POLITECNICO DI MILANO

V Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria delle Telecomunicazioni
Dipartimento di Elettronica e Informazione

INTEGRAZIONE E VALUTAZIONE DI UN PROTOCOLLO PER LA GESTIONE DELLA MOBILITÀ IN UNA RETE DI SENSORI

Relatore: Prof. Matteo CESANA
Correlatore : Ing. Alessandro REDONDI

Tesi di Laurea di:
Paolo SAVINO
Matricola 739634

Anno Accademico 2010-2011

Indice

Introduzione

Negli ultimi anni i passi in avanti nella miniaturizzazione, nella progettazione di circuiti a basso consumo e l'ottimo livello di efficienza raggiunto dai dispositivi di comunicazione a onde radio, hanno reso possibile una nuova prospettiva tecnologica: le reti di sensori. Queste reti combinano le capacità di raccogliere informazioni dall'esterno, effettuare delle elaborazioni e comunicare attraverso un ricetrasmettitore, per realizzare una nuova forma di rete, che può essere installata all'interno di un ambiente fisico, sfruttando le dimensioni ridotte dei dispositivi che la compongono e il loro basso costo.

Questi dispositivi vengono collocati all'interno dell'area dove si verifica il fenomeno che si intende analizzare o nelle sue immediate vicinanze. I nodi sensore per svolgere il loro compito sono dotati di un microprocessore, di un modulo per le comunicazioni radio e di una serie di dispositivi elettronici (i sensori) in grado di percepire le più piccole modificazioni dell'ambiente esterno, quali ad esempio temperatura, luminosità, accelerazione, etc. Tipicamente un nodo sensore svolge le seguenti operazioni: raccoglie i dati provenienti dall'osservazione locale del fenomeno, li elabora, aggrega i risultati con le informazioni provenienti da punti diversi dell'area e infine trasmette i dati verso una o più stazioni base inviandoli ai nodi vicini secondo un protocollo multi-hop.

Sulla base di tale tecnologia vari tipi di applicazioni diventano possibili: le reti di sensori possono essere impiegate nel monitoraggio ambientale, nel controllo industriale, nell'agricoltura di precisione, nel controllo automatizzato di impianti casalinghi, eccetera.

Nel contesto del monitoraggio all'interno di edifici e grazie alla possibilità di dotare determinate persone con dei nodi sensore al fine di monitorare il loro stato nasce il progetto LAURA (Localization And Ubiquitous monitoRing of pAtients for health care support), ideato dal Dipartimento di Ingegneria Elettronica ed Informazione del Politecnico di Milano in collaborazione con la Fondazione Eleonora e Lidia, residenza sanitaria per disabili di Figino Serenza (CO). L'obiettivo del progetto è quello di implementare un sistema per il monitoraggio e la localizzazione dei pazienti, all'interno di un'area dotata di una rete wireless di sensori composta da dispositivi fissi posizionati nell'edificio e alcuni dispositivi mobili posizionati sui pazienti stessi, che inviano dati ad un nodo centrale di controllo. In questo modo è possibile il monitoraggio immediato della posizione e dello stato del paziente senza alcun bisogno della presenza costante di un operatore. Per raggiungere questo scopo è necessario predisporre i nodi sensore con particolari procedure di gestione della mobilità al fine di renderli sempre

raggiungibili senza intasare la rete con un flusso di dati eccessivo e garantendo buone prestazioni.
Questo lavoro di tesi consiste nella evoluzione e integrazione del codice di una Tesi passata[1] sviluppata nel laboratorio ANTlab del Politecnico di Milano, volta a implementare dei meccanismi che permettano ai nodi sensore di muoversi senza perdere la connessione con la rete. Il codice di [1] implementa i meccanismi di mobilità HAT MOBILE ma su una vecchia versione protocollare e con la gestione non ancora ottimizzata di alcune variabili, elemento sempre critico in una WSN dove si vuole limitare il quantitativo di memoria utilizzato dai sensori e l'utilizzo della batteria.
La versione nuova di partenza, invece, implementa il protocollo HAT ma è privo delle features per ottimizzare la mobilità dei nodi in una WSN.
Si è effettuato quindi un lavoro di "merge", di incorporazione, di fusione dei due codici per sensori al fine di portare, nella nuova versione protocollare, tutte quelle caratteristiche di gestione della mobilità che HAT MOBILE possiede.
Si è testato la bontà del merge sia in INSTALLATION su sensori reali, e sia in SIMULATION, mediante TOSSIM su topologie lineari e ad albero.
Sono stati poi raccolti vari risultati sulle performance di HAT Mobile nella nuova versione protocollare del codice in termini di Percentuale di pacchetti persi dal nodo mobile a fronte di un flusso di Traffico offerto a tale sensore; questo allo scopo di valutare quanti pacchetti vengono persi a causa della mobilità e delle procedure di Handover.
La tesi è organizzata in questo modo:

Capitolo 1 - Reti di sensori: introduce le reti di sensori, ne espone le caratteristiche e ne descrive alcune applicazioni e ambiti di utilizzo, in particolare si presenta il progetto LAURA come contesto in cui è applicabile questo lavoro di Tesi;

Capitolo 2 – Strumenti hardware e software utilizzati: descrive la piattaforma hardware e software utilizzata per implementare i meccanismi di mobilità. In particolare descrive i motes MICAz, il sistema operativo TinyOS, il linguaggio NesC e il simulatore Tossim.

