene riconosciuta deve essere semplicemente ignorata, quando un'opzione critical non viene riconosciuta si deve

Tabella 2.1: Risposte ad una richiesta CoAP

Code	Descrizione	Corrispondente HTTP
64	2.00 OK	200 OK
65	2.01 Created	201 Created
66	2.02 Deleted	204 No Content
67	2.03 Valid	304 Not Modified
68	2.04 Changed	204 No Content
128	4.00 Bad Request	400 Bad Request
129	4.01 Unauthorized	400 Bad Request
130	4.02 Bad Option	400 Bad Request
131	4.03 Forbidden	403 Forbidden
132	4.04 Not Found	404 Not Found
133	4.05 Method Not Allowed	405 Method Not Allowed
141	4.13 Request Entity Too Large	413 Request Entity Too Large
143	4.15 Unsupported Media Type	415 Unsupported Media Type
160	5.00 Internal Server Error	500 Internal Server Error
161	5.01 Not Implemented	501 Not Implemented
162	5.02 Bad Gateway	502 Bad Gateway
163	5.03 Service Unavailable	503 Service Unavailable
164	5.04 Gateway Timeout	504 Gateway Timeout
165	5.05 Proxying Not Supported	502 Bad Gateway

rispondere con un errore. Nel caso di una richiesta CON, la risposta deve essere "4.02 Bad Option"; nel caso di una risposta CON, si risponde con un RST, nel caso di un pacchetto NON, esso deve essere ignorato. Viene brevemente descritta la funzione di alcune delle opzioni sopra elencate selezionando quelle più utili:

- Content-Type. È un codice che identifica il contenuto del pacchetto. Lo standard associa ad alcuni codici un sottoinsieme dei tipi MIME che si prevede verranno usati più spesso nelle comunicazioni. Il valore di default è 0, che corrisponde a text/plain; charset=utf-8;
- Max-Age. Esprime il massimo numero di secondi in cui la risposta deve essere considerata fresh, cioè in cui un proxy può usare la risposta che ha in cache anziché interrogare il server. Il valore di default è 60 secondi;
- Token. Serve ad associare una richiesta ad una risposta, specie in caso di risposte deferred. Può essere di vari formati e di varie lunghezze;
- Uri-Host, Uri-Port, Uri-Path, Uri-Query. In un pacchetto CoAP l'URI della risorsa che si desidera viene inserito nel pacchetto già scomposto in host, porta, percorso e query. Inoltre ci possono essere più opzioni Uri-Path, poiché il percorso viene inserito nel pacchetto già scomposto nelle

varie subdirectories. Se l'URI contiene caratteri in notazione percentuale, essi devono essere decodificati prima di essere inseriti nel pacchetto (tranne nella sezione query). Ovviamente queste opzioni hanno senso di esistere solamente all'interno di una request. In questo caso Uri-Host non deve mai essere vuoto, come minimo deve contenere l'indirizzo IP dell'host; UriPort, se non specificato contiene la porta di default 5683, mentre le opzioni Uri-Path e Uri-Query possono non esserci. Le opzioni all'interno del pacchetto sono situate alla fine dell'header. Devono essere poste in ordine di codice. In Fig. 2.10 viene visualizzato il formato di una opzione del pacchetto CoAP;

- Il primo campo è l'Option Delta, occupa 4 bit. Contiene non il codice dell'opzione, ma la differenza tra il codice dell'attuale opzione e il codice della precedente (o zero se è la prima);
- Il secondo campo contiene la lunghezza dell'opzione in byte. Essendo lungo 4 bit, sono permesse solo lunghezze tra 0 e 14. Mettendo come lunghezza 15, il campo viene esteso ad altri 8 bit, con cui si possono quindi rappresentare i numeri da 15 a 270;
- L'ultimo campo contiene il valore dell'opzione. Può quindi essere di lunghezza variabile.

Tabella 2.2: Opzioni di un pacchetto CoAP

Codice CE	CE	Nome	Formato	Lunghezza	Default
1	C	Content-Type	uint	1-2 B	0
2	E	Max-Age	uint	0-4 B	60
3	C	Proxy-Uri	string	1-270 B	(nessuno)
4	E	Etag	opaque	1-4 B	(nessuno)
5	C	Uri-Host	string	1-270 B	(Vedere elenco)
6	E	Location-Path	string	1-270 B	(nessuno)
7	C	Uri-Port	uint	0-2 B	(Vedere elenco)
9	C	Uri-Path	string	1-270 B	(nessuno)
11	C	Token	opaque	1-8 B	(Vuoto)
15	C	Uri-Query	string	1-270 B	(nessuno)

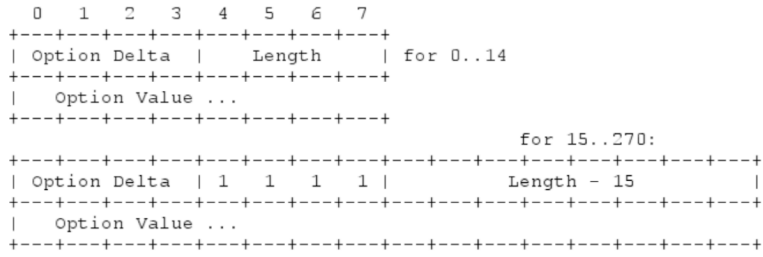

Figura 2.10: Struttura delle opzioni nel pacchetto CoAP

Capitolo 3

Materiali e Metodi

In questo capitolo vengono descritti gli strumenti utilizzati per lo sviluppo del lavoro di ricerca verso la creazione di un sistema completo capace di creare una rete mesh IPv6 per interconnettere dispositivi wireless nel mondo dell'IoT. Nel dettaglio analizzeremo le scelte fatte per il sistema operativo, per le tecniche e i tool di simulazione e testing della rete sviluppata e descriveremo la piattaforma hardware scelta per lo sviluppo delle applicazioni reali.

3.1 Il sistema operativo Contiki

Contiki è un sistema operativo open source pensato per reti di sistemi embedded come nel nostro caso. La prima versione di Contiki è stata rilasciata nel 2003.

Contiki fornisce dei meccanismi, utili a chi sviluppa Smart-Objects Network, come protocolli standard per la comunicazioni dei nodi con il mondo circostante, meccanismi per limitare il consumo di potenza da parte del dispositivo radio, librerie per la gestione della memoria e delle liste dinamiche. Contiki implementa un file system chiamato Coffe che permette ai programmi di utilizzare le memorie flash come dei tradizionali hard disk. È stato il primo sistema operativo per Smart-Objects che fornisce la comunicazione IP fra i nodi della rete attraverso il protocollo μIPv6 e TCP/IP sviluppato all'interno di un progetto iniziato nel 2008. Questo ne consente l'interazione con tutte le altre applicazioni basate su IP, come web services e internet-based services. Descriviamo ora le principali caratteristiche del sistema operativo e gli strumenti che esso fornisce a chi sviluppa applicazioni destinate all'ambito dell'IoT.

- PROCESSI: i programmi applicativi in Contiki possono essere eseguiti in due contesti differenti indicati rispettivamente con i nomi di Cooperative e Preemptive (Fig. 3.1). Il codice che viene eseguito nel contesto Cooperative è eseguito in modo sequenziale ovvero nel rispetto degli altri programmi applicativi che devono essere eseguiti nel medesimo contest. Una applicazione che viene eseguita nel contest cooperative monopolizza

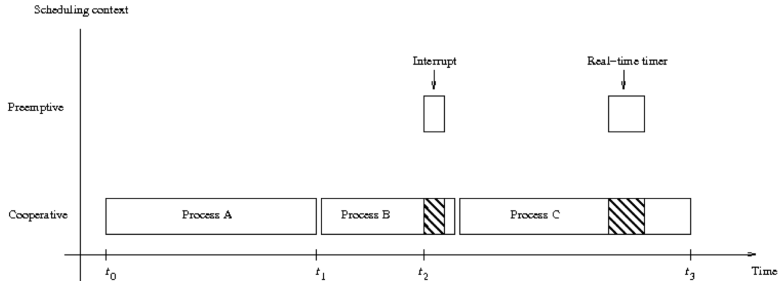

Figura 3.1: Rappresentazione grafica dei contesti esecutivi del codice in Contiki

l'uso della cpu e delle risorse fino al suo completamento, poi restituisce il controllo al kernel del sistema operativo. Il codice che deve essere eseguito può interrompere il codice che viene eseguito nel contesto Cooperative in qualsiasi momento. Quando l'esecuzione di codice cooperative viene bloccata per l'esecuzione di codice preemptive, l'esecuzione del codice cooperative riprenderà solo dopo che è stata ultimata l'esecuzione del codice preemptive. I processi applicativi girano sempre nel contesto cooperative, il contesto preemptive è lasciato ai gestori degli interrupts e ai tasks real-time che devono essere eseguiti necessariamente a istanti temporali programmati;

- TIMERS: Contiki ha un modulo clock e un set di timer libraries: timer, stimer, ctimer, etimer, e rtimer. Le diverse librerie timer hanno diversi usi;
  - Il modulo clock fornisce le funzionalità per gestire il tempo di sistema e per bloccare la cpu per brevi lassi di tempo. Tutte le altre librerie timer sono implementate sopra le funzionalità offerte da questo modulo;
  - Le librerie timer e stimer forniscono la più semplice forma di timer, vengono utilizzate dalle applicazioni per testare se un intervallo temporale è trascorso o meno. L'unica differenza tra le due librerie è la risoluzione del tempo, la libreria timer utilizza i ticks del clock di sistema mentre la libreria stimer utilizza i secondi in modo da permettere più ampi intervalli temporali. Queste librerie contrariamente alle altre possono essere utilizzate anche negli interrupt, risultano particolarmente utili nei driver a basso livello;
  - La libreria etimer fornisce dei particolari timer che allo scadere di un determinato intervallo temporale producono l'evento PROCESS_EVENT_TIMER (descritto in dettaglio in seguito). Questi

timers sono utilizzati nei processi Contiki per attendere un certo periodo di tempo mentre il resto del sistema può continuare a eseguire altri compiti o entrare in un regime low-power;

- La libreria ctimer fornisce timer di callback, viene utilizzata per programmare chiamate a funzioni di callback dopo un certo periodo di tempo. Come gli etimers anche questi ultimi sono utilizzati per attendere un certo intervallo di tempo mentre il sistema continua a svolgere altri compiti o entrare in un regime low-power. Dato che questi timers richiamano delle funzioni quando un certo intervallo temporale è trascorso, sono utili nei codici che non prevedono degli specifici processi Contiki come nelle implementazioni di protocolli;
- La libreria rtimer serve a programmare l'esecuzione dei così detti real-time tasks. La libreria rtimer ha il diritto di prelazione su qualsiasi processo Contiki in esecuzione, in modo da permettere che un determinato real-time task venga eseguito a un preciso istante. I real-time tasks sono utilizzati nei codici time-critical, cioè in quelle implementazioni che non possono tollerare ritardi temporali;

- GESTIONE DELLA MEMORIA: Contiki fornisce tre modi per la allocazione/deallocazione della memoria:
  - MEMB (MEMory Block allocator): è il metodo più utilizzato, fornisce un set di funzioni per la gestione dei blocchi di memoria. I blocchi di memoria vengono allocati nella memoria statica come un array di oggetti a dimensione costante;
  - MEMM (Managed MEMory allocator): viene utilizzato raramente, fornisce un servizio di allocazione dinamica della memoria come la Malloc della libreria standard C, da quest'ultima si differenzia perchè presenta un servizio automatico di deframmentazione dell'area di memoria gestita;
  - MALLOC (Memory ALLOCator): la libreria standard C fornisce un set di funzioni per la gestione dinamica della heap-memory. Il sistema operativo Contiki riserva un'area limitata della memoria complessiva per la heap-memory. Nei sistemi a memoria limitata, come nella nostra trattazione, è conveniente allocare la memoria in modo statico, una gestione dinamica della memoria può portare a problemi di frammentazione, quindi a sprecare della memoria che già risulta scarsa nel suo complesso;

- INPUT - OUTPUT: I dispositivi di I/O disponibili all'utilizzatore finale sono fortemente dipendenti dal tipo di piattaforma hardware che si va a utilizzare. Contiki fornisce due interfacce base che molte piattaforme

condividono come I/O seriale e LEDs. L'output sulla seriale è gestito mediante le API per la stampa fornite dalla libreria standard C. Per quanto riguarda invece l'input da seriale questo dipende fortemente da meccanismi forniti da Contiki. L'API LEDs di Contiki è realizza un'astrazione che permette la gestione dei LEDs in qualsiasi piattaforma Hardware utilizzata;

- LIBRARIES: Contiki fornisce un certo numero di librerie che possono essere utilizzate dai programmi applicativi e dal sistema stesso. Tali librerie contengono moduli per la gestione di liste collegate, per la gestione di buffer ad anello, per la generazione di numeri random, per il calcolo della Cyclic Redundancy Checksums (CRCs) e per il calcolo di una FFT su valori interi;

- FILE SYSTEM: Contiki fornisce un set di file-system per l'utilizzo di diversi tipi di dispositivi di memorizzazione. Tutti questi file-system implementano un sotto-set di funzioni della CFS (Contiki File Systems)-API, solo due di essi hanno funzionalità complete e sono CFS-POSIX e Coffee. Il file system Coffee è pensato principalmente per quei dispositivi equipaggiati con memorie flash o EEPROM (Lo analizzeremo in dettaglio nel Capitolo 4 dedicato all'aggiornamento del firmware over the air). Gli altri file system come CFS-EEPROM, CFS-RAM e CFS-XMEM sono orientati a un singolo file. Le applicazioni accedono ai files system attraverso le funzioni della CFS-API;

- MULTITHREADING: Contiki gestisce i preemptive threads attraverso la libreria mt. Gli mt threads hanno uno stack e un program counter privato che vanno salvati quando si realizza un thread context switch. La libreria è divisa in due parti, una parte che viene definita architecture-indipendent e una parte definita architecture-dependent. Chi realizza applicazioni multhreading in Contiki si deve preoccupare della sola parte architecture-indipendent.

3.1.1 Protothread e programmazione event-driving

Prima di entrare nel dettaglio della tecnica di programmazione fornita da Contiki, descriviamo le tecniche utilizzate nei tradizionali sistemi operativi e non solo. Nei sistemi operativi general-purpouse la tecnica di programmazione utilizzata è quella del multithreading. Il multi-threading è una tecnica di programmazione che permette l'esecuzione contemporanea di più programmi su di un microprocessore condiviso. Nella programmazione multi-threading a ogni programma è associato un proprio thread di controllo che gira parallelamente

a tutti gli altri threads. Per far sì che più programmi vengano eseguiti contemporaneamente il sistema operativo realizza il così detto context switch, così che ogni programma abbia il suo cpu-time per essere eseguito. Ogni thread è protetto da ogni altro, equivale a dire che un thread può comunicare con un altro thread solo attraverso delle funzioni messe a disposizione dal sistema operativo. In un contesto simile i thread vengono comunemente chiamati processi.

La tecnica di programmazione multi-threading mal si presta al contesto degli Smart-Object considerati in questa trattazione per due semplici motivi: il primo motivo è che ogni thread ha bisogno di parecchia memoria per salvare le informazioni di esecuzione cioè il così detto stack del thread; il secondo motivo è che non si può conoscere a priori di quanta memoria ha bisogno ogni thread per salvare il suo stack, quindi è necessario sovradimensionare la memoria di stack. Come è semplice concludere questi due motivi mal si rapportano alla limitatezza di memoria che caratterizza i sistemi embedded in generale. A seguito di quanto detto ne consegue che la tecnica di programmazione multi-threading difficilmente viene utilizzata nella programmazione degli Smart-Objects. La tecnica di programmazione utilizzata nel caso di WSN per l'IoT prende il nome di Event-Driving, tale tecnica richiede molto meno memoria della tecnica-multithreading perché non vi sono thread e quindi non vi sono stack da salvare. L'intero sistema può essere visto come un unico thread che al massimo richiede un singolo stack. Il paradigma di programmazione event-driven ben si sposa con la natura degli smart-object che tipicamente si trovano in un ambiente che è event-driven.

Per lungo tempo la comunità internazionale si è dibattuta su quale delle due tecniche di programmazione sopra descritte sia la migliore. Quello che si può dimostrare è che i due modelli hanno prestazioni equivalenti. Esistono delle tecniche per scrivere il codice che permettono di trarre i vantaggi di entrambe le tecniche di programmazione, i protothread sono una di esse. La tecnica di programmazione che si basa sui protothread è quella adottata da Contiki. In Contiki i programmi girano nei processi, un processo consiste di due parti: un process control block e un process thread. Il process control block viene memorizzato nella RAM e contiene informazioni di esecuzione del processo come il nome del processo, lo stato del processo e un puntatore al thread del processo. Tale struttura dati viene utilizzata solo ed esclusivamente dal kernel di Contiki. È importante sottolineare come tale process control block sia leggero in termini di memoria utilizzata per contenerlo, sono sufficienti due byte di memoria. Il thread del processo contiene il codice del processo ed è memorizzato nella ROM. Il thread è un singolo protothread che viene invocato dallo scheduler dei processi.

Un protothread è un modo di strutturare il codice che permette al sistema operativo di realizzare altre attività mentre quel codice aspetta che qualcosa

accada. Un protothread combina i modelli di programmazione multithreading ed event-driven utilizzando in modo efficiente la memoria. Il concetto di protothread è stato sviluppato nel contesto di Contiki ma ben si adatta ad altri contesti. Possiamo considerare il protothread una funzione C a tutti gli effetti che inizia e termina con delle speciali macro. Il protothread fornisce alle funzioni C dei meccanismi che permettono loro di funzionare come se fossero dei thread ma con minor richiesta di memoria. Il limite al consumo di memoria è un punto di forza dei protothread. I processi in Contiki implementano una loro versione di protothreads, che permette loro di attendere l'arrivo di eventi. Le macro utilizzate nei processi Contiki sono significativamente diverse dalle tipiche macro utilizzati in un contesto protothread puro. Riportiamo qui di seguito le macro dei protothread utilizzate nei processi Contiki:

- PROCESS_BEGIN() : dichiara l'inizio del protothread di un processo;
- PROCESS_END(): dichiara la fine del protothread di un processo;
- PROCESS_EXIT() : esce dal processo;
- PROCESS_WAIT_EVENT(): attende un evento qualsiasi;
- PROCESS_WAIT_EVENT_UNTIL(): aspetta un evento qualsiasi ma con una condizione;
- PROCESS_WAIT_UNTIL(): aspetta per una data condizione;
- PROCESS_PAUSE(): mantiene temporaneamente il processo.

Nel contesto di Contiki un processo viene eseguito quando riceve un evento; Contiki distingue due tipologie di eventi: l'evento Sincrono e quello Asincrono, per ognuno di essi adotta una diversa strategia di gestione. Quando un evento asincrono viene inviato, viene posto nella coda degli eventi del kernel e viene consegnato al processo destinatario un po' di tempo dopo. Nel tempo che intercorre tra l'invio e la consegna di un evento asincrono esso viene mantenuto nella coda degli eventi che è interna al kernel di Contiki. Il kernel legge in modo ciclico la coda degli eventi, mentre la legge invia l'evento al processo destinatario invocandolo. Il destinatario di un evento asincrono può essere un processo specifico o a tutti i processi in esecuzione. Per l'invio di un evento asincrono Contiki mette a disposizione dei programmatori la funzione *process_post()* che prima di inviare l'evento va a testare che nella coda vi sia spazio libero per ospitare il nuovo evento. Nel caso in cui la coda fosse piena ritorna un codice di errore, altrimenti inserisce l'evento alla fine della coda. Quando viene inviato un evento sincrono, questo è direttamente consegnato al processo destinatario senza attese. L'invio di un evento sincrono può essere considerato alla stessa stregua di una chiamata di funzione, il processo destinatario è direttamente

invocato ed il processo che invia l'evento viene bloccato fino a che il processo ricevente non ha terminato di elaborare l'evento. Si ricordi che gli eventi sincroni possono essere inviati a specifici processi.

Contiki permette l'esistenza di uno speciale evento che prende il nome di richiesta di polling. La chiamata alla funzione *process_poll()* per un certo processo fa si che sia programmata l'esecuzione nel più breve tempo possibile. Il processo riceve un particolare evento che lo informa sul fatto che sia stato interrogato. Il polling viene utilizzato per far girare un processo da un interrupt. Affinché un processo riconosca che tipo di evento lo abbia invocato Contiki definisce un sistema d'identificatori degli eventi costituito da costanti numeriche a 8-bit. Quando il kernel di Contiki invoca un processo gli passa uno di questi identificatori in modo che il processo destinatario possa riconoscere l'evento che l'ha invocato, quindi realizzare delle diverse operazioni in base alla tipologia di evento. Gli identificatori con valore sotto a 127 possono essere liberamente utilizzati dai processi d'utente (ovvero le applicazioni create dagli utenti), mentre gli identificatori con valore da 128 in su sono gestiti dal kernel ovvero destinati per identificare determinati eventi. Qui di seguito riportiamo una descrizione sommaria degli eventi gestiti dal Contiki kernel e il relativo identificatore assegnato:

- PROCESS_EVENT_NONE: 128 non identifica alcun evento;
- PROCESS_EVENT_INIT: 129 inviato a tutti i nuovi processi avviati;
- PROCESS_EVENT_POLL: 130 inviato a un processo che è polled;
- PROCESS_EVENT_EXIT: 131 è inviato al processo che sarà "killed" dal kernel, ricevuto tale evento si preoccuperà di liberare tutte le risorse allocate;
- PROCESS_EVENT_CONTINUE: 133 È inviato dal kernel a un processo precedentemente messo in attesa;
- PROCESS_EVENT_MSG: 134 è tipicamente utilizzato dallo stack IP per comunicare ad un processo che un messaggio è arrivato. Può essere utilizzato anche tra processi generici;
- PROCESS_EVENT_EXITED: 135 è inviato a tutti i processi attivi da un processo che sta per essere terminato;
- PROCESS_EVENT_TIMER: 136 è inviato a un processo quando un etimer è scaduto;

Il kernel di Contiki è dotato di un process Sheduler che ha il compito di invocare i processi quando è arrivato il loro momento di essere eseguiti. I

processi vengono invocati semplicemente chiamando la funzione che implementa il thread del processo in questione. Un processo viene invocato in risposta ad un evento che è stato passato, o a seguito di una richiesta di polling; il process scheduler passa l'identificatore dell'evento al processo che sarà invocato.

Contiki fornisce diverse modalità per avviare i processi, la più comunemente utilizzata è la modalità di *autostart* dove i processi presenti in una lista vengono avviati al boot-up del sistema. Tale modalità di avvio dei processi è gestita da un apposito modulo del sistema operativo. I processi utente che devono essere avviati all'avvio del sistema sono contenuti nella lista di autostart che viene creata dall'utente ed utilizzata dal modulo di autostart. I processi vengono avviati con l'ordine in cui compaiono nella lista. L'altra modalità utilizzata per avviare i processi utente è quella di utilizzare un'apposita funzione che il sistema operativo mette a disposizione per l'avvio dei processi. La funzione di cui stiamo parlando è la *process_start()*. Tale funzione opera nel modo seguente: crea la struttura di controllo del processo in memoria, mette il processo nella lista dei processi attivi del kernel e chiama il codice di inizializzazione contenuto nel process-thread. Prima di avviare un processo questa funzione realizza un controllo sulla lista dei processi attivi per verificare che tale processo non sia già stato avviato. Dopo che la verifica ha dato riscontro positivo il processo viene caricato nella lista ed il suo stato settato a *process_state_runnig*. Al processo viene passato in ultimo l'evento PROCESS_EVENT_INIT() che comporta l'esecuzione della prima parte del process-thread, cioè quella parte di inizializzazione che deve essere eseguita una sola volta. Per terminare un processo Contiki mette a disposizione le seguenti alternative:

- un processo può chiudere se stesso chiamando la funzione PROCESS_EXIT();
- un processo termina quando si giunge all'esecuzione della PROCESS_END();
- un processo può essere chiuso da un altro processo che chiama la funzione *process_exit ()*.

Nelle situazioni precedenti il sistema operativo invia un evento sincrono PROCESS_EVENT_EXITED a tutti i processi presenti nella lista dei processi attivi con la differenza che, se la chiamata alla chiusura del processo è impartita dal processo stesso l'evento verrà inviato a tutti i processi tranne che al processo interessato, cosicché gli altri processi provvedano a liberare le risorse occupate dal processo da chiudere. Se invece la richiesta di chiusura di un processo è partita da un altro processo, l'evento viene inviato anche al processo interessato alla chiusura cosicché esso stesso possa provvedere a liberare le risorse occupate.

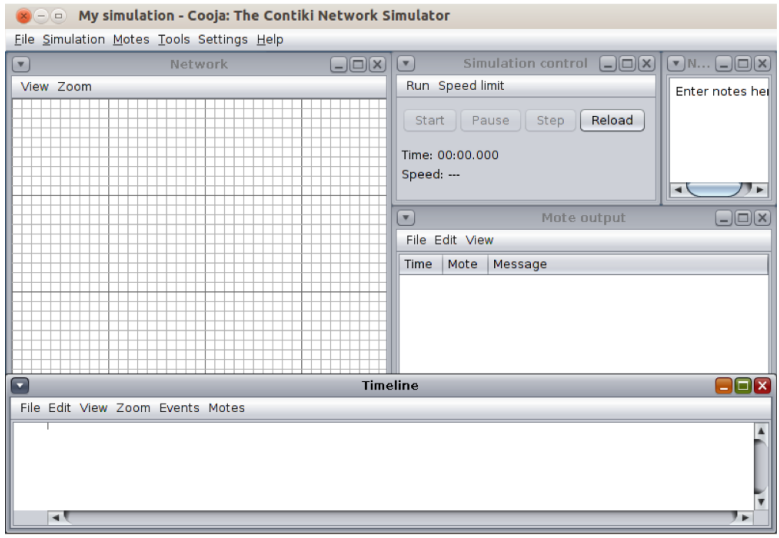

Figura 3.2: Interfaccia grafica di Cooja

3.2 Il simulatore di rete Cooja

Cooja [23] [24] si basa su Java, è contenuto in Instant Contiki e permette di simulare il software destinato ai sensori in un ambiente network-based (Fig. 3.2). In Cooja i nodi possono essere simulati oppure emulati: nella prima metodologia i nodi simulati vengono compilati ed eseguiti nativamente mentre nella seconda metodologia, che richiede più risorse, i nodi utilizzano emulatori di piattaforme dei vari vendors per caricare ed eseguire il firmware. Per utilizzare Cooja, è necessario compilarlo ed eseguirlo, mediante l'istruzione ant run. Dopo la compilazione, viene eseguito Cooja senza simulazione né i plug-in che interagiscono con la simulazione e ricreano i nodi. La simulazione può essere creata tramite la voce di menù File-New Simulation, selezionando le opzioni richieste. Per aggiungere un nodo alla simulazione è necessario che venga prima creato e poi compilato. Dopo la creazione di un nodo è possibile aggiungerne uno o più alla simulazione (Fig. 3.3).

La simulazione può iniziare con due plug-in: un log listener che ascolta le porte seriali di tutti i nodi ed un visualizer che fornisce le informazioni in merito ai nodi. Le simulazioni possono essere salvate in file di estensione .csc, caricabili successivamente tramite la voce di menù File-Open simulation-Browse. Al caricamento della simulazione, tutte le applicazioni Contiki vengono ricompilate. Cooja permette di configurare 4 modelli di propagazione radio:

- UDGM Constant Loss: con questo modello di propagazione il raggio di trasmissione dei nodi è un disco: tutti i nodi all'interno del disco ricevono

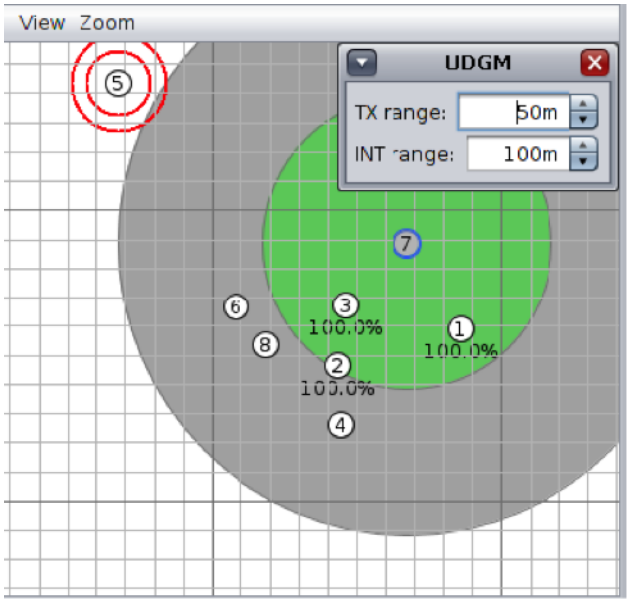

Figura 3.3: Aggiunta di nodi in Cooja

perfettamente i pacchetti trasmessi, quelli al di fuori del disco non ne ricevono alcuno;

- UDGM Distance Loss: simile al modello precedente ma con l'aggiunta di interferenze. È possibile specificare tramite appositi parametri la probabilità di successo della trasmissione e della ricezione dei pacchetti;
- DGRM: Con questo modello è possibile specificare a livello di link i parametri di successo nell'invio/ricezione dei pacchetti. È possibile inoltre considerare i ritardi di propagazione;
- MRM: Questo modello utilizza la tecnica ray-tracing; viene considerata la presenza di oggetti che possono attenuare il segnale nonché provocare rifrazioni, riflessioni e diffrazioni.

Cooja dispone inoltre di un apposito toolbox che permette di valutare il tempo di utilizzo dell'interfaccia radio dei singoli nodi della rete. Esso consente di ricavare i tempi in cui le singole radio sono in stato di ON, TX o RX. Il tempo, oltre che in formato percentuale, è fornito in microsecondi.

Per quanto riguarda i parametri ed i risultati in tempo reale della simulazione è possibile usare il tool Collect-View messo a disposizione nel sistema operativo. Tale tool, una volta fatta partire la simulazione, consente, tramite il semplice clic con il tasto destro su un qualsiasi nodo della rete simulata, di analizzare

direttamente da interfaccia grafica le informazioni relative alla simulazione in corso. Possono essere visualizzati i seguenti elementi:

- TOPOLOGICAL GRAPH:
  - mappa di sensori;
  - grafico di rete;
- SENSOR RELATED PLOTS:
  - sensore di temperatura (temperatura, temperatura media);
  - sensore di umidità (umidità relativa);
  - sensore di batteria (indicatore di batteria e voltaggio di batteria);
  - sensore di luce.
- NETWORK METRICS RELATED PLOTS:
  - nodi vicini;
  - intervallo Beacon (intervallo tra 2 segnali inviati dal client alla rete);
  - network hops (nel tempo e per nodo);
  - router metrico (nel tempo, istantaneo e medio);
  - EXT (fine trasmissione);
  - next hop;
  - latenza;
  - pacchetti persi (nel tempo);
  - pacchetti ricevuti (nel tempo, per nodo, ogni 5 minuti).
- POWER RELATED PLOTS:
  - potenza media;
  - potenza istantanea;
  - andamento della potenza (power history);
  - radio duty-cycle.
- OTHER TABS:
  - node info (sintesi di tutti i nodi e loro caratteristiche);
  - console seriale (l'utente interagisce direttamente,inviando comandi per leggere e trasmettere dati).

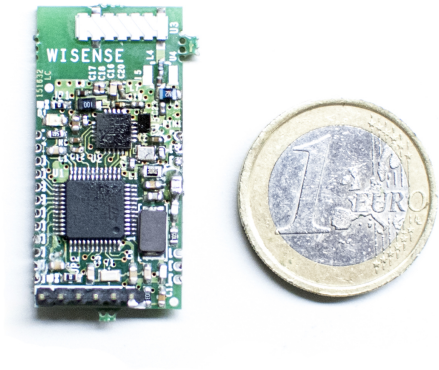

Figura 3.4: Hardware utilizzato

3.3 Piattaforma utilizzata

Nella scelta della piattaforma da utilizzare per lo sviluppo del progetto e per le verifiche sul campo delle soluzioni implementate un aspetto chiave è rappresentato dagli obiettivi e dalle specifiche che ci siamo imposti. Tale sistema, infatti, per rispettarle deve prima di tutto essere basato su tecnologie a bassissimo consumo energetico ed avere caratteristiche di portabilità e customizzazione, oltre ad ingombri molto ridotti, nell'ottica di essere integrato in applicazioni per la creazione di oggetti Smart. I nodi che costituiscono la rete mesh sono delle schedine embedded ad alto grado di integrazione mostrate in Fig. 3.4, di cui ci limitiamo a farne una descrizione degli elementi principali.

Riportiamo un elenco che focalizza l'attenzione sui principali componenti che le costituiscono:

- il microcontrollore STM32L151CCT6 (ST-Microelectronics) `http://www.st.com/en/microcontrollers/stm32l151cc.html`;
- il transceiver radio Spirit1 (ST-Microelectronics);
- una memoria EEPROM;
- l'antenna ceramica integrata.

Il microcontrollore STM32L151CCT6 è un dispositivo a bassissimo consumo di potenza, basato sulla CPU ARM-Cortex- M3 a 32-bit, quindi con delle buone capacità di calcolo; riportiamo qui di seguito le principali feature:

- alimentazione: da 1.65 V a 3.6 V;
- range di temperature di funzionamenteo -40°C a +85°C;
- bassa corrente assorbita in modalità Stanby: 0.29μA;
- 8 s wakeup time;
- Core: ARMCortex-M3 32-bit CPU:
  - da 32 kHz fino a 32 MHz max;
  - memory protection unit;
- gestione dell'alimentazione e funzionalità di Reset:
  - low power, BOR (brown out reset) con 5 soglie selezionabili;
  - ultra-low-power POR/PDR;
  - programmable voltage detector (PVD);
- possibili sorgenti di clock:
  - cristallo oscillatore da 1 a 24 MHz;
  - oscillatore 32 kHz con calibrazione;
  - circuito RC interno ad alta velocità di oscillazione: 16MHz con taratura di fabbrica (+/-1%);
  - circuito RC interno a basso consumo di potenza: 37 kHz;
  - PLL interno a basso consumo di potenza e multi velocità da 65kHz a 4.2MHz, per il clock della CPU e della USB(48 MHz);
- bootloader pre-programmato:
  - USB e USART supportate;
- strumenti per lo sviluppo:
  - supporta Serial wire debug;
  - supporta JTAG and trace;
- memorie:
  - 256 KB Flash con ECC(Error Code Cheksum);
  - 32 KB RAM;

  - 8 KB di vera EEPROM con ECC;
  - 128 byte per il backup dei registri;
- LCD Driver per schermi fino a 8x40 segments;
  - support contrast adjustment;
  - support blinking mode;
  - step-up converter on board;
- periferiche analogiche:
  - 2 Amplificatori operazionali;
  - 12-bit ADC 1Msps fino a 25 canali;
  - 12-bit DAC 2 canali con output buffers;
  - 2 comparatori a basso consumo di potenza;
  - DMA controller a 12 canali;
- interface di comunicazioni per 9 periferiche;
  - 1 USB 2.0 (PLL 48 MHz interno);
  - 3 USART;
  - 3 SPI 16 Mbits/s (2 SPI con I2S);
  - 2 I2C (SMBus/PMBus);
- 11 timers: 1 a 32-bit, 6 a16-bit con 4 canali IC/OC/PWM , 2 a16-bit timer di base, 2 watchdog timers indipendenti;
- fino a 23 canali capacitivi di sensing;
- unità di calcolo per il CRC, ID-univoco a 96-bit.

In merito al transceiver radio Spirt1 `http://www.st.com/en/microcontrollers/stm32l151cc.html`, ed al suo funzionamento, possiamo dire che è un transceiver RF a bassissimo consumo di potenza, destinato alle applicazioni wireless nella banda sub-GHz-band. Può lavorare nelle bande libere da 169, 315, 433, 868, e 915 MHz ma può essere programmato per operare anche nelle seguenti bande: 300-348 MHz, 387-470 MHz, e 779-956 MHz. Il data rate in aria può essere selezionato da 1 a 500 kbps, può lavorare nei sistemi con spaziatura tra i canali che va da 12.5 a 25kHz. Lo Spirit1 incorpora un modem interno per la modulazione/demodulazione dei dati. Esso è in grado di realizzare il CRC sui dati, nonché la FEC (Forward Error Correction) sui pacchetti. Lo spirit1 fornisce un meccanismo automatico per la gestione degli ack, delle ritrasmissioni

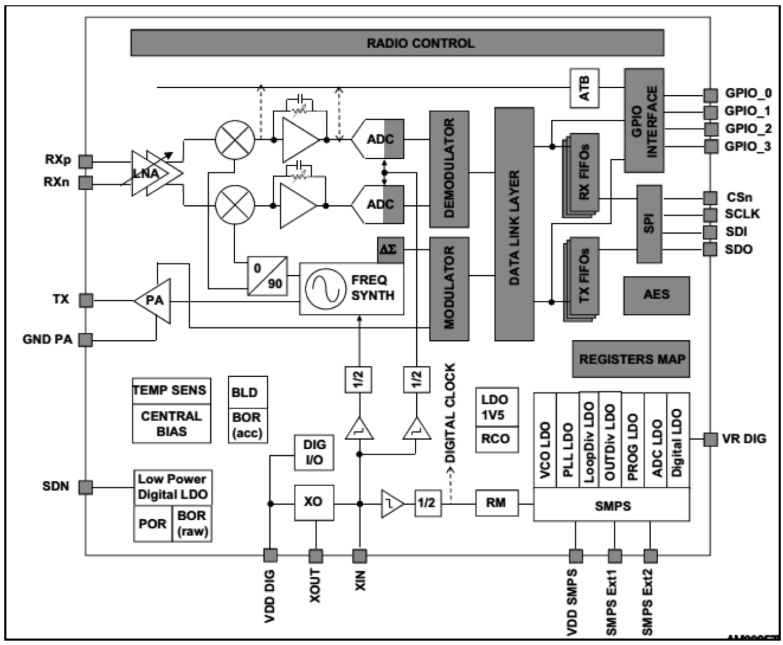

Figura 3.5: Schema a blocchi dello SPIRIT1

e dei time-out. Attraverso un algoritmo di switching control può gestire anche tecniche di antenna diversity. Supporta i seguenti schemi di modulazione: 2-FSK, GFSK, OOK, ASK e MSK. I bytes dei dati ricevuti e da trasmettere sono salvati in due diversi buffer FIFO a tre livelli (TX-FIFO e RX-FIFO), accessibili mediante SPI.

Per comprendere la descrizione qui di seguito occorre osservare lo schema di Fig. 3.5. L'architettura di ricezione realizza una conversione a bassa frequenza intermedia. Il segnale RF ricevuto viene amplificato da un amplificatore (LNA) low-noise a due stadi e convertito in discesa in quadratura (I e Q) alla frequenza intermedia (IF). LNA e gli amplificatori a IF costituiscono il front-end di ricezione (RXFE) che ha un guadagno programmabile. Alla frequenza intermedia I due segnali in quadratura, I e Q, sono digitalizzati attraverso gli ADCs. I dati demodulati sono forniti ad un MCU esterno attraverso una RX-FIFO a 96byte che è leggibile mediante SPI, o direttamente attraverso un GPIO pin programmato per lo scopo. Un co-processore è disponibile per realizzare la cifratura AES-128bit dei dati da trasmettere.

La parte di trasmissione dello SPIRIT1 si basa sulla sintesi diretta della frequenza RF. L'ingresso del Power Amplifier (PA) è costituito dall'uscita LO del sintetizzatore di frequenza; il livello di uscita del PA può essere configurato tra -30dBm e +11dBm, con passi di 0.5 dBm. I dati da trasmettere sono forniti

da MCU all'esterno attraverso la TX-FIFO a 96bytes, direttamente scrivibile mediante SPI, o attraverso un GPIO pin opportunamente configurato.

Lo SPIRIT1 supporta la tecnica di Frequency Hopping in TX/RX e realizza strategie di switching per tecniche di antenna diversty. È dotato di una unità di gestione della Potenza (Power Managment System) davvero efficiente; è munito di un regolatore SMPS (Switched Mode Power Supply) che permette allo spirit1 di essere alimentato da batterie nel range di +1.8V a 3.6V. Un cristallo può essere connesso tra i pin XIN-XOUT. Si può configurare digitalmente il tipo di cristallo da utilizzare. Per specifiche applicazioni si può alimentare il canale XIN con una sorgente esterna di clock. Lo SPIRIT1 ha anche un oscillatore RC integrato che genera il segnale a 34.7kHz, utilizzato come clock per quelle funzioni che richiedono lunghi timeout.

Uno standard SPI-bus a 4-pin viene utilizzato per comunicare con la MCU esterna. Quattro GPIO pins configurabili sono disponibili. Ci soffermiamo brevemente su come viene calcolato l'indice RSSI (Received Signal Strength Indication) dallo SPIRIT1 e sul significato che ha il dato digitale restituito in uscita. RSSI è un indice che individua la potenza del segnale ricevuto in antenna, valutata nella banda del filtro di canale. Quindi dalle conoscenze sulle modulazioni numeriche possiamo sottolineare come nel filtro di canale avremo una certa densità spettrale di potenza dovuta solo ed esclusivamente al simbolo trasmesso. Tale densità di potenza dipende fortemente dal tipo di modulazione scelto, dalla sequenza informativa trasmessa, dalla potenza utilizzata in trasmissione. Nel nostro caso avendo adottato modulazione GFSK_BT1, equivale a dire che la parte deterministica sarà caratterizzata dalla trasformata di Fourier di un impulso di durata pari al tempo di bit ma caratterizzato da shaping smussato e non dal tradizionale impulso rettangolare. Lo smussamento dell'impulso lo si ottiene facendo passare l'impulso rettangolare attraverso un filtro gaussiano. Per quanto riguarda il contributo della parte statistica allo spettro complessivo non entreremo nel dettaglio, comunque possiamo sottolineare come sia fortemente influenzato dalle proprietà di correlazione che vi sono tra i bit prodotti dalla sorgente di informazione.

La lettura di potenza ricevuta che ci troviamo nel RSSI_LEVEL Register dello Spirit1 sarà ottenuta a partire da questa densità spettrale di potenza. Il contenuto del RSSI_LEVEL Register viene aggiornato alla fine della ricezione di un pacchetto, quindi ciò ci permette di dire che il valore di RSSI-LEVEL letto alla fine di un certo pacchetto sarà disponibile per la tutta la durata della ricezione del pacchetto successivo. Il registro RSSI_LEVEL_Register è un registro ad 8-bit ed il dato è formattato come un intero senza segno; ecco perché il valore di RSSI restituito è compreso tra 0 e 255. La formula che viene riportata con le testuali parole del data-sheet è utile per convertire il conteggio presente nel RSSI_LEVEL_Register in un valore in dBm.

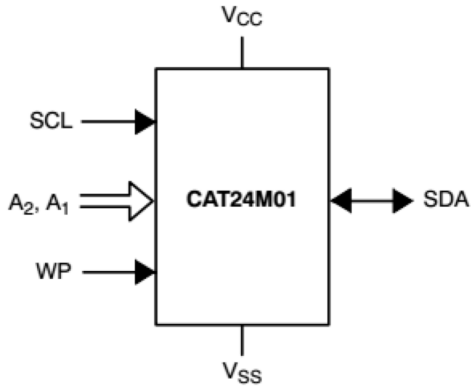

Figura 3.6: Simbolo della CAT24M01

Concludiamo la descrizione dell'hardware utilizzato con la memoria EEPROM installata il cui schema è riportato in Fig. 3.6. Si è scelta la CAT24M01 della ON Semiconductor. La CAT24M01 è una Serial EEPROM CMOS da 1024 Kb, organizzata internamente come 131,072 words da 8 bit ciascuna. È equipaggiata con un buffer di scrittura delle pagine da 256 byte e supporta il protocollo Standard Fast e Fast-Plus I2C. Le operazioni di scrittura possono essere inibite attraverso il PIN WP, così da proteggere l'intera memoria. I PIN Address esterni invece rendono possibile il collegamento fino a quattro dispositivi sullo stesso bus. Offre inoltre un On-Chip ECC (Error Correction Code) che la rendono utile anche per applicazioni dove è richiesta una alta affidabilità. Di seguito le specifiche:

- standard supportati Fast and Fast'Plus I2C Protocol;
- alimentazione da 1.8 V to 5.5 V;
- 256 Byte Page Write Buffer;
- protezione in scrittura per l'intera memoria hardware;
- filtri di Schmitt Triggers and Noise Suppression su I2C Bus Inputs (SCL and SDA);
- Low Power CMOS Technology;
- 1,000,000 Program/Erase Cycles;
- 100 Year Data Retention;
- Industrial and Extended Temperature Range;
- 8 pin PDIP, SOIC, TSSOP and 8 pad UDFN Packages.

Capitolo 4

FOTA

Nell'esperienza con le reti di sensori wireless una delle problematiche più evidente è che molto spesso riprogrammazione dinamica del nodo è una caratteristica indispensabile nello sviluppo di nuove applicazioni e prodotti. Tale procedura prende il nome di FOTA (Firmware Over The Air).

Gli aggiornamenti firmware per reti di sensori sono necessari per una varietà di ragioni che vanno, dalla implementazione e test di nuove caratteristiche di un programma esistente, alla completa riprogrammazione dei nodi quando viene installata una nuova applicazione, alla correzione di bugs rilevati in servizio. I vincoli di risorse in termini di energia, di memoria, e potenza di elaborazione fanno sì che la riprogrammazione di una rete di sensori possa risultare un compito molto oneroso. Nel mondo della ricerca e, anche in quello industriale, sono stati sviluppati molti meccanismi diversi per la riprogrammazione dei nodi sensori che vanno dalla sostituzione completa dell'immagine, alle macchine virtuali. Se a questo aggiungiamo che spesso i nodi vengono posizionati in luoghi non sempre facilmente accessibili e dove si desidera il minore intervento possibile da parte di un operatore, si rende anche necessaria una completa affidabilità del processo.

4.1 Principali tecniche di aggiornamento firmware Over the Air

In questa vengono presentate le principali tecniche per l'aggiornamento in servizio dei nodi sensore all'interno di una WSN, focalizzando l'attenzione sui principi di funzionamento e sui vantaggi e svantaggi tipici di ciascuna procedura. Molti studi sono stati condotti nel campo delle WSN utilizzate nel settore automobilistico e soprattutto nei veicoli Smart di prossima generazione come le auto elettriche [25] [26].

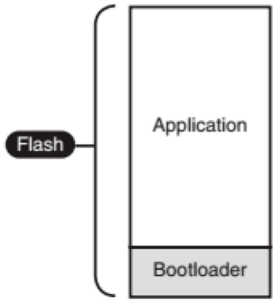

Figura 4.1: Organizzazione della memoria con Bootloader

4.1.1 Remote system update tradizionale

I sistemi tradizionali di upgrade firmware da remoto fanno affidamento su di una applicazione di bootloader che viene eseguita al reset del sistema, lo schema è riportato in Fig. 4.1. Un server remoto invia il nuovo firmware alla WSN e poi ciascun dispositivo si occupa della scrittura del file ricevuto nella memoria del microprocessore. In molti casi, quando il procedimento d'aggiornamento è iniziato, l'applicazione principale viene fermata.

Nella maggior parte dei casi tale metodo comporta la completa cancellazione e riprogrammazione dell'applicazione principale. Così facendo si dispone di una sola copia dell'applicativo, quindi se abbiamo degli errori non rilevati quando il nuovo codice è ricevuto o scritto in memoria, il sistema potrebbe non operare correttamente o addirittura non funzionare affatto dopo l'aggiornamento. Lo scenario peggiore è quando il sistema non risponde più ed è incapace di riavviarsi per eseguire nuovamente l'aggiornamento non andato a buon fine. Vantaggi:

- facile da implementare;
- necessita di poca memoria per l'assenza di qualsiasi tipo di backup.

Svantaggi:

- l'applicazione principale deve essere fermata durante l'aggiornamento;
- non è possibile tornare indietro ad una versione funzionante;
- possibile sensibilità alla perdita di alimentazione durante il processo di ricezione ed scrittura del nuovo firmware.

Questa tipologia di procedura di aggiornamento è utilizzata, ad esempio, dalla ATMEL con alcune accortezze appunto per cercare di mitigare gli aspetti

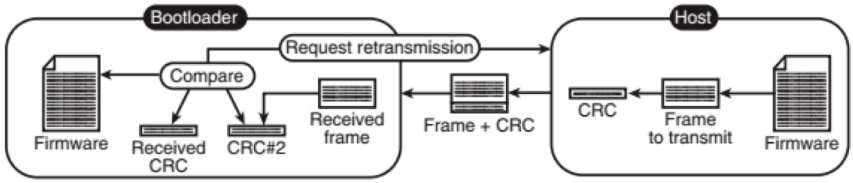

Figura 4.2: Individuazione degli errori durante la trasmissione

negativi sopra descritti. Tali interventi di ottimizzazione della procedura sono di seguito descritti:

- Correzione/individuazione degli errori.

  Si utilizzano dei semplici codici di rivelazione degli errori e, laddove questo non influisca troppo sulla durata delle batterie o sull'onere computazionale, anche di correzione di questi ultimi. La scelta sulla politica da adottare, correzione dell'errore o ritrasmissione dell'informazione contenente errori, viene di volta in volta scelta in funzione delle risorse energetiche e di memoria che si hanno a disposizione. In ogni caso, nella maggior parte delle situazioni, alla correzione degli errori viene preferita la ritrasmissione dell'intero pacchetto firmware o parte di esso. In Fig. 4.2 viene mostrato lo schema di un applicativo che fa uso di un codice rilevazione degli errori basato su CRC e, laddove rilevi anomalie, procede alla richiesta di ritrasmissione del pacchetto e non al tentativo di correzione dell'errore.

- Numerazione dei pacchetti.

  La numerazione dei pacchetti viene utilizzata per evitare che i pacchetti arrivino in ordine sbagliato o che ne manchi qualcuno. Questa è una situazione critica in un aggiornamento file-oriented. Un errore del genere può rendere il codice ricevuto inutilizzabile. Come suggerisce il suo nome, la numerazione dei pacchetti avviene semplicemente aggiungendo un campo con il numero di pacchetto. Questo numero viene incrementato di uno per ogni nuovo pacchetto. In questo modo il Bootloader può facilmente riscontrare eventuali anomalie. Un esempio del meccanismo di ricetrasmissione dei pacchetti numerati e di come questo risulti utile nella procedura di aggiornamento è riportato in Fig. 4.3.

- Acknowledgement del pacchetto

  L'Acknowledgement lavora nel seguente modo. Ciascuna volta che viene trasmesso un pacchetto di dati, si aspetta l'ack del ricevente. Nessun altro pacchetto viene trasmesso se non arriva prima l'ack. Ne consegue

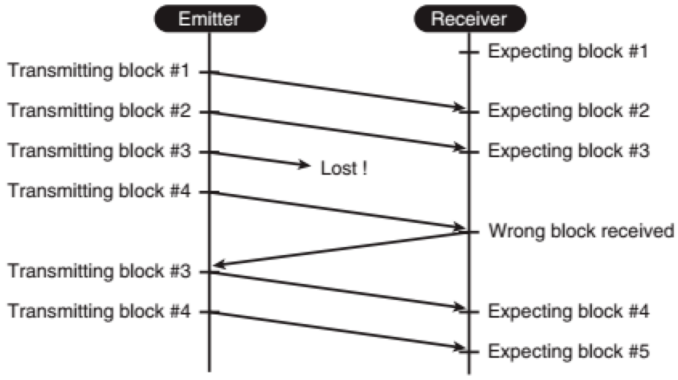

Figura 4.3: Esempio di individuazione degli errori tramite numerazione dei pacchetti

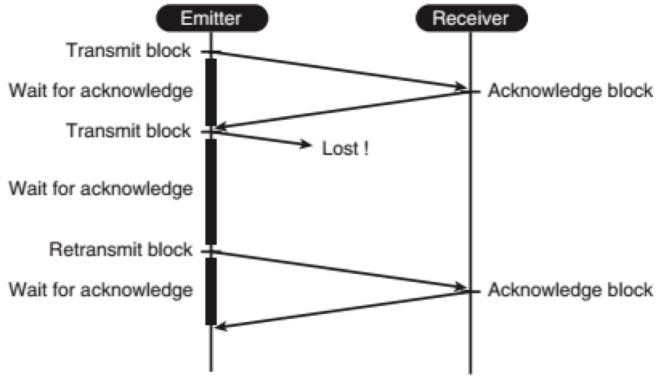

Figura 4.4: Esempio di trasmissione con acknowledge

che i pacchetti non potranno essere ricevuti nell'ordine errato visto che ne viene inviato uno alla volta. Tale procedura, seppur estremamente affidabile, risulta troppo poco efficiente in termini di tempo e, quindi di risorse energetiche, pertanto molto spesso anziché procedere con il singolo riscontro dei pacchetti si utilizzano tecniche per il riscontro di blocchi di pacchetti contemporaneamente come mostrato in Fig. 4.4.

4.1.2 Flash Swap

Nelle periferiche con due o più blocchi di flash interna può essere supportato lo swap delle memorie, cioè l'indirizzo base delle memorie di ciascun blocco può essere scambiato nella mappa logica del dispositivo. Dopo il reset del dispositivo, il sistema swap incorporato nella flash selezionerà quale software eseguire

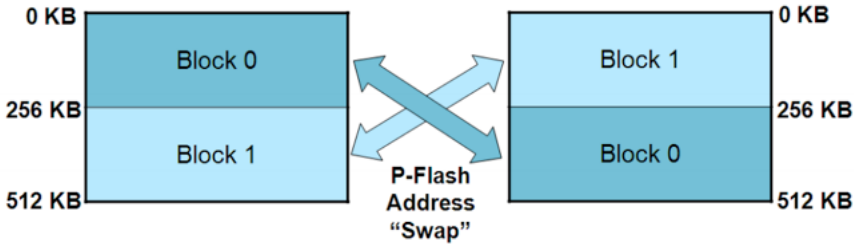

Figura 4.5: Esempio di P-flash swap (Kinetis K60, 512K p-flash)

in funzione della mappa della memoria logica. Questa procedura permette il backup del sistema con l'aggiunta della facilità di programmazione. È possibile eseguire uno dei due blocchi mentre l'altro viene cancellato/programmato.

Vantaggi:

- facile da implementare;
- insensibilità alla perdita di alimentazione;
- non necessita di bootloader;
- ben supporta i sistemi operativi multitasking;
- backup del codice. Possibile recupero di una precedente configurazione funzionante.

Svantaggi:

- richiede il doppio della flash.

La Freescale offre questa tipologia di FOTA sui suoi microprocessori della serie Kinetics Fig. 4.5. Sulle periferiche Kinetics, il sistema flash swap monitora/controlla tutti i passi per lo switch dalla vecchia applicazione alla nuova; ed è aggiunta una procedura di recupero del sistema in caso di perdita di alimentazione durante questi passaggi.

Esistono quattro opzioni per i comandi di swap:

- impostazione dell'indirizzo base dello swap.

  Questo comando inizializza il sistema di swap. Quando viene eseguito bisogna fornire un indirizzo per il puntatore alla swap area. Questo comando abilita lo swap e l'indirizzo fornito verrà utilizzato al prossimo riavvio del sistema;

- impostazione dell'area di swap in modo Update.

  Questo comando serve ad indicare al gestore di swap che un blocco di memoria è stato selezionato per essere aggiornato. In aggiunta viene rimossa la protezione ed il blocco viene cancellato. La cancellazione è una condizione necessaria per la procedura di swap.

- impostazione dell'area di swap in modo Complete.

  Questo comando serve ad indicare al gestore di swap che il blocco non attivo di flash è stato riprogrammato ed è pronto per essere eseguito.

- informazione sullo stato dello swap.

Questo comando controlla lo stato di un blocco di memoria. Ritorna il corrente stato di swap o informazioni sull'eventuale presenza di errori. La procedura di swap segue i seguenti 4 passi per essere espletata:

1. inizializzazione del sistema fornendo l'indirizzo base del nuovo blocco di swap;
2. cancellazione del blocco non attivo;
3. riprogrammazione del blocco con il nuovo software;
4. impostazione dello stato del blocco a Complete.

Una volta impostato lo stato a Complete lo swap verrà eseguito al prossimo riavvio del sistema che andrà ad eseguire il nuovo blocco.

4.2 Tecnica di FOTA sviluppata

Come descritto nelle precedenti sezioni esistono diversi metodi utilizzati per eseguire l'aggiornamento del firmware da remoto, ciascuno con i propri vantaggi e svantaggi. In generale bisogna valutare caso per caso quale tipologia utilizzare e, se necessario anche creare ibridi di metodologie per meglio adattarsi ai vari scenari. Di seguito analizziamo gli aspetti chiave di ciascuna metodologia facendo riferimento al nostro particolare sistema di rete mesh IPv6. Nel primo caso di aggiornamento si deve implementare un boot di avvio che riceva l'aggiornamento e lo scriva direttamente sulla flash del micro. Nella nostra particolare architettura di rete, significherebbe duplicare l'intero stack protocollare descritto nel capitolo 2 all'interno del boot. Questo comporterebbe un incremento eccessivo delle dimensioni del boot che andrebbe ad occupare circa il 50% della flash, limitandone la disponibilità per l'applicazione principale. Nel caso di aggiornamento tramite swap il requisito fondamentale per l'utilizzo di questa tecnica è la disponibilità di memoria Flash nel microprocessore. L'

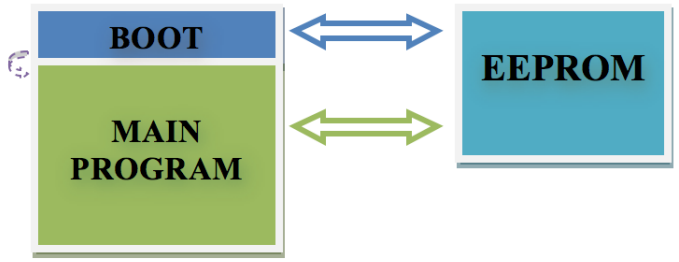

Figura 4.6: Suddivisione della memoria flash

STM32L151CCT6 da noi utilizzato offre 256 Kbyte di memoria che in ambito embedded non sono solitamente pochi. Questo potrebbe far propendere per una scelta di questo tipo. Purtroppo però, l'utilizzo di un sistema operativo e la struttura modulare del firmware comportano una dimensione di base del firmware senza applicativo che supera, seppur di poco, i 100kB. La scelta di questa seconda tecnica comporterebbe una limitazione a 128 kB per la size massima del software che rappresenterebbe un vincolo troppo stringente per lo sviluppo di future applicazioni o integrazione di moduli o evoluzione di protocolli. Tale restrizione risulta inaccettabile su una piattaforma che nasce proprio come modulare e per poter essere impiegata in ambiti diversi. Viste le premesse e vista la presenza nella piattaforma hardware di una EEPROM esterna si è scelto di implementare un metodo ibrido tra le due tipologie. Anziché effettuare lo swap di zone di memoria interna al micro, su utilizzerà la EEPROM esterna per salvare il file binario relativo al nuovo aggiornamento che poi verrà caricato nella memoria interna in fase di aggiornamento. La ricezione del file binario a questo punto può essere effettuata nel main program, senza alcuna necessità, quindi, di interrompere la normale esecuzione dell'applicazione e con il grande vantaggio di utilizzare l'implementazione dello stack IPv6 per la propagazione dei pacchetti di aggiornamento dal border node verso i nodi periferici. In questo caso il Boot si occupa solo dell'interfacciamento con la memoria esterna, del controllo di eventuali errori nel file binario che si trova nella EEPROM e della sua successiva scrittura nella flash, una volta validato. In questo modo il boot può essere relativamente semplice ed occupare una zona di memoria ridotta. Lo schema di funzionamento è riportato in Fig. 4.6.

La procedura di aggiornamento proposta offre anche il vantaggio di poter implementare l'individuazione di eventuali errori di trasmissione e/o scrittura a più livelli come mostrato in Fig. 4.7. Il primo livello di controllo è implementato in fase di ricezione dei singoli pacchetti che costituiscono il file binario dell'aggiornamento sia tramite il protocollo IPv6, che si occupa degli eventuali errori sul canale, sia tramite checksum a 32bit inserito in ogni singolo pacchetto

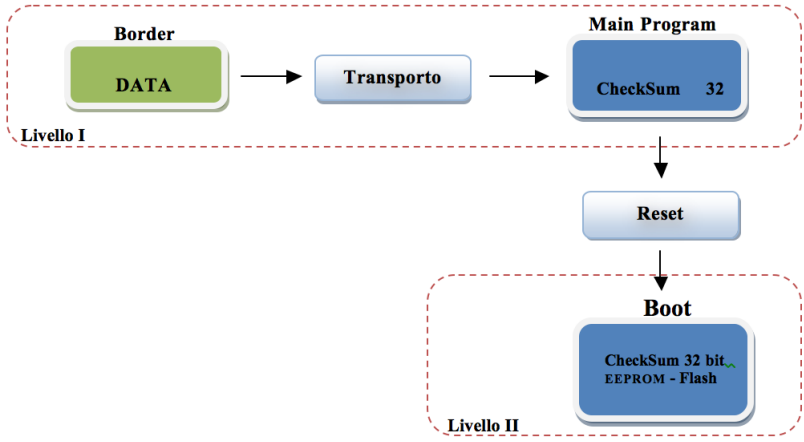

Figura 4.7: Livelli per l'individuazione degli errori

per verificare la correttezza dei dati trasmessi. Il secondo livello è implementato in fase di Boot, dove oltre ad essere controllato l'intero file binario salvato sulla EEPROM esterna, sempre tramite checksum a 32 bit, viene controllato, ad ogni avvio del nodo, l'integrità del firmware presente all'interno del microprocessore. Quest'ultimo controllo è stato introdotto per risolvere eventuali perdite di alimentazione durante la scrittura della flash interna che, nella procedura proposta, rappresenta il momento più critico del procedimento. Infatti, se la scrittura della flash per qualche motivo viene interrotta, al riavvio del nodo il boot si accorgerà che qualcosa è andato storto e ripeterà di nuovo la procedura di aggiornamento del firmware senza incappare in loop o blocchi che causerebbero la perdita di funzionalità del nodo.

Il funzionamento generale, in caso di avvenuta ricezione dell'aggiornamento rappresentato in Fig. 4.8, è il seguente:

1. avvio del Boot;
2. il Boot controlla l'integrita del firmware sul microprocessore;
3. se a buon fine controlla, tramite flag, se deve essere eseguito un aggiornamento;
4. se deve effettuare l'aggiornamento inizializza l'interfaccia verso la EEPROM esterna;
5. controllo dell'integrità del firmware contenuto nella EEPROM;
6. trasferimento del nuovo programma nella memoria interna del micro;
7. avvio del programma aggiornato.

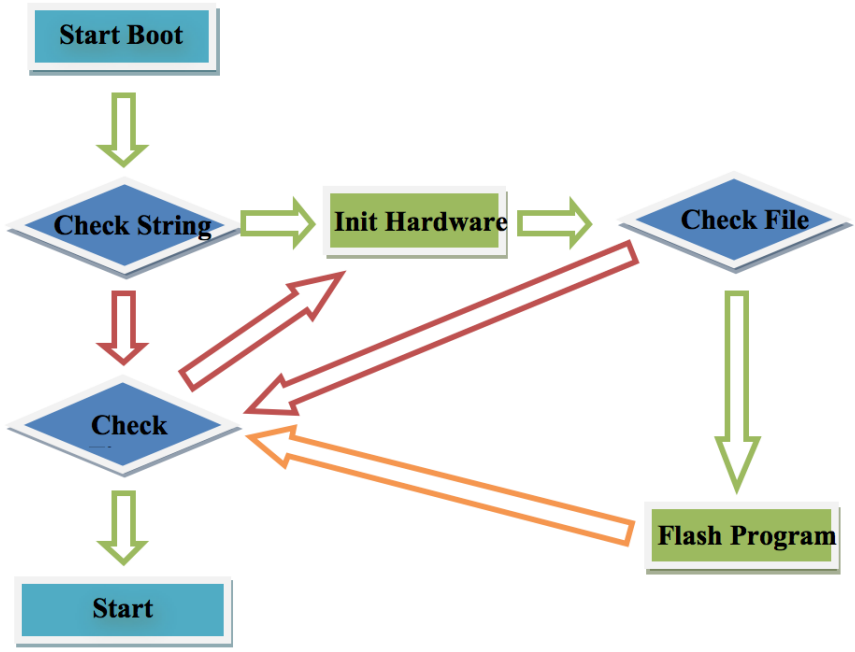

Figura 4.8: Diagramma di stato per la procedura di scrittura dell'aggiornamento in flash dal boot

Una volta avviato il boot è necessario gestire il "jump" all'applicazione. Al programma di boot è stato assegnato uno spazio di indirizzi in ROM da 0x8000000 a 0x8001FFF ed il vector table posizionato a 0x8000000. Il programma da eseguire viene indirizzato dall'indirizzo 0x08004000 a 0x807FFFF ed il vector table posizionato a 0x8004000. Con questo tipo di indirizzamento, all'avvio, il processore esegue prima il codice precedentemente scritto (boot), poi esegue il salto all'indirizzo del codice da eseguire.

La tipologia di FOTA proprosta unisce molti vantaggi delle tipologie già utilizzate, mitigandone allo stesso tempo i rispettivi svantaggi.

Vantaggi:

- boot relativamente facile da implementare;
- insensibilità alla perdita di alimentazione;
- l'applicazione principale non deve essere fermata durante l'aggiornamento.

Svantaggi:

- necessità di memoria esterna.

4.3 Protocollo per la propagazione dell'aggiornamento

Descritta la procedura di aggiornamento una volta ricevuto il file binario, passiamo ora a descrivere come questo file binario viene propagato dal web e dal border fra i nodi della rete mesh. Un importante aspetto da tenere in considerazione in una rete mesh è il numero dei nodi. Questo può essere significativamente elevato ed aggiornare un singolo nodo alla volta potrebbe significare un grosso impegno in termini temporali dell'ordine di giorni per numerosità prossime alle centinaia. Per ovviare a questo problema si è proposta ed implementata una tecnica di propagazione del file binario a livelli come in Fig. 4.9. Il border inizializza la trasmissione dell'aggiornamento solo per i nodi in diretta visibilità radio e con un valore di RSSI medio stimato in funzione dei precedenti 10 pacchetti scambiati non inferiore a -80dBm. Si utilizza cioè la modalità di invio multicast che consente di raggiungere tutti i vicini e di non congestionare la rete in quanto i messaggi non vengono inoltrati dai nodi riceventi. Una volta terminata la ricezione dell'aggiornamento da parte dei nodi in visibilità, soltanto uno di questi, opportunamente selezionato dal border, ritrasmetterà l'aggiornamento ai suoi vicini. Il procedimento sarà poi ripetuto fino a che non si sarà trasmesso l'aggiornamento a tutti i nodi della rete. A questo punto, per evitare possibili buchi nella rete, sarà il border ad inviare in unicast, cioè al singolo nodo, il comando per riavviarsi e quindi iniziare la procedura di aggiornamento del firmware.

4.3.1 Protocollo FOTA per il livello 1

Descriviamo ora nel dettaglio il protocollo implementato per la trasmissione del firmware da parte del border ai propri vicini. La struttura generale di un singolo pacchetto di trasmissione è la seguente (Fig. 4.10):

Elenco dei possibili header da border a nodo:

- richiesta aggiornamento intero firmware

  #define MWN_MESSAGE_UPDATE_REQUEST **WUr**;

- richiesta aggiornamento app

  #define MWN_MESSAGE_UPDATE_NPCK_APP **WUu** ;

- singolo pacchetto app

  #define MWN_MESSAGE_UPDATE_REQUEST_APP **WUb** ;

- singolo pacchetto firmware

  #define MWN_MESSAGE_UPDATE_NPCK **WUc** ;

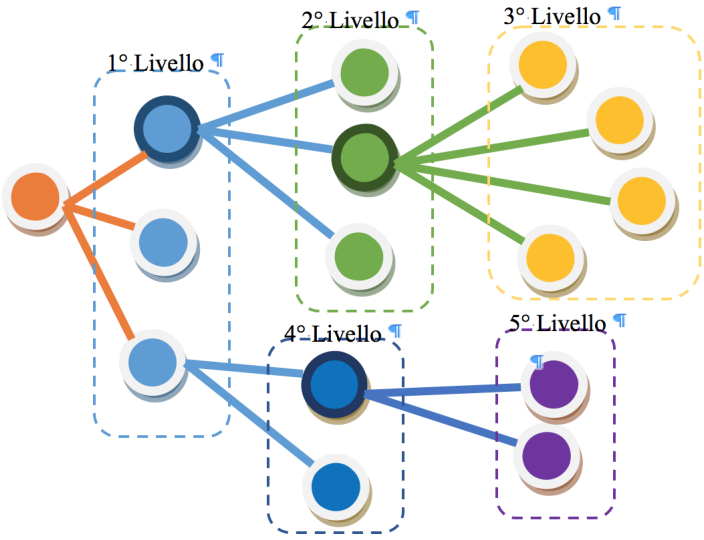

Figura 4.9: Suddivisione a livelli della rete

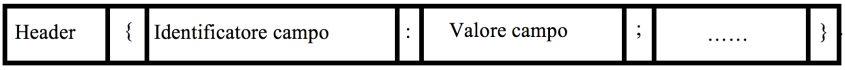

Figura 4.10: Struttura del pacchetto FOTA

- invio dell'update completato da parte del master
  #define MWN_MESSAGE_UPDATE_SEND_COMPLETE **WUd**;
- comando per ordinare la ritrasmissione del firmware
  #define MWN_MESSAGE_UPDATE_REAPETER_CMD **WUe** ;
- comando per effettuare la scansione dei vicini
  #define MWN_MESSAGE_WHO_NEIGHBORS **WUf** ;
- comando per far partire la procedura di aggiornamento
  #define MWN_MESSAGE_UPDATE_START **WUs** ;
- comando per verificare la versione di aggiornamento a disposizione
  #define MWN_MESSAGE_UPDATE_VERIFY **WUv** ;
- comando per scrivere la versione del firmware all'indirizzo MEMORI_ADDRESS_VE
  #define MWN_MESSAGE_WRITE_VER_FR **WUy** .

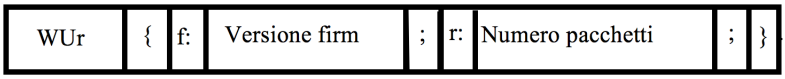

Figura 4.11: Struttura del pacchetto WUr

Elenco possibili header da nodo a border:

- messaggio dei pacchetti mancanti dopo la ricezione del MWN_MESSAGE_UPDATE_S

  #define MWN_MESSAGE_NEEDED_PCK **WUp** ;

- messaggio per informare che tutti i pacchetti sono stati ricevuti dopo il MWN_MESSAGE_UPDATE_SEND_COMPLETE

  #define MWN_MESSAGE_IHAVEALL_PCK **WUh** ;

- messaggio che informa di non avere un aggiornamento disponibile, segue dal messaggio MWN_MESSAGE_UPDATE_START

  #define MWN_MESSAGE_NOREADY_FIR **WUz** ;

- messaggio contiene versione del firmware installato e in memoria

  #define MWN_MESSAGE_SEND_VER_FIR **WUg** .

Elenco degli identificatori di campo:

- campo dati raw

  #define DATA_TYPE_RAW_DATA **r:** ;

- campo dati contenente indirizzo ip

  #define DATA_TYPE_ADDRESS **i:** ;

- campo dati contenente il numero del pacchetto mancante

  #define DATA_TIPE_NEED_PCK **n:** ;

- campo dati per ritrasmissione ack

  #define DATA_TYPE_ENABLE_ACK **a:** .

Vediamo come avviene la trasmissione del firmware da parte del border. Per inizializzare la ricezione del file binario dell'aggiornamento, il border invia il comando **WUr** con all'interno l'informazione della versione del firmware che si sta per inviare e il numero totale dei pacchetti che si invieranno (Fig. 4.11).

I nodi che ricevono tale comando verificano se la versione che sta per essere trasmessa è diversa da quella attualmente installata e se hanno disponibile e valida la stessa versione all'interno della EEPROM esterna. Se il nodo non

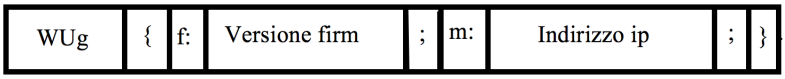

Figura 4.12: Struttura del pacchetto WUg

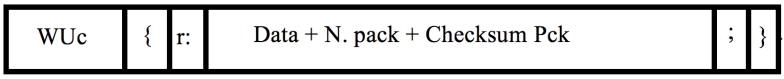

Figura 4.13: Struttura del pacchetto WUc

ha la stessa versione firmware provvederà a cancellare la EEPROM esterna per poter scrivere il file binario dell'aggiornamento. Questo verrà fatto anche se la versione firmware coincide con quella attualmente in uso ma il firmware non è disponibile nella memoria esterna. Se invece coincide in tutte e due le memorie non viene fatto nulla. Il salvataggio nella memoria esterna del firmware, in ogni caso diverso da quello che non sia lo stesso sulla memoria esterna, è dovuto al fatto che quel nodo poi potrebbe essere selezionato per ritrasmettere il firmware, quindi lo deve avere disponibile. Per essere certi dell'avvenuta ricezione da parte di tutti i nodi vicini, il border invia più volte il primo comando. Naturalmente il nodo terrà conto solo del primo utile non ripetendo la formattazione della EEPROM tutte le volte. Terminata l'eventuale formattazione della EEPROM esterna il nodo risponde, sempre in multicast con il comando **WUg** (Fig. 4.12) con all'interno un campo con la sua versione firmware attualmente in uso ed un campo con il suo indirizzo ip, in modo che il border possa iniziare a costruirsi una mappa della struttura della rete con le varie versioni installate in ciascun nodo.

Dopo un determinato numero di invii del primo comando, il border inizia l'invio di tutti i pacchetti che compongono l'aggiornamento. L'intestazione di ogni pacchetto e **WUc** (Fig. 4.13) con all'interno un campo di dati raw da 64 byte più 4 byte di checksum a 32 bit e 4 byte di indice del pacchetto. Grazie alla size fissa del pacchetto e al suo indice non è necessario che tutti i pacchetti vengano ricevuti in ordine dai nodi, ma questi possono essere ricevuti in ordine sparso. Il nodo terrà traccia dei pacchetti arrivati.

Una volta terminato l'invio di tutti i pacchetti che compongono l'aggiornamento il border invia un pacchetto, con header **WUd** (Fig. 4.14), per notificare a tutti i nodi vicini che l'aggiornamento è stato trasmesso tutto.

Alla ricezione di tale comando i nodi verificano di aver ricevuto tutti i pacchetti. Se ne dovesse mancare qualcuno chiederanno, tramite un pacchetto con header **WUp** (Fig. 4.15), la ritrasmissione dei pacchetti mancanti.

Il border, ricevuto il messaggio, provvede a ritrasmettere i pacchetti mancanti

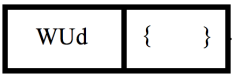

Figura 4.14: Struttura del pacchetto WUd

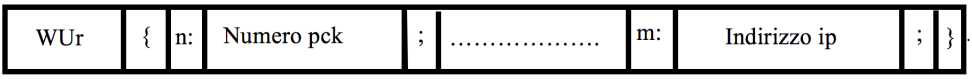

Figura 4.15: Struttura del pacchetto WUp

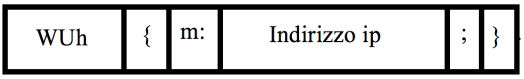

Figura 4.16: Struttura del pacchetto WUh

e, terminata la ritrasmissione, invia nuovamente il comando **WUd** con cui i nodi ripetono la procedura di verifica di eventuali pacchetti mancanti. Se tutti i pacchetti risultano pervenuti il nodo invia un messaggio con header **WUh** (Fig. 4.16) ad indicare la corretta ricezione dell'aggiornamento.

Nello schema in Fig. 4.17 è mostrato il diagrammi a stati per la trasmissione del nuovo firmware da parte de border.

4.3.2 Selezione del nodo per la propagazione dell'aggiornamento

Terminata la trasmissione del firmware da parte del border o di un nodo, il border invia ai singoli nodi che hanno già ricevuto il firmware per l'aggiornamento, quindi in unicast, la richiesta della lista dei vicini. In base al numero dei vicini e del grafo della rete che il border si costruisce sulla base delle informazioni ricevute dai nodi, viene scelto il dispositivo che ha più vicini che necessitano di aggiornamento. Sulla base di queste informazioni il border seleziona i vari nodi da utilizzare come repeater in modo da raggiungere tutti i nodi della rete e con il minor numero di ritrasmissioni del firmware. In Fig. 4.18 lo schema della rete.

Pertanto l'aggiornamento dei nodi non direttamente visibili dal border viene delegato da quest'ultimo ad un nodo che abbia nella tabella dei vicini proprio il nodo da aggiornare. Una volta selezionato, il border invia il comando **WUe** (Fig. 4.19) in unicast al nodo che deve ritrasmettere l'aggiornamento.

Questo quindi invia il comando **WUr** (Fig. 4.20) con all'interno l'informazione della versione del firmware che si sta per inviare e il numero totale dei pacchetti che si invieranno.

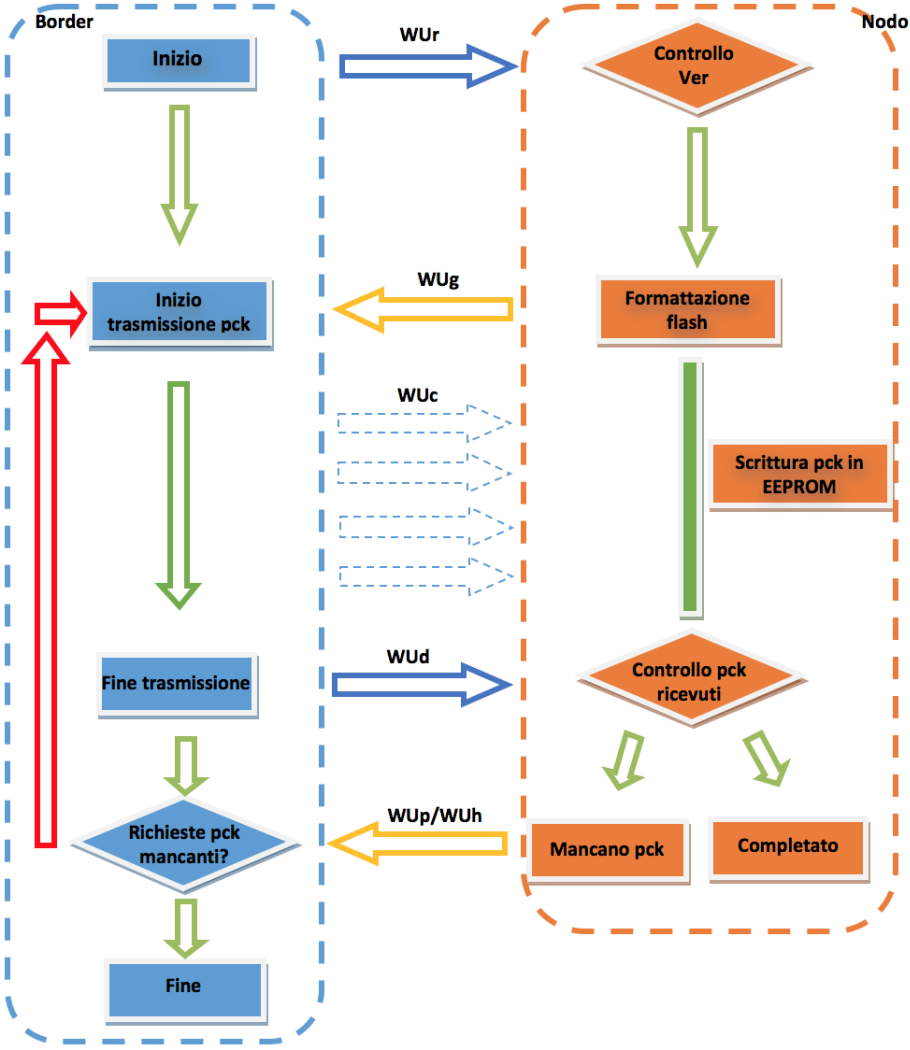

Figura 4.17: Diagramma a stati della trasmissione del Firmware da parte del Border

Come nel caso dell'invio da parte del border i nodi da aggiornare risponderanno con l'informazione della loro versione firmware ed il loro indirizzo. Queste informazioni però non verranno gestite dal nodo che invia l'aggiornamento, ma sarà il border, per motivi legati alle risorse hardware della scheda, a tenere traccia dell'evoluzione degli aggiornamenti. Anche in questo caso il nodo che propaga l'aggiornamento invierà il primo comando di inizio invio del file più volte, per aumentare la possibilità che tutti i nodi vicini lo ricevano e si predispongano alla ricezione del binario. Il nodo inizia quindi la trasmissione di

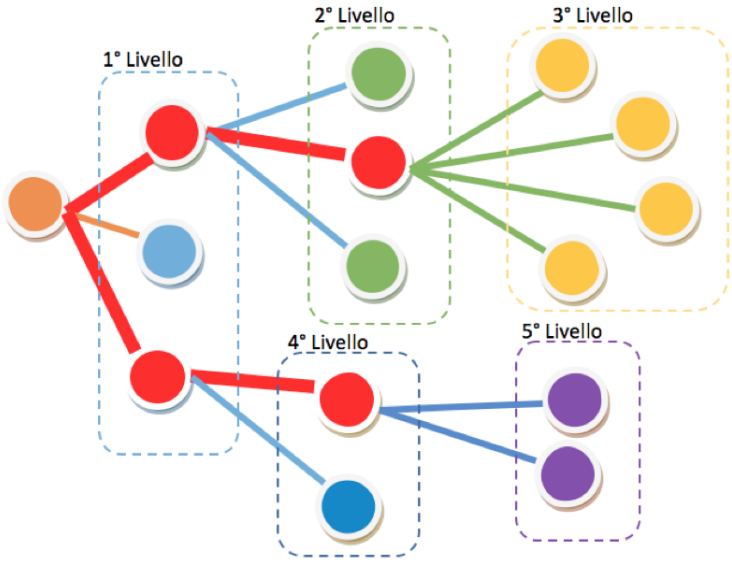

Figura 4.18: Schema per la selezione del nodo repeater durante il FOTA

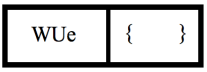

Figura 4.19: Struttura del pacchetto WUe

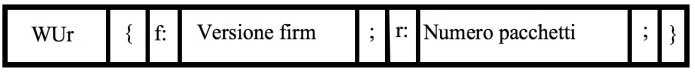

Figura 4.20: Struttura del pacchetto WUr

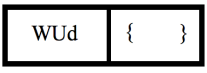

Figura 4.21: Struttura del pacchetto WUd

tutti i pacchetti dell'aggiornamento. Alla fine della trasmissione viene inviato il comando **WUd** (Fig. 4.21) per segnalare ai nodi riceventi che sono stati trasferiti tutti i pacchetti.

Se viene eseguito un determinato numero di invii del comando **WUd** senza che vengano ricevute richieste di pacchetti mancanti da parte degli altri nodi si considera che i nodi abbiano ricevuto tutti i pacchetti dell'aggiornamento. Solo ora viene inviato al border il messaggio **WUo** (Fig. 4.22) per informarlo

Figura 4.22: Struttura del pacchetto WUo

di aver terminato la procedura.

In Fig. 4.23 viene mostrato il diagramma a stati della propagazione dell'aggiornamento da parte di un nodo incaricato dal border.

4.3.3 Avvio della procedura di aggiornamento

Per avviare la procedura di installazione dell'aggiornamento ricevuto, il border invia in unicast il comando **WUs** (Fig. 4.24) come in Fig. 4.25.

Il nodo scrive ad un determinato indirizzo (0x20003FF0) in RAM la stringa LOADER_REQUEST e genera un system reset attraverso il comando *NVIC_SystemReset()*

In questo modo il micro si riavvia dal boot senza perdere i dati presenti nella RAM. Come detto in precedenza la stringa funge da flag per notificare al boot che c'è un aggiornamento pronto per essere installato.

4.4 Interfaccia Desktop per la gestione della procedura di aggiornamento

Il meccanismo di aggiornamento over-the-air del firmware di tutti i nodi della rete può essere comodamente effettuato e gestito tramite un applicativo Python, dotato anche di interfaccia utente grafica mostrata in Fig. 4.26.

L'applicativo può essere lanciato sia direttamente in locale, a bordo del PC o del mini-computer collegato al Border Node (che ha anche il compito di fornire la connettività Internet alla rete, di immagazzinare tutti i dati acquisiti dai nodi, di mantenerli sincronizzati al server remoto, e di permettere l'accesso a varie impostazioni di ogni singolo nodo), tramite l'interfaccia grafica, sia da remoto, tramite l'applicativo Web di gestione della rete (il server remoto instaurerà una connessione con il server locale della rete tramite protocollo http). Nel caso di utilizzo dell'interfaccia (il funzionamento è analogo tramite applicativo Web), il primo passo è la selezione del file *bin* di aggiornamento firmware. Automaticamente l'applicativo andrà a verificare la versione di tale firmware. Successivamente, tramite un semplice bottone è possibile visualizzare la lista di tutti i nodi facenti parte della rete, e potrà essere richiesta loro (in modalità unicast o multicast) l'attuale versione del firmware in esecuzione. A

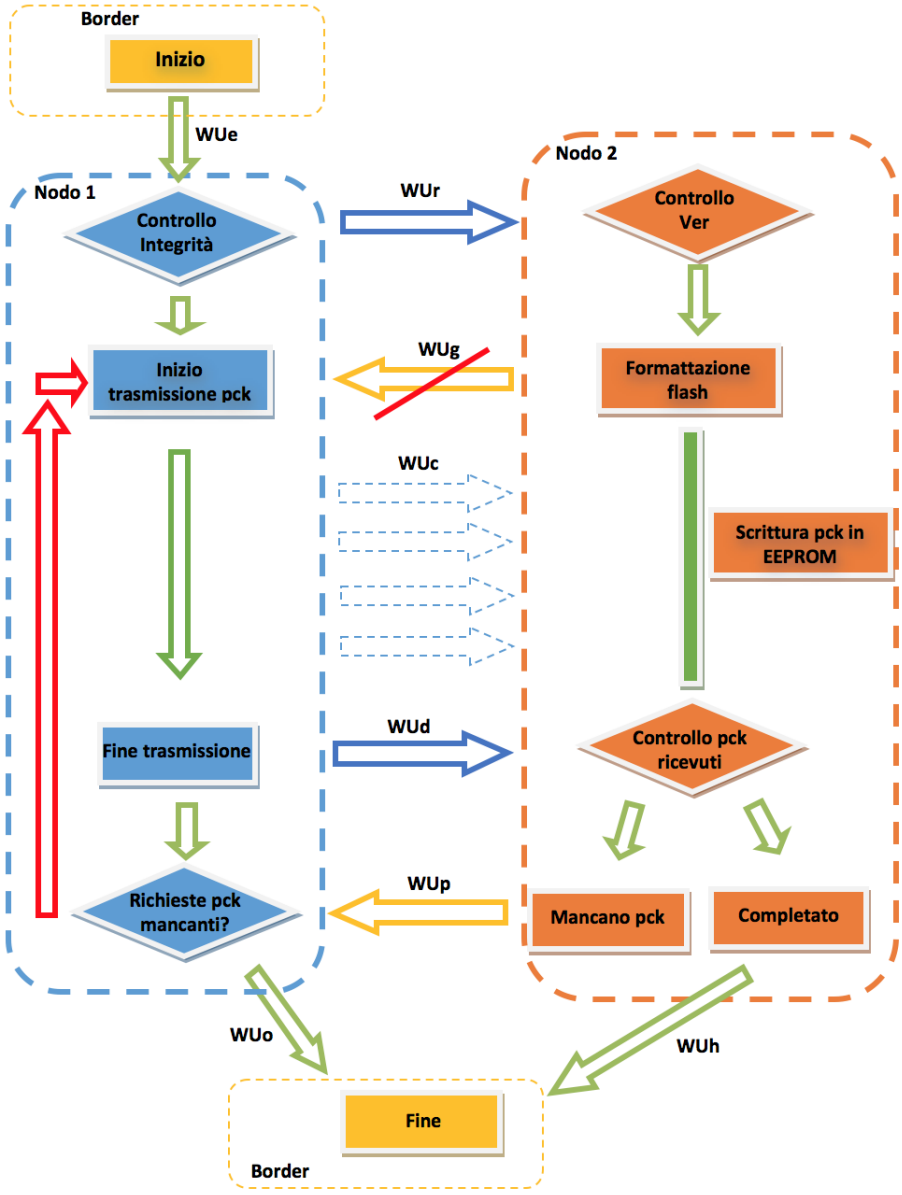

Figura 4.23: Grafo a stati del reinoltro da parte di un nodo dell'aggiornamento

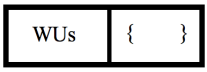

Figura 4.24: Struttura del pacchetto WUs

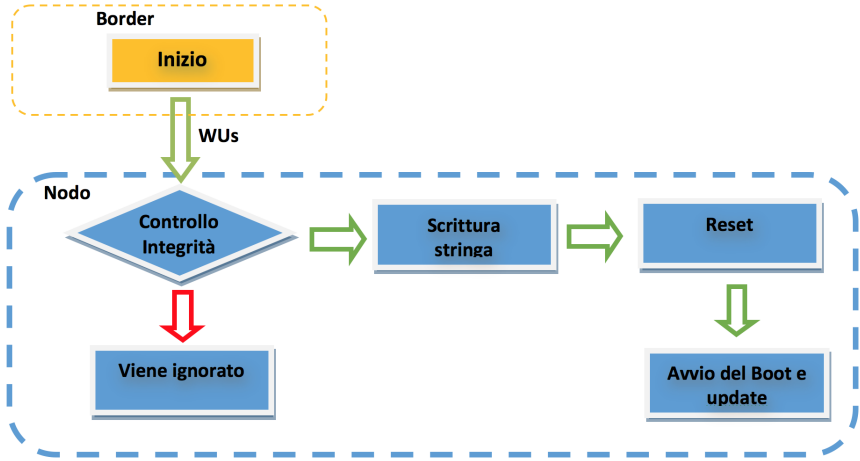

Figura 4.25: Schema di avvio della fase di aggiornamento del nodo in possesso del nuovo Firmware

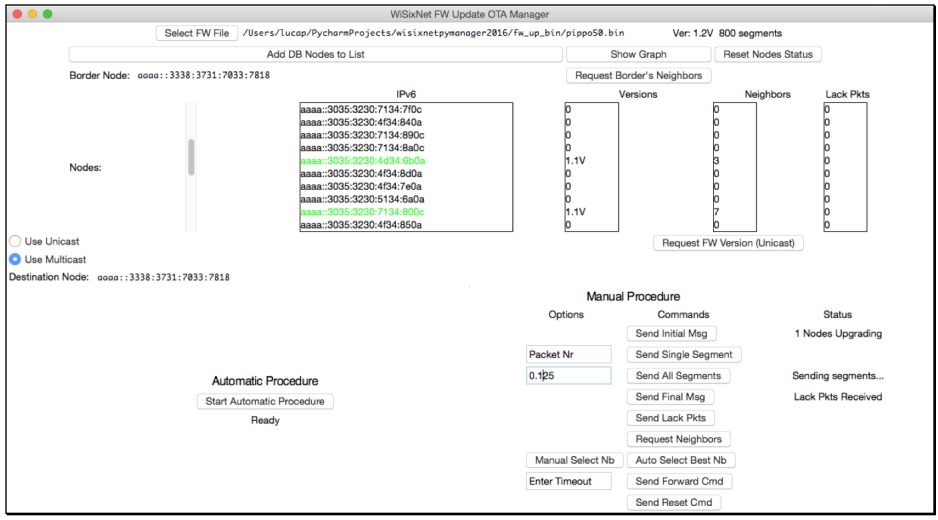

Figura 4.26: Interfaccia Desktop per la gestione del FOTA da PC remoto

questo punto è possibile lanciare la procedura automatica di aggiornamento, o procedere in maniera manuale ad ogni singolo step, ovvero:

- invio messaggio iniziale ed attesa Ack e versione del fw in esecuzione sul singolo nodo;
- invio del singolo pacchetto contenente un segmento di fw o di tutti i pacchetti, con possibilità di scelta della periodicità di invio (default: 5

pkt/s);

- invio del messaggio finale ed attesa di risposta contenente l'informazione del numero di pacchetti mancanti al nodo;
- invio della lista di pacchetti mancanti;
- invio della richiesta della lista dei nodi vicini (in visibilità radio) - unicast;
- invio del comando di reinoltro del fw unicast;
- invio del comando di reset del nodo per il riavvio e l'esecuzione vera e propria dell'aggiornamento unicast;

Dove non specificato la trasmissione può essere sia multicast che unicast. In entrambi i casi sarà possibile visualizzare nell'interfaccia lo stato dell'aggiornamento, e la lista dei nodi verrà tenuta costantemente aggiornata relativamente allo stato in cui si trova ogni singolo nodo. Un'ultima feature resa disponibile dall'applicativo è la visualizzazione in forma grafica della rete mesh, che può mostra tutti i collegamenti tra i nodi in termini di nodi genitori e nodi figli (Fig. 4.27), oppure mostrare tutti i vicini di ciascun nodo (Fig. 4.28). Questa caratteristica può essere molto utile nella procedura manuale per la scelta del nodo designato alla ritrasmissione del firmware, sulla base del numero di nodi neighbors in visibilità radio con il suddetto nodo non ancora aggiornati

4.5 Test e risultati

Le prove sono state fatte per valutare sia la robustezza del procedimento sia i tempi necessari per effettuare l'aggiornamento di più nodi all'interno della rete mesh. Per quanto riguarda la robustezza, il codice scritto e la metodologia descritta nei paragrafi precedenti hanno dato ottimi risultati nelle varie prove, portando sempre a termine la procedura di aggiornamento, o comunque rilevando la presenza di eventuali errori di trasmissione o di pacchetti non pervenuti, notificando così l'errore al border che può provvedere alla ritrasmissione dell'aggiornamento al nodo fino al corretto espletamento della procedura di aggiornamento. Come spiegato in precedenza l'aggiornamento consiste di una fase di trasmissione del file e di una fase di scrittura del file ricevuto in flash. Non essendo necessarie serie di prove per la seconda fase, ci siamo concentrati sulla trasmissione del file, avendo più variabili su cui poter lavorare. La prima è la grandezza del singolo pacchetto dati. Lavorando con 802.15.4 abbiamo una lunghezza massima del pacchetto di 128 Byte, pacchetti più grandi vengono frammentati per poter essere trasmetti. Volendo evitare procedure di frammentazione e riassemblaggio si è scelta la size di 100 Byte per ciascun

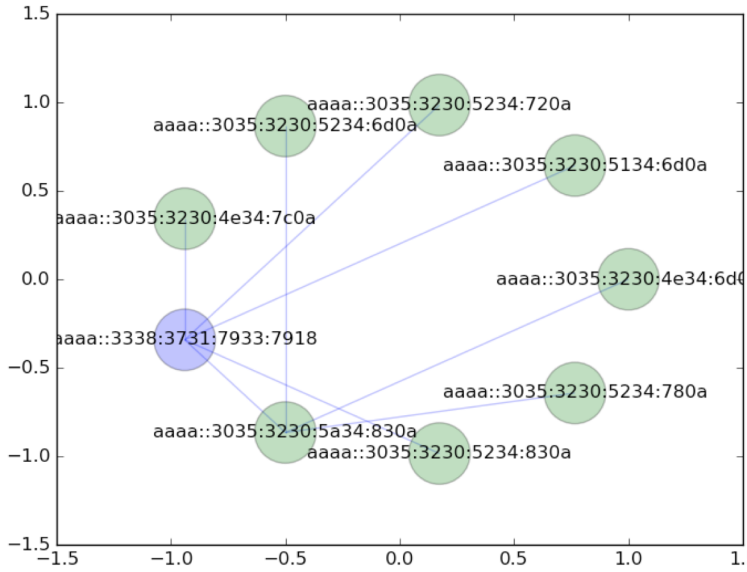

Figura 4.27: Grafo della rete. I collegamenti indicano quali sono i parent di ciascun nodo. Il nodo viola rappresenta il border

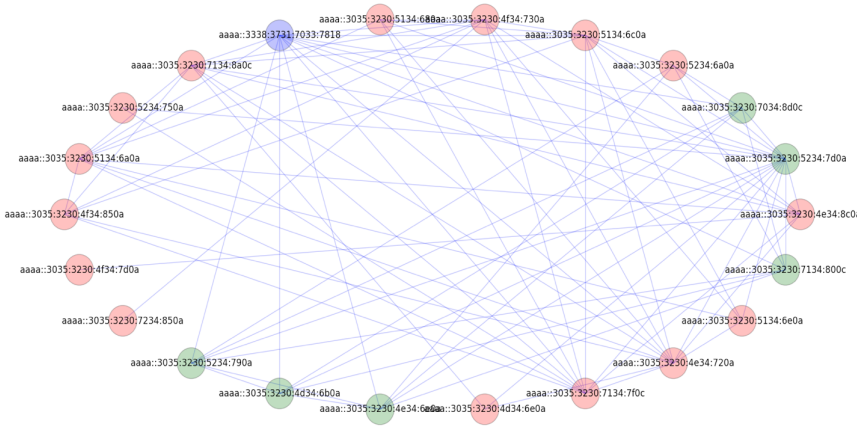

Figura 4.28: Nel grafo sono evidenziati tutti i vicini di ciascun nodo appartenente alla rete

pacchetto. Il secondo parametro indagato è stata la frequenza di trasmissione dei pacchetti. Inviando i pacchetti dati troppo velocemente sulla rete infatti si rischia di congestionare la stessa, di aumentare la probabilità di pacchetti persi e di provocare la perdita di pacchetti di controllo inviati da RPL tramite ICMPv6 con conseguente perdita di rotte. Inoltre, il nodo deve essere in grado

di salvare nella EEPROM esterna il pacchetto ricevuto prima di accettarne un altro. In 6.3 viene mostrata una tabella riassuntiva con il tasso medio di perdita di pacchetti in base alla velocità di spedizione.

Tabella 4.1: La tabella contiene la media della percentuale dei pacchetti persi variando la velocità di trasmissione in termini di pacchetti al secondo, per l'invio del nuovo firmware ai nodi. Risultati ottenuti da 25 test condotti con una numerosità di nodi pari a 10

Pck al secondo	Pck loss rate %
100	70
20	28
10	17
8	3.5
6.7	0.7
5	0.3

Essendo il packet loss rate presente anche nella ritrasmissione dei pacchetti mancanti, in caso di una forte perdita di questi dovranno essere eseguite molte ritrasmissioni, quindi il rischio è quello di vanificare l'elevato packet rate utilizzato, anche perché una ritrasmissione avviene solo dopo un tempo di time out pari a 4 secondi. Di seguito, in Fig. 4.29, viene riportato un grafico con i tempi medi rilevati in seguito a 25 test effettuati per la trasmissione di un intero aggiornamento composto da 800 pacchetti.

4.6 Conclusioni

In questo capitolo si è descritta una procedura per l'aggiornamento da remoto dei nodi sensori appartenenti alla rete mesh IPv6 sviluppata. Dall'analisi delle tipologie di procedure di aggiornamento più comunemente utilizzate abbiamo deciso di utilizzare una EEPROM esterna per potere implementare un metodo robusto ad eventuali errori sia in trasmissione che in scrittura. Per quanto riguarda il trasferimento del file contenente l'aggiornamento si è cercato di sfruttare al meglio le caratteristiche della rete mesh e dell'RPL implementato. Per fare questo si è concepito e realizzato un trasferimento degli aggiornamenti a più livelli, il cui flusso è controllato dal nodo principale, cioè il border, che ha il compito di supervisionare tutta la procedura di aggiornamento. Per rendere il tutto gestibile è stata inoltre creata un'interfaccia grafica di facile utilizzo. Le prove effettuate hanno rilevato la robustezza del sistema e la completa garanzia di successo nella procedura di aggiornamento. Infatti, il sistema è sempre stato

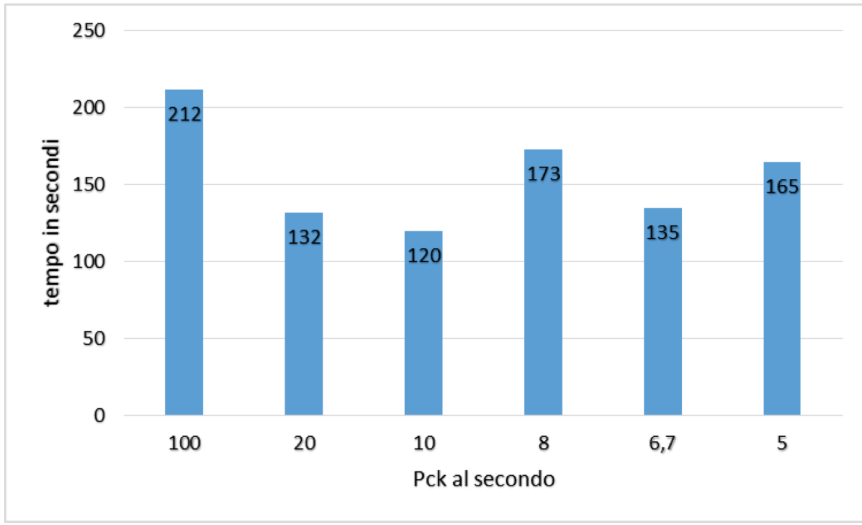

Figura 4.29: Grafico dei tempi di aggiornamento in funzione del Packet rate

capace di individuare eventuali errori e quindi, al massimo, di non procedere con la scrittura dell'aggiornamento in flash ma comunicando al border la presenza di un firmware corrotto nella memoria esterna e procedere ad un secondo invio dei dati.

Capitolo 5

Prototipo di un sistema di localizzazione indoor basato su rete mesh IPv6

Sfruttando il potenziale della rete mesh IPv6 realizzata abbiamo prototipato un sistema destinato alla localizzazione indoor. La localizzazione di dispositivi e persone in ambiente indoor è un argomento molto comune in molti ambiti di ricerca, non a caso esistono molti studi e molte differenti realizzazioni di sistemi finalizzati alla localizzazione indoor; a tal proposito si possono osservare degli studi leggermente datati e riportati in [27], [28], [29], [30], [31], [32], [33], [34], [35] e più recenti riportati in [36], [37], [38], [39], [40].

Il sistema presentato in questo lavoro di tesi presenta dei punti di forza, quali:

- il basso costo del singolo nodo e quindi dell'intero sistema;
- la possibilità di interagire con ciascun oggetto in movimento tramite l'indirizzo IPv6 che lo identifica univocamente;
- per la stima della posizione utilizza un algoritmo di facile implementazione, leggero, testato ed affidabile.

Precedenti lavori realizzati in letteratura non hanno le caratteristiche di interoperabilità e di scalabilità presentate da questo sistema. Scalabilità ed interoperabilità sono garantite, in questo caso, dall'utilizzo della struttura IP, ed in particolar modo dalla scelta di protocolli standard come l'IPv6 o l'802.15.4.

5.1 Tecniche di localizzazione

Tutti i sistemi di localizzazione, siano essi indoor o outdoor, si basano sulla presenza di due dispositivi:

- I nodi mobili che chiameremo da qui in avanti Target; di solito nodi alimentati a batteria, quindi liberi di muoversi senza vincoli all'interno di una certa area. La nomenclatura di Target serve a indicare quel dispositivo come bersaglio di un processo di localizzazione.

- I nodi fissi che vengono comunemente chiamati Anchor. Un nodo Anchor occupa stabilmente una posizione all'interno di una certa area e per questo possono essere alimentati anche mediante collegamento diretto alla rete elettrica. I nodi Anchor fungono da riferimento ai nodi Target, è infatti attraverso di essi che il nodo Target riesce a determinare la sua posizione.

Una tradizionale procedura di localizzazione consta di due fasi principali; la prima fase prende il nome di fase di Ranging, è attraverso questa fase che si realizzano le stime di distanza tra il nodo target ed i nodi anchor. Esistono diverse tecniche per realizzare la stima della distanza, qui di seguito ne vedremo alcune. La seconda fase prende il nome di fase di positioning, è quella fase in cui il target stesso, o un nodo preposto, a partire dalle informazioni di distanza stimata, calcolate in fase di setup, cerca di determinare la sua posizione assoluta, o la posizione assoluta del nodo target di interesse. Per individuare la posizione assoluta, chi realizza le elaborazioni si avvale di opportuni algoritmi di stima della posizione; di seguito ne vediamo alcuni.

5.1.1 Tecniche di ranging

Sono di seguito introdotte e brevemente descritte le principali tecniche di ranging:

- Ranging mediante RSSI.

  Fra le tante tecniche di ranging la più comune e la meno costosa è quella che si basa sull'indicatore RSSI. È la tecnica meno costosa per il semplice motivo che non richiede hardware aggiuntivo. Nel caso delle WSN abbiamo nodi equipaggiati con transceiver radio e oggi ogni transceiver radio dispone di tutto ciò che serve per determinare il valore di RSSI, che viene digitalizzato e memorizzato in un registro. Il valore di RSSI viene direttamente estrapolato dal bit/simbolo ricevuto. Il valore di RSSI ottenuto per opera del transceiver mentre realizza le sue normali operazioni di ricezione è un indicatore della potenza ricevuta a partire da un segnale trasmesso da una sorgente che dista di una quantità d dal ricevitore.

  $$P_R = P_T \frac{G_T G_R \lambda^2}{(4\pi)^2 d^n} \tag{5.1}$$

  dove:

- P_R: potenza del segnale ricevuto in Watt;
- P_T: potenza del segnale trasmesso in Watt;
- G_R: guadagno dell'antenna ricevente;
- G_T: guadagno dell'antenna trasmettente;
- λ: lunghezza d'onda dove c è la velocità della luce e f è la frequenza;
- d : distanza in metri;
- n : costante di propagazione del segnale, dipendente dall'ambiente.

A partire dall'equazione di Friis (5.1), possiamo introdurre delle semplificazioni; ad esempio ponendo GT e GR entrambi ad 1 la relazione sopra si semplifica molto. L'aver posto i due guadagni ad uno equivale a fare un ipotesi semplificativa molto forte, cioè a considerare le due antenne Tx e Rx omnidirezionali ideali.

Dei modelli di propagazione outdoor statici se ne possono prendere le modalità di operare, ad esempio si va a determinare la potenza ricevuta ad una certa distanza dal trasmettitore. Poiché il dato che abbiamo a disposizione è il valore di RSSI al ricevitore, possiamo usare la formula 5.2 per trovare il suo legame la potenza misurata ad una distanza di riferimento e quella misurata alla distanza in cui si trova l'oggetto da localizzare.

$$RSSI = -(10 \cdot n \cdot log_{10}d - A) \tag{5.2}$$

Dopo aver legato l'indice RSSI alla distanza (d) e alla Potenza ricevuta alla distanza di riferimento di 1m espressa in dBm (A), si possono ottenere le relazioni di Formula 5.3 e Formula 5.4 che sono quelle cercate. La relazione di Formula 5.3 ci permette di risalire al valore della distanza d tra trasmettitore e ricevitore noto A, RSSI ed n, dove n è la costante di propagazione dell'onda elettromagnetica che dipende fortemente dalle caratteristiche ambientali. Sempre a partire dalla relazione di Formula 5.2 si può ricavare una relazione per il calcolo della costante di attenuazione n. Per ottenere n sarà necessario ricevere il segnale a distanze prefissate, noti i parametri A ed RSSI.

$$log_{10}d = \frac{A - RSSI}{10 \cdot n} \tag{5.3}$$

$$d = 10^{(\frac{A-RSSI}{10 \cdot n})} \tag{5.4}$$

Abbiamo classificato questa tecnica di ranging come la meno costosa, ma è opportuno sottolineare come la potenza ricevuta in un ambiente indoor

sia estremamente variabile, dipende fortemente dalle caratteristiche dell'ambiente ed inoltre è sensibile alle riflessioni multiple che caratterizzano tale ambiente. Tutto ciò per dire che anche l'indice RSSI è fortemente variabile, quindi le stime di distanza ottenute utilizzando l'RSSI non sono molto accurate.

- Ranging basato su tempo di volo.

  Un'altra tecnica utilizzata per stimare la distanza tra un nodo trasmittente ed un nodo ricevente è quella che si basa sulla misura del tempo di volo dell'onda elettromagnetica nello spazio. Misurato il tempo impiegato per coprire la distanza di separazione tra Tx ed RX e nota la velocità di propagazione dell'onda elettromagnetica si può risalire al valore di distanza semplicemente moltiplicando il dato sul tempo per la velocità di propagazione. È facile comprendere come sia necessario misurare con accuratezza il tempo di volo; un'onda impiega 3,3ns per percorrere un metro, ne consegue che un errore di 1ns sulla stima del ToF (Time of Flight) comporta un errore sulla distanza stimata di 30cm.

  Il sistema GPS, che non può essere usato in ambienti indoor in quanto i segnali provenienti dai satelliti non riescono ad oltrepassare i muri degli edifici utilizza questa tecnica per la stima della distanza. I sistemi UWB invece possono essere utilizzati anche in ambienti ostili come lo è l'ambiente indoor, sono i più precisi ma anche i più costosi. Le tecniche di ranging che si basano sulla misura del ToF sono diverse, ma le più comunemente utilizzate sono la OWR , la TWR e la TDoA. La tecnica OWR(One Way Ranging) richiede che i nodi siano perfettamente sincronizzati; un nodo A invia un segnale all'istante T0 , il ricevitore lo riceve all'istante T1 , data la sincronizzazione tra i due dispositivi si ha che il Tempo di Volo è dato dalla differenza tra i due istanti temporali. La tecnica TWR (Two Way Ranging) misura il tempo che il segnale impiega per arrivare al ricevitore e per tornare al trasmettitore, tale tempo viene comunemente indicato con il nome di RTT (Round Trip Time). Un trasmettitore A invia un segnale all'istante T0 e avvia un timer, il ricevitore lo riceve all'istante T1 e avvia il suo timer. Il ricevitore elabora il segnale e lo ritrasmette in un tempo Treply . Il segnale di risposta giunge al trasmettitore all'istante T2 , a questo punto si può calcolare il RTT come la differenza tra T2 e T0 diminuita della quantità Treply . Ne consegue che il ToF si ottiene dividendo per due il RTT calcolato al passo precedente. La tecnica TWR non richiede il sincronismo fra i nodi; questo è un significativo vantaggio. In Formula 5.5 i calcoli per la determinazione del tempo di volo nella tecnica TWR.

$$T_oF = \frac{(T_2 - T_1) - T_{replay}}{2} \tag{5.5}$$

La tecnica TDoA è particolarmente interessante, si basa sull'utilizzo di nodi anchor sincronizzati e di un nodo mobile non sincronizzato. Il nodo mobile invia un segnale in multicast ad un certo istante temporale, tale segnale verrà ricevuto dai vari anchor con diversi ritardi. I nodi anchor, non essendo sincronizzati con il target, non conoscono quando il target invia il segnale, di conseguenza si fisserà un istante temporale di riferimento per gli anchor che non ha nulla a che vedere con l'istante di emissione del segnale da parte del target. Quello che ci interessa per la stima della distanza non è il tempo di volo assoluto del segnale ma la differenza dei tempi di arrivo, ovvero interessano le differenze tra gli istanti di arrivo del segnale tra i vari anchor. Vengono così stimate le distanze tra i nodi anchor, poi attraverso l'utilizzo di curve iperboliche centrate sugli anchor, si risale alla localizzazione del nodo mobile.

- Ranging basato sull'angolo di provenienza.

  Tali sistemi sono comunemente noti con il nome di AoA (Angle of Arrival). Attraverso questa tecnica abbastanza complessa il nodo ricevitore riesce a discriminare gli angoli di arrivo dei segnali trasmessi dalle stazioni fisse; tale tecnica di ranging ben si presta ad algoritmi di triangolazione per la stima della posizione. E facile comprendere che per implementare questa tecnica sia necessario equipaggiare i nodi con delle antenne direttive o con array di antenne; ne consegue che non è una tecnica facilmente applicabile al contesto delle WSN.

5.1.2 Tecniche di positioning

Sono di seguito introdotte e brevemente descritte le principali tecniche di positioning:

- Positioning mediante Algoritmo Min-Max.

  L'algoritmo Min-Max, noto anche con il nome di Bounding Box, è un algoritmo per la stima della posizione caratterizzato da un'elevata semplicità implementativa, anche se la stima ottenuta in prima interazione può non essere estremamente accurata. L'idea su cui si basa questo algoritmo è quella di determinare la posizione dei nodi come il baricentro dell'intersezione dei quadrati costruiti attorno a ciascun anchor node ed aventi il lato uguale al doppio della distanza stimata fra il nodo incognito e ogni nodo fisso. L'algoritmo Min-Max costruisce un quadrato attorno ad ogni anchor-node a partire dalle coordinate riportate in Formula 5.6 rappresentante la relazione che individua i vertici estremi del quadrato costruito attorno all'anchor-node i-esimo dall'algoritmo Min-Max. x_i, y_i sono le coordinate che individuano la posizione dell'i-esimo anchor-node

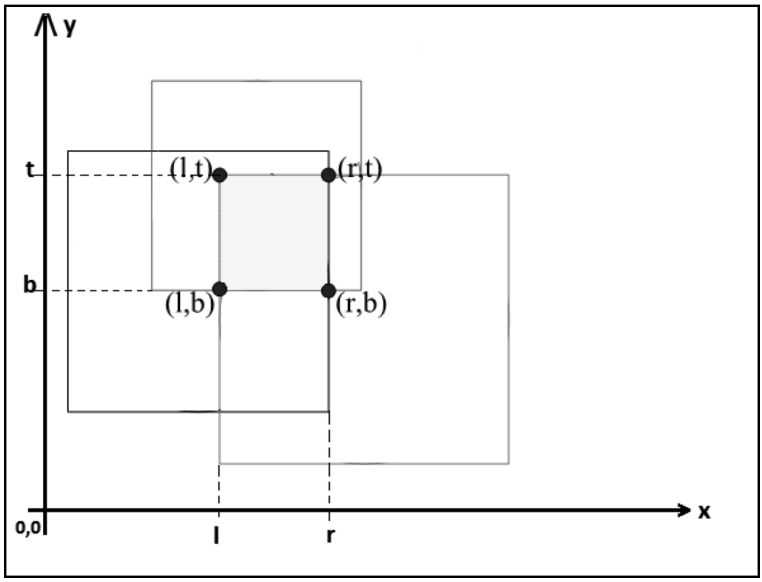

Figura 5.1: Principio di funzionamento del MIN-MAX

mentre di è la distanza stimata tra anchor i-esimo e nodo target ottenuta con una delle tecniche di ranging viste in precedenza.

$$[(x_i - d_i), (y_i - d_i)] \times [(x_i + d_i), (y_i + d_i)] \tag{5.6}$$

Il quadrato si costruisce aggiungendo il doppio di d_i ad x e ad y del vertice inferiore di Formula 5.6 e sottraendo il doppio di di al vertice superiore.

In Fig.5.1 vediamo come opera il Min-Max per la stima della posizione. Si utilizzano tre anchor per la localizzazione di un nodo mobile. Si costruiscono quindi i tre quadrati come descritto in precedenza, i tre quadrati si intersecano costituendo un'ulteriore regione dalla forma di un quadrato. La regione quadrata, costruita intersecando le altre regioni, rappresenta lo spazio con maggior probabilità che sia occupato dal target-mobile. La regione di intersezione è colorata in modo uniforme, essa ha vertici (l,b), (l,t), (r,b) e (r,t). Il Min-Max non si limita a determinare la regione che ha maggior probabilità di essere occupata dal target-mobile, ma stima la posizione del target mobile come il punto mediano di quella regione. L'algoritmo Min-Max produce un errore non-trascurabile sulla stima della posizione anche se l'errore sulla stima della distanza è contenuto.

- Positioning mediante Algoritmo ML.

L'algoritmo ML (Maximum Likelihood) utilizza il criterio di minimizzazione dell'errore quadratico medio (Minimum Mean Square Error) per la stima della posizione del nodo target. Come per tutti gli altri algoritmi di stima della posizione, la fase di positioning viene preceduta dalla fase di ranging attraverso la quale si ottengono le stime delle distanze tra nodi anchor e nodo mobile. Nella situazione bidimensionale si ha bisogno di almeno tre anchor per poter utilizzare questo algoritmo. L'algoritmo ML è molto più complesso degli altri in quanto cerca di minimizzare l'errore della stima della posizione all'aumentare del numero di osservazioni, azzerando la varianza quando questo numero tende all'infinito. Sfortunatamente, negli scenari realistici il numero degli anchor node è limitato, perciò le performance del ML possono essere insoddisfacenti.

- Positioning mediante algoritmo di Trilaterazione.

  Anche questo algoritmo, come il Min-Max, si basa su dei principi geometrici per ottenere la stima della posizione del nodo mobile. Analizzeremo un caso relativo ad una situazione bidimensionale, parleremo quindi di cerchi e non di sfere. Dopo aver realizzato la fase di ranging si conoscano dei valori di distanza. Proprio a partire da questi valori di distanza l'algoritmo di trilaterazione inizia a costruire dei cerchi aventi come centro le coordinate dei nodi anchor e raggio uguale alla distanza stimata nella precedente fase di ranging. Idealmente la posizione del nodo target è rappresentata dall'intersezione dei tre cerchi come in Fig.5.2

  Per calcolare le coordinate del punto di intersezione occorre risolvere il sistema per il calcolo di un punto di intersezione tra tre cerchi. Nella realtà le stime della distanza non sono mai perfette e non viene quasi mai ottenuto un unico punto come intersezione. È più probabile che il nodo target si trovi all'interno di un'area di intersezione tra i cerchi, o è addirittura possibile che i cerchi non si tocchino. Per ovviare a questi inconvenienti si devono utilizzare delle tecniche di miglioramento del tradizionale algoritmo di trilaterazione. Ciò rende l'algoritmo di trilaterazione più complesso del Min-Max, almeno in linea di principio, ma offre performance migliori.

5.2 Il sistema realizzato

Focalizziamo l'attenzione sulle specifiche che ci siamo imposti per la realizzazione del sistema di localizzazione basato su nodi sensore IPv6:

- possibilità di avere un numero elevato di nodi per coprire superfici grandi, visto il target di edifici cui può essere destinato (università, ospedali, grandi aree di stoccaggio merci, ecc...);

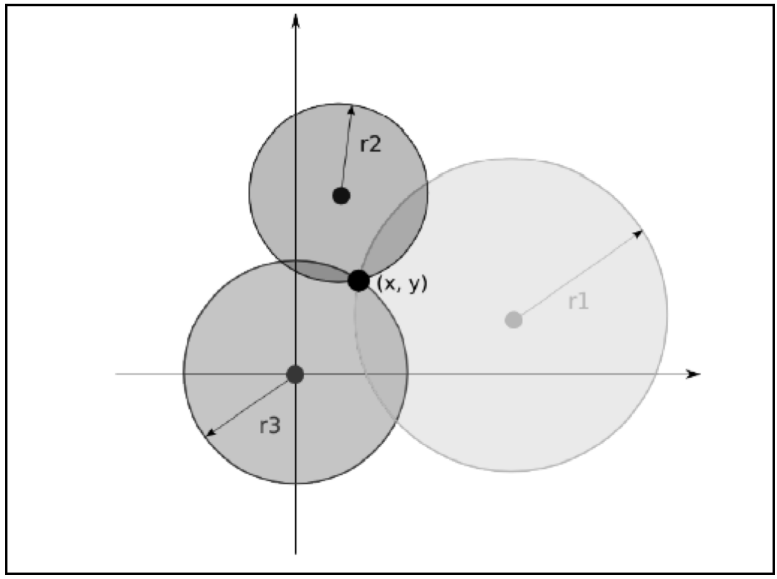

Figura 5.2: Rappresentazione grafica del principio di funzionamento dell'algoritmo di Trilaterazione. Il punto di intersezione tra I tre cerchi rappresenta la posizione stimata del nodo target

- possibilità per l'installazione di essere ampliata/riconfigurata;
- costo per nodo contenuto;
- controllo da postazione remota o da terminale mobile;
- possibilità di prevedere la presenza di nodi alimentati a batteria, quindi consumo energetico ottimizzato.

Dato che l'obiettivo è quello di realizzare un'applicazione di localizzazione indoor si è dovuta fare una scelta in termini di tecnica di ranging utilizzata per stimare le distanze tra nodi mobili e nodi fissi che costituiscono la rete; la scelta è ricaduta sull'indicatore RSSI, al semplice scopo di non complicare il sistema e per mantenere basso il costo, infatti lo Spirit1 determina il RSSI dei pacchetti ricevuti. La localizzazione di un dispositivo ha luogo se oltre alla fase di ranging vi è una fase di positioning che nel nostro caso sarà implementata dal Min-Max. Nella rete realizzata possiamo distinguere tre tipologie di nodi:

- ROOT, ha funzionalità di elaborazione e di gestione della rete;
- ANCHOR, installati in maniera da fornire copertura radio all'intera area di interesse, sono tipicamente alimentati da rete elettrica;
- TARGET dispositivi mobili da localizzare.

Tutte e tre le tipologie di nodi sono realizzate sulla stessa piattaforma hardware, ciò che li differenzia sono le funzioni che implementano ai fini della gestione della procedura di localizzazione. Il nodo ROOT controlla la creazione e la gestione della struttura logica della rete. Nodo ROOT e nodi ANCHOR sono classificati da RPL come nodi MESH; secondo la logica di RPL un nodo mesh è un nodo che è raggiungibile da tutti gli altri nodi della rete ed allo stesso tempo è in grado di smistare i pacchetti verso gli altri nodi quando non sono a lui diretti. Contrariamente i nodi TARGET sono stati configurati come nodi LEAF; secondo RPL un nodo leaf è un nodo che è raggiungibile da tutti gli altri nodi della rete e non presenta funzionalità di routing dei pacchetti a lui non destinati.

Il nodo ROOT può fungere da border router nell'eventualità in cui si colleghi la rete mesh IPv6 locale alla rete internet globale. Ad ogni nodo ANCHOR sono state assegnate delle coordinate che ne identifica la posizione su di un sistema di riferimento opportunamente scelto in riferimento all'area di copertura del nostro sistema di localizzazione. Vediamo ora il ruolo che ha ogni diversa tipologia di nodo nel sistema di localizzazione realizzato. Nel nodo ROOT è presente un codice applicativo che si occupa di dialogare con i nodi TARGET connessi alla rete e di memorizzarli in una apposita tabella dei target vivi. Il nodo ROOT sta in attesa di ricevere un messaggio chiamato Target Alive da parte di un nodo TARGET che viene avviato e risulta connesso alla topologia logica della rete, cioè connesso al così detto DODAG-RPL. Il messaggio Target Alive viene inviato da un nodo target appena è alimentato e appena conosce la rotta per recapitare il messaggio al root; un messaggio target alive può essere di nuovo inviato dal medesimo target nel caso in cui, per motivi dovuti a riorganizzazione della rete, il target perde e riacquisisce la rotta per il nodo Root. L'invio del messaggio da parte del target viene ripetuto periodicamente ogni 30 secondi fino a che non riceve conferma dal root mediante l'invio di un messaggio di acknowledgment. Il root utilizza il messaggio target alive per memorizzare in una tabella dei target l'indirizzo IPv6 del target che ha segnalato di essere vivo.

Nel nostro sistema prototipale si fa in modo che appena il nodo root aggiunge un nuovo target alla tabella dei target vivi inizia ad inviare un messaggio detto richiesta di localizzazione. La procedura di invio ripetuto di tale messaggio viene bloccata non appena il target, oggetto della richiesta di localizzazione, risponde al root con un messaggio di ack. A questo punto il root rimane in attesa dei dati utili per la localizzazione, settando un flag che impedisce l'avvio di una nuova procedura di localizzazione fintantoché ne è in corso un'altra. Il processo è continuativo i quanto vogliamo un tracciamento costante della posizione del target. Il target dopo aver inviato l'ack alla richiesta del root fa partire trenta pacchetti udp con indirizzo multicast con una velocità di quattro

pacchetti al secondo che saranno intercettati dagli anchor in visibilità radio.

Ogni anchor che riceve i multicast può utilizzarli per la determinazione di un valore di RSSI medio. Il calcolo della media serve a compensare eventuali fluttuazione dell'indice RSSI. Ogni Anchor coinvolto nel processo di localizzazione impacchetta le informazioni riguardo al suo indirizzo IPv6, al valore medio dell'indice RSSI, al numero dei campioni sui quali è stato calcolato RSSI medio, alla potenza ricevuta alla distanza di un metro, alla costante di attenuazione e alle le sue coordinate x ed y per inviarle al ROOT. La potenza alla distanza di un metro è la potenza che l'anchor node ha captato da un target di test utilizzato in fase di installazione. Alla fine della procedura di installazione ogni anchor calcola la costante di attenuazione che gli compete. Tutto ciò serve a realizzare una caratterizzazione del canale radio. Il ROOT riceverà questi pacchetti e provvederà a stimare la distanza per tutto il set di anchor che partecipano alla localizzazione, a questo punto può essere richiamata la procedura che implementa l'algoritmo di localizzazione Min-Max. Nell'ipotesi che il root abbia ricevuto questi pacchetti informativi da almeno tre anchor diversi la funzione che implementa il Min-Max può essere richiamata perché è potenzialmente in grado di restituire dei dati validi per il positioning del nodo target. Nel caso in cui il root non riceva un numero sufficiente di pacchetti informativi la funzione che implementa il Min-Max non viene invocata, ma il root riavvia la procedura che invia il messaggio richiesta di localizzazione al target e la procedura ha di nuovo inizio.

Sul root è stato definito un timer di controllo che si preoccupa di far in modo che una procedura di localizzazione non duri complessivamente più di 45 secondi. L'introduzione di questo timer serve in realtà ad ovviare all'inconveniente che si può manifestare nel caso in cui nessun anchor partecipi al processo di localizzazione. Scaduto questo timer il root controlla che almeno un pacchetto di informazioni da un qualche anchor sia arrivato, se la risposta è affermativa verifica che il numero degli anchor da cui ha ricevuto pacchetti siano il numero minimo utile a far si che la localizzazione possa aver luogo, se la risposta è negativa si riavvia la procedura di localizzazione. Prima di riavviare la procedura di localizzazione tutte le strutture dati vengono azzerate. Il risultato ottenuto, cioè le coordinate (x,y) che individuano la posizione assoluta del nodo target viene visualizzato nella mappa importata nell'interfaccia grafica realizzata e descritta nei prossimi paragrafi.

Il sito di prova è stato scelto in modo che avesse requisiti simili, il più possibile, ad un futuro scenario applicativo del sistema di localizzazione quale può essere un ospedale, un grande magazzino per stoccaggio merci o un'università; abbiamo così deciso di installare la nostra rete mesh IPv6 per il testing del nostro sistema di localizzazione all'interno del Dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche. I nodi Anchor sono

stati disposti per la maggior parte lungo le pareti perimetrali del dipartimento in modo da coprire tutta la superficie utile. La disposizione scelta per questi nodi ci ha permesso di realizzare una sperimentazione del sistema su di un'area complessiva di circa 1000mq.

5.3 Procedura di installazione

Particolare attenzione è stata dedicata alla caratterizzazione del canale wireless che rappresenta un aspetto essenziale per il buon funzionamento del sistema di localizzazione, in quanto da essa dipendono i risultati della stima della distanza. Caratterizzare il canale radio significa determinare la sua costante di propagazione. Determinare la costante di propagazione in un ambiente indoor non è cosa banale, vi sono un'infinità di contributi da tenere in considerazione. Un metodo è quello di utilizzare le costanti di attenuazione ottenute da modelli statistici ma spesso il modello ottenuto si discosta in maniera importante rispetto al comportamento reale e quindi tutto il sistema di localizzazione non avrebbe fondamenta solide.

L'altra alternativa è quella di determinare la costante di attenuazione del canale wireless per via sperimentale cioè facendo delle misure direttamente nello ambiente indoor per il quale ci interessa determinare la costante di attenuazione. Per risalire ad n si è pensato bene di posizionare l'antenna ricevente a distanza nota dal trasmettitore in modo che l'unica incognita sia proprio n. Per ogni valore di distanza noto si va a valutare l'indice RSSI, otterremo un diverso valore di n per ogni punto; la costante di attenuazione non può avere un diverso valore per ogni valore di distanza ma deve essere un valore che descrive la propagazione in un determinato contesto a tutte le distanze. Come prima cosa si è definito un set di distanze di riferimento dove si andranno a realizzare le misure di RSSI e qundi si risalirà al valore di n per ogni valore di distanza. Il set delle distanze di riferimento utilizzato è il seguente: 1, 2, 3, 4 e 5metri.

Così facendo ottengo una caratterizzazione del canale per punti, se vogliamo una caratterizzazione migliore l'alternativa sarebbe ripetere la procedura per un numero maggiore di punti o adottare delle tecniche di interpolazione quali, interpolazione mediante cubica, interpolazione mediante spline o interpolazione mediante minimizzazione dell'errore quadratico; il problema di questa tecnica è che parte comunque da pochi valori ed il valore di n ottenuto con più valori rispecchierebbe quello ottenuto con pochi. Abbiamo scelto di calcolare la costante di attenuazione media sulla base dei cinque valori ottenuti alle distanze di riferimento. Il calcolo di n avviene a partire dal valore in dBm dell'indice RSSI, abbiamo sottolineato più volte come in un contesto indoor-dinamico tale indice sia per sua natura rapidamente variabile allora si realizza l'acquisizione di almeno 25 valori di RSSI per ogni distanza di riferimento. A partire da

questi 25 valori si calcola un indice medio che poi viene utilizzato per calcolare la costante di attenuazione relativa a quella distanza di riferimento. La ragione per cui il nodo target invia trenta pacchetti in multicast agli anchor, serve a far in modo che ogni anchor calcoli un indice RSSI medio su un numero di pacchetti non inferiore a venticinque.

Per automatizzare la procedura di installazione si è definito un nodo "target for installation" che contiene un processo dedicato, tale processo viene avviato dall'utente mediante pressione del push-button. Dato che si deve far in modo che l'anchor acquisisca dei valori di RSSI a distanze prefissate la prima cosa da fare è posizionare l'anchor node nella posizione definitiva che occuperà all'iterno del sistema di localizzazione, occorre poi posizionarsi con il target a debita distanza.

Il push-button viene schiacciato dall'operatore non appena ha assunto la posizione ad un metro di distanza (prima distanza di riferimento) dal nodo anchor che si sta per installare. Appena il processo viene avviato il nodo target invia, alla frequenza di 4 pacchetti al secondo, un totale di 160 pacchetti con indirizzo multicast, nella migliore delle ipotesi tali pacchetti possono essere tutti ricevuti dal nodo anchor in questione che provvederà ad estrarre, per ogni pacchetto, il valore di RSSI. Il nodo anchor realizza un controllo sul totale dei pacchetti ricevuti ogni volta che ne riceve uno nuovo, non appena ne ha ricevuti 50 realizza delle operazioni di calcolo della media e della deviazione standard sui valori di RSSI associati. La media e la deviazione standard vengono utilizzati per ripulire l'array dei valori di RSSI salvati, in modo da eliminare quei valori che si discostano dalla media in eccesso o in difetto di una quantità pari alla deviazione standard. Dopo la ripulitura avremo un array di valori ridotto ma ripulito da valori anomali che possono condizionare il calcolo della media. Se il numero dei valori di RSSI all'interno dell'array ripulito è maggiore o uguale a 25 si passa a calcolare la nuova media che verrà memorizzata come valore di potenza ricevuta alla prima distanza di riferimento ed inoltre si va a settare un flag che permette l'invio di un messaggio di ack da parte dell'anchor al target di installazione. Una volta che il target riceve questo messaggio considerata andata a buon fine l'acquisizione dei valori di RSSI alla prima distanza di riferimento, quindi permette all'utente di continuare la procedura di installazione segnalandolo all'utente tramite il led integrato nell'Hardware.

Nell'ipotesi che la procedura sia andata a buon fine l'utente si sposta alla seconda distanza di riferimento dal nodo anchor per poi schiacciare di nuovo il push-button che da inizio alla procedura di acquisizione dei valori di RSSI alla seconda distanza di riferimento, e così fino alla quinta ed ultima distanza di riferimento. Alla quinta distanza di riferimento il target controlla che il pacchetto di ack dell'anchor sia stato ricevuto, in caso affermativo invia all'anchor un pacchetto di 'fine installazione' che permette all'anchor di settare un flag

ed accendere un led in modo definitivo, dando un feed-back all'utente sullo stato dell'installazione. Il flag settato serve a far in modo che d'ora in avanti il nodo anchor installato non processi più pacchetti multicast di questo genere, che possono venir captati mentre si realizzerà l'installazione di altri anchor. Il messaggio di fine installazione serve all'anchor per richiamare quella procedura che calcola la costante di attenuazione come media dei valori di n calcolati in relazione alle distanze di riferimento.

5.4 Interfaccia multifunzione di installazione e gestione

Tutte le procedure di installazione e gestione della rete di sensori IPv6 sono state implementate in un'interfaccia web che consente l'immediata e facile interazione dell'utente con i dispositivi. Nella Fig.5.3 è riportata la schermata di visualizzazione dello scenario di lavoro del sistema di localizzazione.

La prima operazione da effettuare è quella di caricare la mappa del piano sul quale vogliamo lavorare attraverso l'appositá funzionalità messa a disposizione. Fatto questo, ogni qual volta viene acceso un nuovo nodo IPv6, esso viene riconosciuto dal border che invia una notifica in remoto, questo permette la visualizzazione sull'interfaccia di un nuovo dispositivo che inizia a lampeggiare in attesa che gli vengano assegnate la posizione sulla mappa semplicemente trascinando il nodo oppure inserendo manualmente le coordinate x,y in metri rispetto ad un punto preso come riferimento e la tipologia di sensore che rappresenta (Border, Anchor, Sensor Node generico, Target). Schiacciando il tasto salva la configurazione viene memorizzata e da quel preciso momento è possibile interagire da remoto con l'oggetto. Se ad esempio si vuole chiedere la posizione in cui si trova un target sarà sufficiente selezionarlo ed in pochi secondi si vedrà l'oggetto spostarsi sulla mappa in funzione della posizione stimata tramite gli algoritmi descritti. Oltre alla funzione di localizzazione esiste la possibilità di interrogare i nodi equipaggiati con sensori, ad esempio temperatura, umidità, ecc. . . , e chiedere in tempo reale il valore misurato, oppure visualizzare l'ultimo valore inviato o lo storico dei valori acquisiti.

5.5 Risultati

Le prove del sistema di localizzazione basato su rete mesh IPv6 sono state svolte all'interno del Dipartimento di Ingegneria dell'Informazione (DII) dell'Università Politecnica delle Marche. In Fig.5.4 riportiamo la planimetria dell'intero dipartimento e mettiamo in evidenza il sistema di riferimento fissato. La scala della Planimetria è 1:400, adottata solo per esigenze di spazio. Attraverso

Figura 5.3: Interfaccia di installazione e servizio per la rete di sensori IPv6 con modulo per l'applicazione di localizzazione indoor

l'interfaccia descritta nel paragrafo 5.4 abbiamo installato la rete di test con 9 anchor ed il border node ad abbiamo inserito un solo target fatto spostare in un certo numero di punti di misura differenti, per ogni punto di misura il sistema realizza in automatico 50 prove consecutive di localizzazione per un totale di 550 prove effettuate in 11 diverse posizioni. Tutte le prove sono state ripetute con la stessa potenza di trasmissione di 0dBm.

In Fig.5.5 è riportato un esempio di dispositivo target da localizzare ottenuta dalle prove effettuate.

Mediando tutti i risultati ottenuti dai 550 test effettuati si ottiene un errore medio nella localizzazione pari a 0.98m ed un errore massimo di 3,12m.

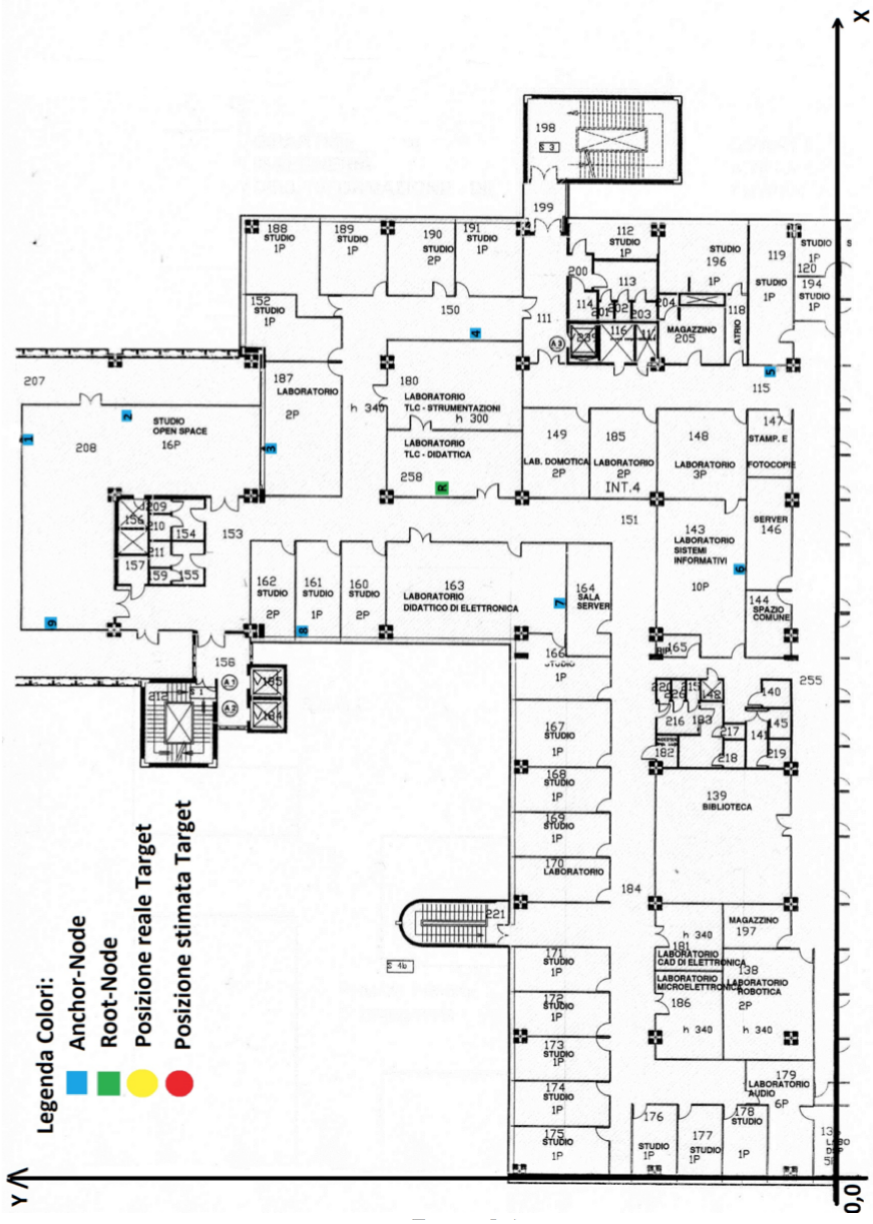

Figura 5.4: Mappa del sistema di test installato

5.6 Conclusioni

L'obiettivo di realizzare una prima forma prototipale di un sistema di localizzazione indoor basato su una WSN-IPv6 è stato raggiunto. Una demo è stata installata ed è funzionante all'interno del Dipartimento di Ingegneria dell'In-

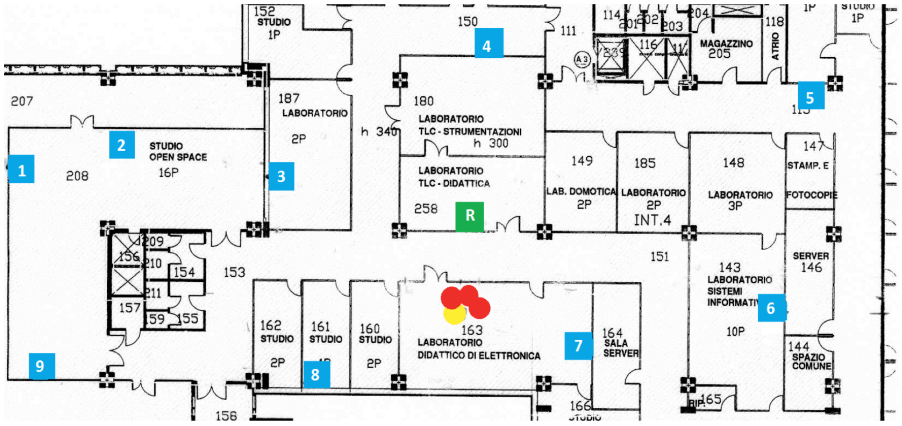

Figura 5.5: Esempio di localizzazione del nodo target (giallo) e dei risultati del processo di localizzazione ottenuti dagli 11 test nella stessa posizione (bollini rossi)

formazione dell'Università Politecnica delle Marche. Il sistema realizzato, visti i risultati conseguiti su errore medio e massimo nella stima della localizzazione, ci ha dato conferma che una tecnica di localizzazione indoor basata sul semplice indice RSSI è possibile e restituisce dei risultati in linea con gli obiettivi di accuratezza prefissati anche in relazione all'hardware a basso costo e al basso consumo energetico. Non era fra gli obiettivi la realizzazione di un sistema di localizzazione di precisione paragonabile ai moderni sistemi UWB, visti anche gli strumenti low-cost scelti per lo sviluppo.

La possibilità di utilizzare la rete ed interagire con quest'ultima attraverso l'interfaccia sviluppa ne consente un facile utilizzo e permette l'estensione dell'applicazione ad ampi scenari, grazie anche alla modularità che permette di caricare e rimuovere mappe, sensori e scenari applicativi.

Capitolo 6

Applicazioni e risultati

In questo capitolo descriveremo le diverse applicazioni implementate che sfruttano la rete mesh IPv6 sviluppata. Essendo il sistema molto flessibile ed adattabile ai diversi scenari, vedremo che molteplici e diversificati sono i possibili campi di utilizzo della tecnologia proposta. Fra questi troviamo il mondo dell'agricoltura intelligente, quello dell'Ambient Assisted Living e quello delle smart cities l'efficientamento energetico. In questi settori sono stati sviluppati prototipi di sistemi o dispositivi industriali veri e propri che verranno brevemente descritti nei seguenti paragrafi con lo scopo di dimostrare l'effettiva applicabilità nel mondo commerciale del sistema proposto. La seconda parte del capitolo è dedicata ai risultati ottenuti dalle varie installazioni e dai test effettuati al simulatore Cooja.

6.1 Agricoltura di precisione

L'agricoltura si trova sempre più ad affrontare degli scenari economici, sociali e ambientali in rapida evoluzione che la obbligano a individuare innovazioni tecnologiche volte a mettere a punto sistemi colturali a basso impatto ambientale e a costo ridotto, attraverso l'impiego di strumenti per il controllo automatico della distribuzione di tutti i fattori di produzione, con particolare riguardo ai potenziali inquinanti (fertilizzanti e fitosanitari). Serve inoltre innovazione nei processi con conseguenti incremento della produttività del lavoro e riduzione dei costi di produzione. Sistemi smart porterebbero anche ad una gestione attenta della tracciabilità dei prodotti e favorirebbe forme di certificazione di qualità. Il tutto al fine di ottenere un'agricoltura sostenibile in termini ecologico-ambientali senza per questo diminuire la profittabilità economica, anzi incrementarla. Le proposte scientifiche e commerciali della cosiddetta agricoltura di precisione cercano, appunto, di soddisfare, a vari livelli, tutti questi obiettivi.

Sfruttando la rete mesh IPv6 sviluppata è stato messo a punto un prototipo per il monitoraggio dei parametri che influiscono sulla crescita e sulla qualità dei prodotti agricoli. Nel dettaglio i sensori sono stati utilizzati per l'acquisizione

di grandezze quali la temperatura, sia dell'aria che del suolo, l'umidità del terreno, il ph, il livello di illuminazione che arriva al suolo e sulle piante. La temperatura può essere utilizzata per prevenire danneggiamenti alle colture causati da bruschi cali o incrementi termici, inoltre, in combinazione con il livello di umidità, risulta utile per gestire sistemi di irrigazione automatica e tarata in funzione dell'effettiva necessità di acqua che le coltivazioni richiedono nelle varie zone del terreno. Il ph può essere utilizzato per programmare le tipologie di colture da mettere a dimora e rappresenta un utile indicatore per programmare le fertilizzazioni. La misura del livello di illuminazione va ad influire sul sistema irriguo e permette di stimare i diversi livelli di maturazione nelle varie aree agricole. Oltre a sistemi di automazione intelligente dei processi ed indipendenti dal lavoro e dalla presenza umana, la raccolta in remoto di numerose grandezze misurate in maniera diffusa nei terreni agricoli permette di avere feedback circa i metodi migliore di coltivazione in relazione alla qualità e alla produttività conseguita in determinate aree piuttosto che in altre.

Il prototipo realizzato è stato presentato e mostrato al fuori EXPO della regione Marche a Milano nel 2015.

6.2 Efficientamento energetico e Smart Cities

Quello delle Smart Cities rappresenta un settore in fortissima espansione ed in continua evoluzione. Questo rappresenta un impulso determinante verso la ricerca di nuove soluzioni e soprattutto di tecnologie ad alto contenuto innovativo come la rete sviluppata nell'ambito di questo progetto. Altro aspetto che favorisce l'affermazione della rete mesh IPv6 proposta nell'ambito delle Smart Cities è l'interoperabilità con altri sistemi basati si tecnologia IP, questo rende il sistema scalabile ed integrabile anche con dispositivi prodotti da terze parti.

Nel dettaglio si è realizzato un sistema basato su hardware e firmware contenente lo stack protocollare sviluppato che viene installato, in collaborazione con aziende del settore energetico ed elettrico, all'interno dei quadri della pubblica illuminazione per monitorare:

- consumi energetici;
- corrente istantanea consumata da ciascuna linea di lampioni;
- stato di funzionamento dei lampioni;
- eventuali anomalie sui magnetotermici;
- interventi di manutenzione nel quadro;
- atti di vandalismo o allacci abusivi;

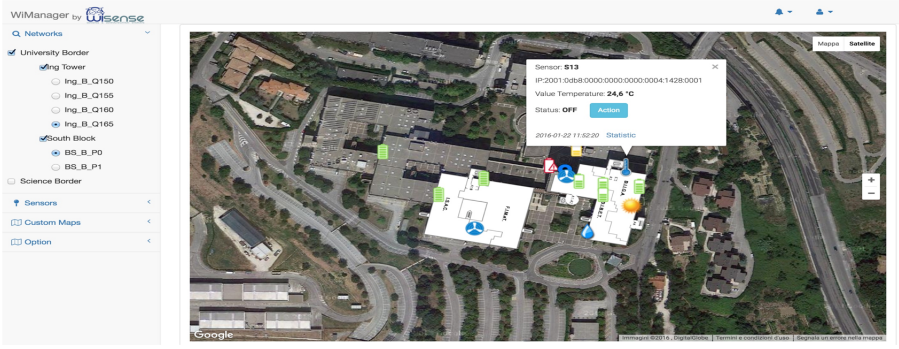

Figura 6.1: Interfaccia web di gestione della rete mesh IPv6

Per questo particolare progetto è stato anche integrato il protocollo MODBUS all'interno del sistema operativo scelto in quanto il campionamento delle grandezze fisiche viene effettuato da slave MODBUS collegati all'hardware su cui è installato lo stack protocollare proposto. Tale sistema ha portato inoltre allo sviluppo di un'interfaccia web di monitoraggio basata sulle API di Google Maps in grado di consentire l'interazione con la rete e i dispositivi, di facilitare l'installazione della rete con la possibilità di assegnare una posizione geografica univoca ai nodi e di consultare gli storici dei consumi o i dati in tempo reale. L'interfaccia web è riportata in Fig.6.1.

Oggi il sistema è installato nel Comune Toscano di Stia e si stanno valutando le estensioni ad altri comuni.

6.3 Ambient Assisted Living

Un ulteriore settore in cui l'architettura protocollare sviluppata sta trovando applicazione è quello dell'Ambient Assisted Living. Il settore risulta in forte crescita per via dell'innalzamento dell'età media della popolazione e la richiesta di smart object che aiutino sia la persona nella vita quotidiana, che i familiari ed il personale medico nell'assistenza prestata a chi è più debole. Questo ha portato all'incremento di tecnologia in modo particolare a casa degli assistiti con la creazione di veri e propri ambienti di vita intelligenti capaci, autonomamente, di interagire con gli inquilini e migliorarne la qualità della vita.

L'integrazione di molteplici sensori ha tratto consentito di sfruttare a pieno le potenzialità della rete mesh sviluppata. È stato creato un ambiente di prova all'interno del Dipartimento di Ingegneria dell'Informazione dell'UNIVPM che ha permesso di analizzare le prestazioni del sistema [41]. Partendo dai lavori sulle WSN sviluppati all'interno del gruppo di ricerca si sono integrati i seguenti sensori:

- controllo luminosità ambientale;
- sensori di presenza nelle varie stanze [42];
- sensori per la rilevazione della caduta e la segnalazione in remoto [43];
- sensori per il monitoraggio cardiaco [44];
- sensori per il monitoraggio di pazienti con la malattia di Parkinson [45].

Nel complesso si sono installati 40 dispositivi di cui 10 mobili per la rilevazione delle cadute e per l'analisi dei pazienti parkinsoniani. I dati raccolti dalla rete di sensori sono resi disponibili nel cloud e l'accesso è possibile grazie alla medesima interfaccia web di Figura 6.1, questo a dimostrare la modularità e duttilità di quest'ultima. La potenza di trasmissione di ciascun elemento è impostata a 0 dBm ed i nodi sono programmati per operare in modalità di risparmio energetico con RDC abilitato. Essi scambiano solo i pacchetti strettamente necessari per garantire il mantenimento della rete in modo da avere una risposta immediata nel caso di un evento anomalo come la caduta. Direttamente dall'interfaccia web possiamo monitorare in tempo reale lo stato di funzionamento dei nodi e notificare la possibile perdita delle rotte verso i sensori e quindi l'impossibilità di instradare correttamente i pacchetti. A livello di trasporto usiamo UDP con un meccanismo di acknowledgment della corretta ricezione a livello applicazione al fine di garantire la corretta ed immediata ricezione laddove si verifichi un allarme caduta.

L'ambiente di test è stato anche utilizzato per valutare le prestazioni della rete in condizioni reali, i dettagli sono riportati nel Paragrafo 6.5.

6.4 Risultati ottenuti con il simulatore Cooja

Il simulatore Cooja è risultato uno strumento molto utile nel percorso di sviluppo ed integrazione dei protocolli; ha consentito di testare diverse configurazioni di rete e di verificare passo a passo gli upgrade fatti al progetto nel corso degli anni. In funzione delle caratteristiche del nodo simulato (piattaforma hardware reale o native) si possono testare diversi parametri, quelli analizzati nel nostro caso sono stati: configurazione della rete con relativi tempi di setup, analisi dei pacchetti persi, analisi dei consumi energetici, studio delle metriche per la costruzione della rete con la selezione del parent.

Molto interessanti sono stati i risultati ottenuti con la simulazione di un numero crescente di dispositivi connessi. Qui, infatti, abbiamo ottenuto che la rete, senza limiti di risorse e memoria imposti dall'hardware reale, può interconnettere oltre i 1000 nodi ma in linea teorica infiniti. Simulando, invece, le caratteristiche del nostro hardware, riusciamo a collegare fino a 60 dispositivi

allo stesso root, il limite à dato dalla modalità di funzionamento *storing* scelta per RPL, la quale porta alla saturazione della memoria disponibile sul nodo root.

6.4.1 Tempi di setup

L'utilizzo del simulatore ci consente di valutare le performance della rete in presenza di un collegamento radio ideale, senza le innumerevoli variabili introdotte dalla propagazione in ambiente reale. Per valutare il tempo di setup della rete assumiamo che la rete sia configurata non appena il root ha memorizzato tutte le rotte verso i nodi appartenenti al suo DAG. Per la valutazione del tempo di autoconfigurazione della rete abbiamo effettuato cinque test utilizzando rispettivamente 3, 10, 20, 30, 40 nodi sensori e, naturalmente, il border node (root). In questa particolare prova la variabile più importante è rappresentata dalla numerosità dei nodi e dal numero di device che il singolo sensore deve attraversare affinché il messaggio possa raggiungere il root. Per questo motivo abbiamo creato una topologia di rete in cui, al massimo, un messaggio, proveniente dal dispositivo più lontano deve compiere 6 salti prima di arrivare a destinazione. In Tab.6.1 sono riportati i risultati conseguiti per le differenti numerosità di nodi.

Tabella 6.1: Tempi di configurazione della rete al variare dei nodi ottenuti con il simulatore Cooja

Numero Nodi	3	10	20	30	40
Tempo (s) per il setup della rete	3	5	9	11	15

Per l'esecuzione del test si è utilizzata una configurazione del nodo che non prevedeva il protocollo RDC e come metrica è stata utilizzata l'ETX per la scelta dei parent.

6.4.2 Pacchetti persi

Per la valutazione del numero di pacchetti persi sono stati effettuati cinque test utilizzando rispettivamente 3, 10, 20, 30, 40 nodi sensori in aggiunta al nodo root. In condizioni ideali, ovvero senza la presenza di ostacoli che disturbino la propagazione radio il test darebbe un risultato del tutto scontato ed inutile, in quanto si avrebbe il 100% di pacchetti recapitati correttamente. Per questo motivo abbiamo creato uno script che simula la presenza di un ostacolo modellato come un armadio in metallo interposto fra ciascun nodo ed i suoi

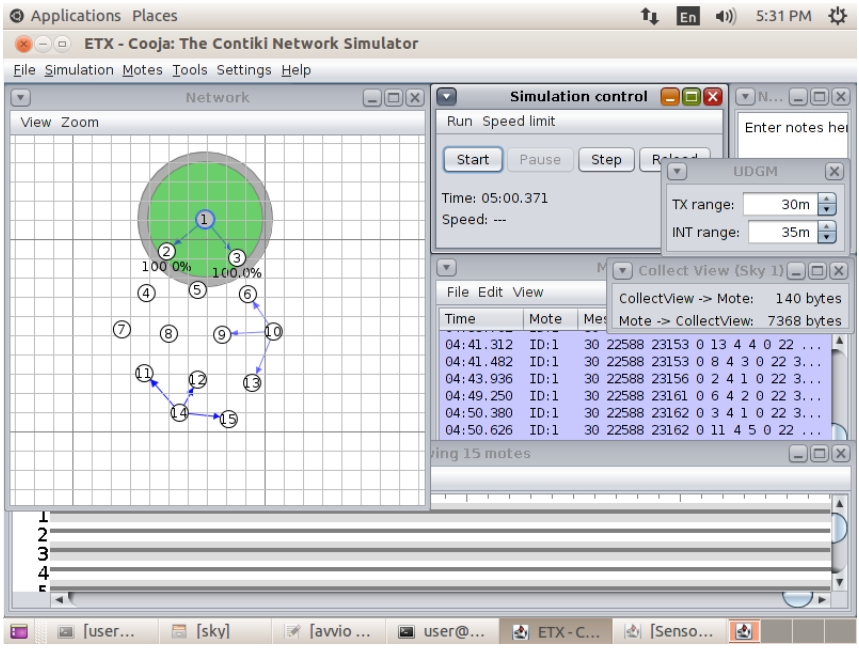

Figura 6.2: Posizionamento dei sensori nell'interfaccia di Cooja

vicini. Si è, inoltre, introdotta una variabile che simula lo spostamento random di un ulteriore ostacolo in maniera continua all'interno della rete, così da creare situazioni impreviste che costringono i nodi a cambiare parent rinegoziando le rotte. In Tab.6.2 sono riportati i risultati conseguiti per le differenti numerosità di nodi, ciascuna simulazione ha avuto una durata di 48 ore.

Tabella 6.2: Risultati del test sui pacchetti correttamente ricevuti. Nella tabella sono riportati i risultati medi fra le 12 prove effettuate per ciascuna numerosità di nodi

Numero Nodi	3	10	20	30	40
% di pacchetti ricevuti correttamente	99,98	99,84	99,67	99,28	99,19

6.4.3 Consumi energetici

Per la valutazione dei consumi energetici di ciascun nodo e per valutare le prestazioni del protocollo RDC si è effettuato un test con 15 nodi posizionati come mostrato in Fig.6.2.

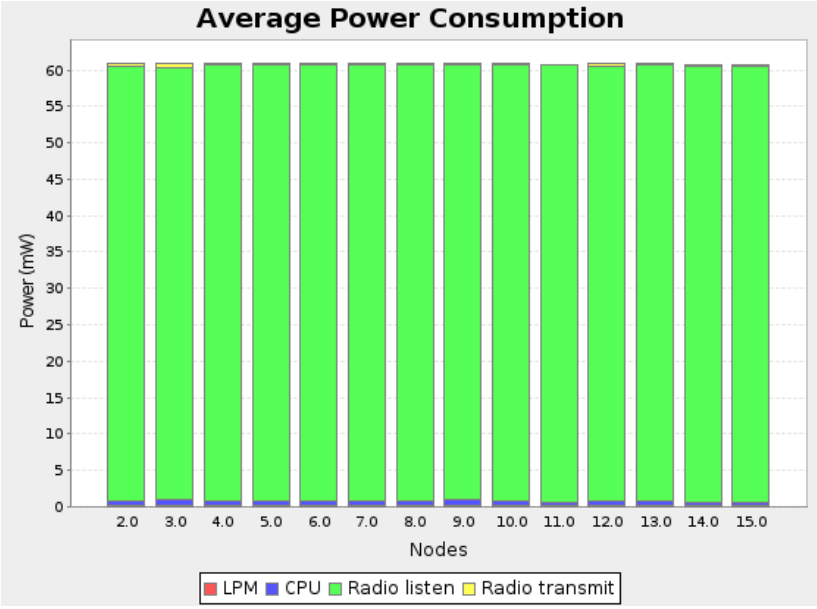

Figura 6.3: Risultati della simulazione senza protocollo RDC

Il primo test è stato effettuato selezionando la modalità *null_rdc* per il driver RDC. Come evidenziato in Fig.6.3 la scelta di non utilizzare politiche di ottimizzazione della gestione della radio comporta un consumo energetico inaccettabile nella realtà dove tipicamente i sensori sono alimentati da batterie che devono durare anni. E' importante notare che, in assenza del driver RDC il transceiver viene mantenuto costantemente in ascolto del canale radio e questo comporta dei consumi esagerati per la fase di ricezione, tanto che i contributi dovuti alla trasmissione e alla CPU diventano del tutto trascurabili.

Verificata la necessità di utilizzare protocolli RDC si è passato al test di quest'ultimo utilizzando la stessa configurazione di rete precedente. Da Fig.6.4 vediamo che, in presenza del driver RDC i consumi complessivi, per la stessa configurazione di rete, si riducono mediamente del 90%, mentre la potenza dissipata per la sola fase di ricezione scende sotto l'1% del valore impiegato senza politiche di gestione dello sleep della radio.

6.4.4 Metriche per la selezione del parent

La configurazione a 15 nodi di Fig.6.2 è stata utilizzata anche per la valutazione della formazione della rete e della scelta del parent al variare delle metriche utilizzate in RPL per la configurazione del grafo. Essendo in simulazione abbiamo anche in questo caso inserito un disturbo, questa volta costante, fra i vari link, in modo tale da far configurare la rete in maniera conforme alla

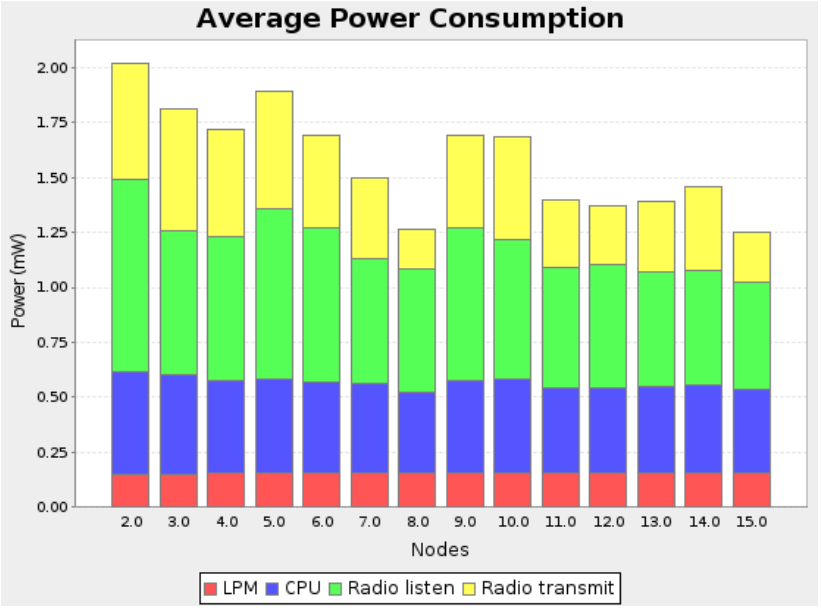

Figura 6.4: Risultati della simulazione con protocollo RDC

metrica scelta. Il primo test è stato effettuato utilizzando la metrica ETX che ha prodotto il grafo in Fig.6.5. Il secondo test è stato effettuato utilizzando la metrica ENERGY che ha prodotto il grafo in Fig.6.6.

6.5 Risultati installazione reale

Come ambiente di test abbiamo utilizzato il Dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche come descritto nel Paragrafo 6.3.

La rete IPv6 è stata testata per valutare statistiche sulla perdita dei pacchetti e sui tempi di set-up all'avvio. Entrambe le caratteristiche sono direttamente legate alla numerosità dei nodi. Infatti, l'aumento del numero di dispositivi causa un aumento della complessità dell'instradamento all'interno della rete e della creazione delle tabelle di routing. Abbiamo effettuato cinque test utilizzando rispettivamente 3, 10, 20, 30, 40 nodi sensori e, naturalmente, il border node. Le prove hanno avuto una durata di 48 ore e sono state ripetute per 12 volte. Per analizzare le performance in termini di pacchetti persi, i test sono stati effettuati durante il weekend in cui non vi era movimento di cose e persone nell'ambiente tale da alterare i risultati o, comunque, introdurre variabili non facili da gestire. Abbiamo scelto di attivare i dispositivi in ordine di distanza dal più vicino al più lontano dalla posizione del nodo confine. Dalla creazione

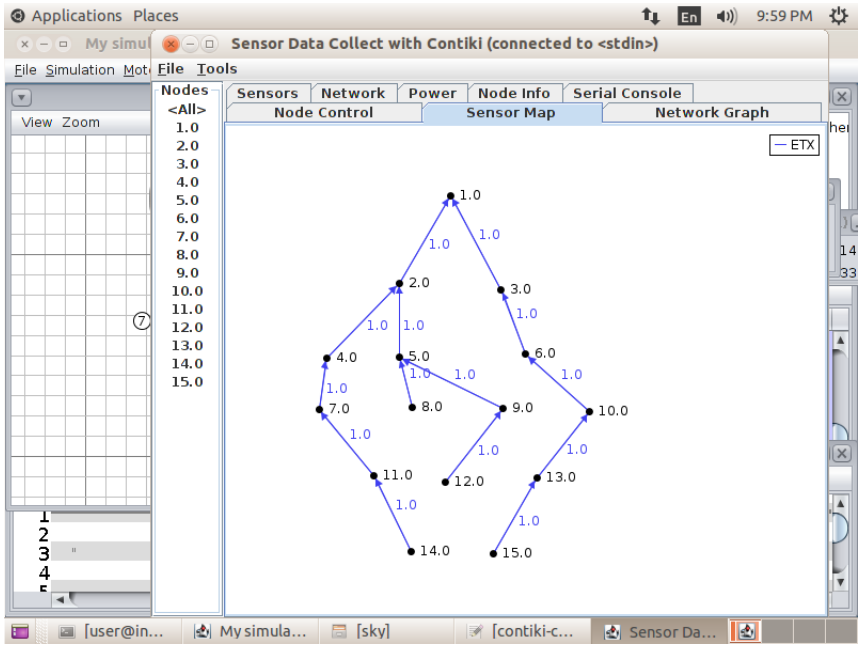

Figura 6.5: Grafo ottenuto con la metrica ETX

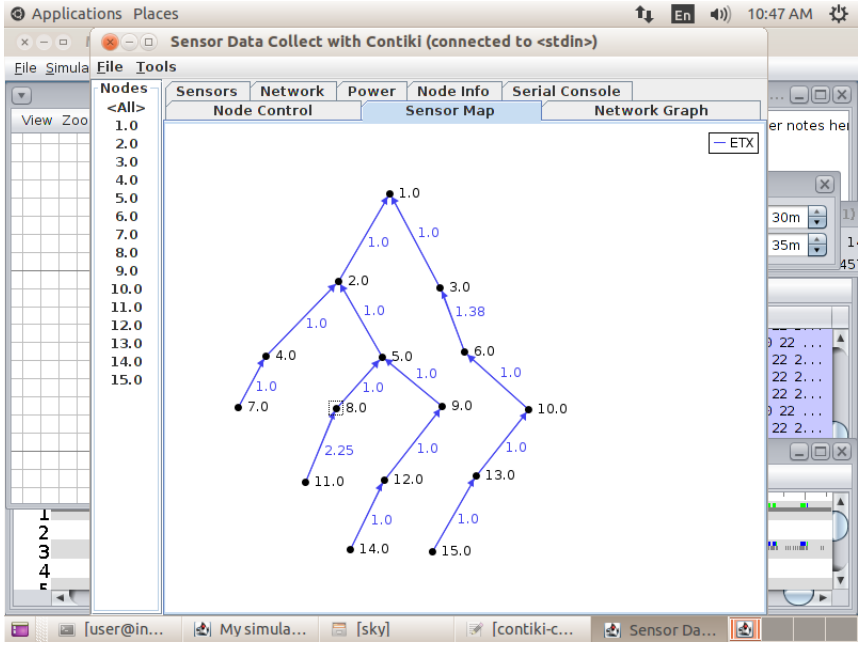

Figura 6.6: Grafo ottenuto con la metrica ENERGY

delle rotte per il border ciascun nodo ha inviato 1 pacchetto ogni 30 secondi. Come protocollo di trasporto è stato utilizzano l'UDP senza alcun meccanismo di riscontro a livello applicativo. I dati in Tab.6.3 mostrano la media dei risultati di tutte le prove.

Tabella 6.3: Risultati del test sui pacchetti correttamente ricevuti. Nella tabella sono riportati i risultati medi fra le 12 prove effettuate per ciascuna numerosità di nodi

Numero Nodi	3	10	20	30	40
% di pacchetti ricevuti correttamente	99,90	99,70	99,44	99,13	98,94

È importante notare che una rete di questo tipo nasce per l'invio di pochi dati intervallati da lunghi periodi, pertanto il packet rate di 30s scelto risulta piuttosto impattante sul sistema, ma proprio questo era l'obiettivo dei test. I risultati confermano che le prestazioni e l'affidabilità dipendono dal numero di nodi. Vista la presenza di pacchetti persi abbiamo testato con lo stesso setup un meccanismo di ack e di ritrasmissione in caso di pacchetto perso. Come ci aspettavamo il tasso di pacchetti ricevuti dai nodi aumenta raggiungendo il 100%. Per quanto riguarda il tempo di setup ed installazione della rete, la prova viene eseguita mentre si accendono tutti i dispositivi in contemporanea attraverso un messaggio inviato in multicast a tutti i nodi coinvolti nel test. Anche in questo caso prova è stata effettuata per 12 volte con una nemerosità di sensori, rispettivamente di 3, 10, 20, 30 e 40 dispositivi. I risultati riportati nella Tab.6.4 sono i tempi medi di setup ottenuti dalle 12 prove effettuate per ciascun numero di sensori. Il setup si considera concluso quando il border node ha acquisito i percorsi per tutti i nodi della rete.

Tabella 6.4: Tempi medi di configurazione della rete

Numero Nodi	3	10	20	30	40
Tempo (s) per il setup della rete	3	6	12	17	25

Come indicato dai risultati, all'aumentare del numero di nodi aumenta la complessità della configurazione e il tempo di setup aumenta in maniera quasi lineare con l'incremento dei dispositivi connessi. Questo potrebbe costituire una limitazione in applicazioni che prevedono un numero molto elevato di sensori ed è un fenomeno da studiare ed ottimizzare durante i successivi lavori.

Oltre ai test sulla rete sono state testate anche le funzionalità delle applicazioni attraverso l'installazione permanente. In primo luogo va detto che l'applicazione web ha permesso di rendere operativa la rete di 40 sensori in tempi di circa 1 ora; gettando uno sguardo oltre la ricerca verso il mondo industriale questo rappresenta un notevole risparmio sui costi di installazione. L'interfaccia web ed il database online ci hanno consentito di analizzare i dati raccolti relativamente agli ultimi quattro mesi di funzionamento.

Dalla loro analisi si è dedotto che il sistema installato risulta molto stabile permettendo di avere un riscontro immediato in caso di allarme (meno di 2 secondi di latenza) e ci ha anche permesso di creare uno storico dei parametri ambientali per studiare le abitudini delle persone che vivono in un determinato ambiente. Per quanto riguarda il consumo di energia durante i quattro mesi di attività della rete, le batterie da 800mAh che alimentavano i dispositivi mobili non sono mai state sostituite e alla fine hanno indicato un livello di carica (misurata attraverso il STC3115 STMicroelectronics installato sulla scheda di espansione) pari al 18%. Nei prossimi lavori si effettueranno prove specifiche volte ad ottimizzare ulteriormente i consumi energetici.

Conclusioni

In questo lavoro di tesi si è descritta un'architettura di rete basata su topologia mesh e protocolli alla base dell'Internet globale. L'approccio congiunto fra mondo della ricerca e mondo industriale ha da un lato permesso di fornire connotazioni innovative al lavoro svolto, dall'altro ha dettato delle linee guida alla base dell'utilizzo del sistema in applicazioni commerciali. Il carattere innovativo è dato dalla scelta di protocolli di nuova concezione all'interno dello stack protocollare, come ad esempio RPL nato appositamente per la gestione del routing all'interno delle WSN, quindi in presenza di limitate risorse energetiche e di calcolo e di fronte alla possibilità di perdere pacchetti lungo la rete o dalla possibilità di scegliere il CoAP fra i protocolli applicativi per la raccolta dati dai sensori. Inoltre l'innovazione e la ricerca sono alla base di tutte le applicazioni sviluppate, in quanto nate proprio da un'analisi dello stato dell'arte di ciascun settore e dallo sviluppo di algoritmi innovativi, come nel caso della rilevazione della caduta o nel monitoraggio di tremore e freezing nei pazienti parkinsoniani, sviluppati all'interno del gruppo di ricerca.

L'impronta industriale è stata data nella scelta della frequenza di lavoro nella banda SubGHz nell'ottica di limitare l'impatto degli ostacoli sul segnale radio e di evitare la saturazione della banda a 2,4GHz, nella scelta di mantenere estremamente contenuti costi e dimensioni dell'hardware sviluppato per favorirne la diffusione, nella creazione dell'interfaccia web multifunzione che possa essere utilizzata sia in fase di installazione dai tecnici, che in fase di monitoraggio dagli utenti. Altro aspetto non trascurabile nato nel mondo della ricerca, ma che offre ricadute anche in quello industriale è il protocollo di FOTA sviluppato e che consente l'aggiornamento del firmware dei nodi direttamente da remoto. Oltre ad avere un approccio innovativo, lo sviluppo di questa feature consente a chi utilizza il sistema di installare i dispositivi anche in luoghi difficilmente accessibili senza più la preoccupazione di doverli raggiungere per il fix di un bug o per l'aggiornamento degli applicativi.

Questa stretta sinergia ha prodotto come risultato una tecnologia estremamente innovativa, che perfettamente si presta ad essere utilizzata nel mondo sempre più in espansione dell'Internet of Things. Lo sviluppo delle applicazioni descritte nei precedenti capitoli sono esempi concreti della vasta gamma di settori e prodotti che potrebbero beneficiare di quanto sviluppato nel corso di questi anni di Dottorato.

Molti degli obiettivi sono stati raggiunti: miniaturizzazione, interoperabilità, funzionamento della rete mesh con un numero di sensori limitato a 60 nella realtà per via dei limiti di memoria dell'hardware e ai 1000 nelle simulazioni, autoconfigurazione e mantenimento automatico della rete. Altri aspetti vanno ancora perfezionati ed approfonditi, in particolare il futuro lavoro si concentrerà sul miglioramento delle prestazioni energetiche così da prolungare il tempo di vita delle batterie fino a diversi anni.

Bibliografia

[1] Joseph M Kahn, Randy H Katz, and Kristofer SJ Pister, "Next century challenges: mobile networking for "smart dust"," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 271–278.

[2] Kevin Ashton, "That 'internet of things' thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[3] Ping Song, Chang Chen, Kejie Li, and Li Sui, "The design and realization of embedded gateway based on wsn," in *Computer Science and Software Engineering, 2008 International Conference on*. IEEE, 2008, vol. 4, pp. 32–36.

[4] Hong-jiang He, Zhu-qiang Yue, and Xiao-jie Wang, "Design and realization of wireless sensor network gateway based on zigbee and gprs," in *2009 second International Conference on Information and Computing Science*. IEEE, 2009, vol. 2, pp. 196–199.

[5] Hongyu Chu, Zhijiang Xie, Yanhua Shao, Qin Liu, and Zengzhen Mi, "Design and implement of wsn based on bluetooth and embedded system," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*. IEEE, 2010, vol. 5, pp. V5–641.

[6] Mirko Franceschinis, Claudio Pastrone, Maurizio A Spirito, and Claudio Borean, "On the performance of zigbee pro and zigbee ip in ieee 802.15. 4 networks.," in *WiMob*, 2013, pp. 83–88.

[7] Muhammad Amjad, Muhammad Sharif, Muhammad Khalil Afzal, and Sung Won Kim, "Tinyos-new trends, comparative views, and supported sensing applications: A review," *IEEE Sensors Journal*, vol. 16, no. 9, pp. 2865–2889, 2016.

[8] Tuhin Borgohain, Uday Kumar, and Sugata Sanyal, "Survey of operating systems for the iot environment," *arXiv preprint arXiv:1504.02517*, 2015.

[9] Yi-Hua Zhu, Kaikai Chi, Xianzhong Tian, and Victor CM Leung, "Network coding-based reliable ipv6 packet delivery over ieee 802.15. 4 wireless

personal area networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 4, pp. 2219–2230, 2016.

[10] Joseph Polastre, Jason Hill, and David Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 95–107.

[11] Amre El-Hoiydi and Jean-Dominique Decotignie, "Wisemac: An ultra low power mac protocol for multi-hop wireless sensor networks," in *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*. Springer, 2004, pp. 18–31.

[12] Michael Buettner, Gary V Yee, Eric Anderson, and Richard Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006, pp. 307–320.

[13] Chieh-Jan Mike Liang, Andreas Terzis, et al., "Koala: Ultra-low power data retrieval in wireless sensor networks," in *Proceedings of the 7th international conference on Information processing in sensor networks*. IEEE Computer Society, 2008, pp. 421–432.

[14] Yanjun Sun, Omer Gurewitz, and David B Johnson, "Ri-mac: a receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008, pp. 1–14.

[15] Adam Dunkels, "The contikimac radio duty cycling protocol," 2011.

[16] Chen Yibo, Kun-mean Hou, Haiying Zhou, Hong-ling Shi, Xing Liu, Xunxing Diao, Hao Ding, Jian-Jin Li, and Christophe De Vaulx, "6lowpan stacks: a survey," in *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*. IEEE, 2011, pp. 1–4.

[17] Meenakshi Gupta, "Ipv4 vs. ipv6," 2010.

[18] Gary Malkin, "Rip version 2-carrying additional information," 1994.

[19] John T Moy, *OSPF: anatomy of an Internet routing protocol*, Addison-Wesley Professional, 1998.

[20] Tim Winter, "Rpl: Ipv6 routing protocol for low-power and lossy networks," 2012.

[21] Jean-Philippe Vasseur, Mijeom Kim, Kris Pister, Nicolas Dejean, and Dominique Barthel, "Routing metrics used for path calculation in low-power and lossy networks," Tech. Rep., 2012.

[22] Pedro Diogo, Luís Paulo Reis, and Nuno Vasco Lopes, "Internet of things: A system's architecture proposal," in *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2014, pp. 1–6.

[23] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt, "Cross-level sensor network simulation with cooja," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, 2006, pp. 641–648.

[24] Joakim Eriksson, Fredrik Österlind, Niclas Finne, Nicolas Tsiftes, Adam Dunkels, Thiemo Voigt, Robert Sauter, and Pedro José Marrón, "Cooja/mspsim: interoperability testing for wireless sensor networks," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 27.

[25] Dennis K Nilsson and Ulf E Larson, "Secure firmware updates over the air in intelligent vehicles," in *ICC Workshops-2008 IEEE International Conference on Communications Workshops*. IEEE, 2008, pp. 380–384.

[26] Moshe Shavit, Andy Gryc, and Radovan Miucic, "Firmware update over the air (fota) for automotive industry," Tech. Rep., SAE Technical Paper, 2007.

[27] Vladimir Savic, Adrián Población, Santiago Zazo, and Mariano García, "An experimental study of rss-based indoor localization using nonparametric belief propagation based on spanning trees," in *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*. IEEE, 2010, pp. 238–243.

[28] Yeong-Sheng Chen, Tai-Lin Chin, and Yi-Chen Huang, "Collaborative localization in wireless sensor networks based on dependable rssi," in *Wireless Personal Multimedia Communications (WPMC), 2012 15th International Symposium on*. IEEE, 2012, pp. 341–347.

[29] Emanuele Goldoni, Alberto Savioli, Marco Risi, and Paolo Gamba, "Experimental analysis of rssi-based indoor localization with ieee 802.15. 4," in *Wireless Conference (EW), 2010 European*. IEEE, 2010, pp. 71–77.

[30] Jiing-Yi Wang, Chia-Pang Chen, Tzu-Shiang Lin, Cheng-Long Chuang, Tzu-Yun Lai, and Joe-Air Jiang, "High-precision rssi-based indoor localization using a transmission power adjustment strategy for wireless

sensor networks," in *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE, 2012, pp. 1634–1638.

[31] Md Al Shayokh, Ugur Alkasi, and Hakan P Partal, "Performance improvement techniques for rssi based localization methods," in *Electrical Information and Communication Technology (EICT), 2013 International Conference on*. IEEE, 2014, pp. 1–6.

[32] Heiko Will, Thomas Hillebrandt, Yang Yuan, Zhao Yubin, and Marcel Kyas, "The membership degree min-max localization algorithm," in *Ubiquitous Positioning, Indoor Navigation, and Location Based Service (UPINLBS), 2012*. IEEE, 2012, pp. 1–10.

[33] Young-Bae Kong, Young-Goo Kwon, and Gwi-Tae Park, "Practical robust localization over obstructed interferences in wireless sensor networks," in *2009 Digest of Technical Papers International Conference on Consumer Electronics*. IEEE, 2009, pp. 1–2.

[34] Xin Song, Feng Yang, Lianghui Ding, and Liang Qian, "Weight adjust algorithm in indoor fingerprint localization," in *Signal Processing and Communication Systems (ICSPCS), 2012 6th International Conference on*. IEEE, 2012, pp. 1–5.

[35] Panarat Cherntanomwong and Dwi Joko Suroso, "Indoor localization system using wireless sensor networks for stationary and moving target," in *Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on*. IEEE, 2011, pp. 1–5.

[36] Teresa Garcia-Valverde, Alberto Garcia-Sola, Hani Hagras, James A Dooley, Victor Callaghan, and Juan A Botia, "A fuzzy logic-based system for indoor localization using wifi in ambient intelligent environments," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 4, pp. 702–718, 2013.

[37] Jianwei Niu, Banghui Lu, Long Cheng, Yu Gu, and Lei Shu, "Ziloc: Energy efficient wifi fingerprint-based localization with low-power radio," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 4558–4563.

[38] Essam Elkhouly, Nathan Rowe, Aly E Fathy, Michael J Kuhn, and Mohamed R Mahfouz, "Precise indoor localization systems: Alternative methods for sub-sampling uwb pulses and associated error sources," in *Biomedical Wireless Technologies, Networks, and Sensing Systems (BioWireleSS), 2013 IEEE Topical Conference on*. IEEE, 2013, pp. 1–3.

[39] Alberto Rolando and Emanuele Amoruso, "An ubiquitous positioning system based on ieee 802.15. 4 radio signals," in *Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on*. IEEE, 2013, pp. 1–10.

[40] Ahmet Bacak and Hasari Celebi, "Practical considerations for rss rf fingerprinting based indoor localization systems," in *2014 22nd Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2014, pp. 497–500.

[41] Lorenzo Palma, Luca Pernini, Alberto Belli, Simone Valenti, Lorenzo Maurizi, and Paola Pierleoni, "Ipv6 wsn solution for integration and interoperation between smart home and aal systems," in *2016 IEEE Sensors Applications Symposium (SAS)*. IEEE, 2016, pp. 1–5.

[42] Paola Pierleoni, Alberto Belli, Lorenzo Palma, Michele Palmieri, and Luca Pernini, "A wsn integrated solution system for technological support to the self-sufficient elderly," in *Ambient Assisted Living*, pp. 281–290. Springer, 2014.

[43] Paola Pierleoni, Alberto Belli, Lorenzo Palma, Marco Pellegrini, Luca Pernini, and Simone Valenti, "A high reliability wearable device for elderly fall detection," *IEEE Sensors Journal*, vol. 15, no. 8, pp. 4544–4553, 2015.

[44] Paola Pierleoni, Luca Pernini, Alberto Belli, and Lorenzo Palma, "An android-based heart monitoring system for the elderly and for patients with heart disease," *International journal of telemedicine and applications*, vol. 2014, pp. 10, 2014.

[45] Paola Pierleoni, Lorenzo Palma, Alberto Belli, and Luca Pernini, "A real-time system to aid clinical classification and quantification of tremor in parkinson's disease," in *IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*. IEEE, 2014, pp. 113–116.