Capitolo 3 - Mobilità nella WSN: presenta i protocolli di gestione di Reti Wireless di Sensori presenti in letteratura HAT e HAT Mobile;

Capitolo 4 - Dettagli Implementativi: Raffronto tra vecchia versione protocollare e nuova. Fusione delle due versioni. Analisi delle modifiche apportate alla nuova versione protocollare in modo da avere le features di HAT Mobile;

Capitolo 5 – Verifica Funzionamento e Risultati : mostra la bontà del lavoro svolto testandone il funzionamento su uno scenario reale e su vari scenari simulati; valuta le prestazioni del sistema;

Capitolo 6 - Conclusioni e sviluppi futuri: considerazioni sul lavoro di tesi svolto e proposta di possibili sviluppi futuri in grado di migliorarne ulteriormente l'applicabilità di impiego.

Capitolo 1

Reti di sensori

Le reti di sensori rappresentano l'ultima frontiera della ricerca nel campo dei sistemi cosiddetti embedded [14]. Questi dispositivi sono stati concepiti per interagire con l'ambiente che li circonda, ma mentre fino ad ora la possibilità di cooperare con altri sistemi era limitata in molti casi dalle connessioni via cavo, al giorno d'oggi i progressi nelle comunicazioni senza fili hanno eliminato questo vincolo facilitandone la collocazione all'interno dell'ambiente da monitorare. Si vengono a formare in questo modo delle vere e proprie reti nelle quali ciascun nodo assolve uno specifico compito e collabora con gli altri nodi della rete per raggiungere determinati obbiettivi.

1.1 Reti di sensori e reti ad hoc

Una rete di nodi sensori condivide molti elementi con le reti ad hoc esistenti.
Ma esistono numerose differenze e specificità che hanno suscitato una particolare
attenzione da parte del mondo della ricerca. Alcuni dei punti che rendono le reti di sensori diverse dalle altre sono:

- Applicazioni

L'enorme varietà delle applicazioni possibili richiede soluzioni diversificate. Ad esempio, rispetto ai protocolli da utilizzare, la tolleranza ai guasti deve essere tenuta in considerazione se i sensori operano in un ambiente critico. Ma esistono delle applicazioni in cui i guasti si verificano raramente e quindi quest'aspetto non rappresenta un vincolo.

- Scalabilità

Potenzialmente, il numero di unità che formano una rete varia da qualche decina alle migliaia di nodi e richiedono soluzioni fortemente scalabili.

- Energia

Spesso i nodi sono alimentati unicamente da batterie ed è quindi necessario adottare misure che prolunghino al massimo la vita dei sensori.

- Autoconfigurazione

Il comportamento di un nodo, in alcuni casi, deve variare a seconda dello stato in cui si trova, come energia, posizione, ruolo, etc. In pratica i nodi devono auto configurarsi e rendere noti i parametri scelti ai vicini.

- Centralità dei dati

Rispetto alle comuni reti l'importanza di un nodo è ridotta notevolmente, quello che veramente interessa è l'osservazione del fenomeno. Se un sensore si guasta è molto probabile che ciò non influisca sulle prestazioni dell'intera rete, in quanto la ridondanza è una caratteristica delle reti di sensori.

- Semplicità

Per poter realizzare nodi sensore a basso costo, basso consumo e dimensioni ridotte, sia l'hardware che il software devono essere il più semplici possibili.

- Fattore ambientale

L'ambiente in cui una rete di sensori si troverà a lavorare impone delle scelte sostanziali, in relazione al tipo di fenomeno da osservare. Ad esempio il tipo di traffico prodotto, come banda minima, frequenza dei picchi di traffico, vincoli real-time, etc., influisce sulla scelta tra un protocollo ed un altro.

1.2 Una rete di sensori

Una rete di sensori è un insieme, eventualmente eterogeneo, di nodi sensore che forma una predeterminata topologia. I nodi sensore possono essere equipaggiati con dispositivi in grado di rilevare una grande varietà di condizioni ambientali, come temperatura, umidità, movimento dei veicoli, luminosità, pressione, livello di rumore, presenza o assenza di certi tipi di oggetti, grado di stress meccanico, valori istantanei di velocità, direzione e dimensioni di un oggetto.

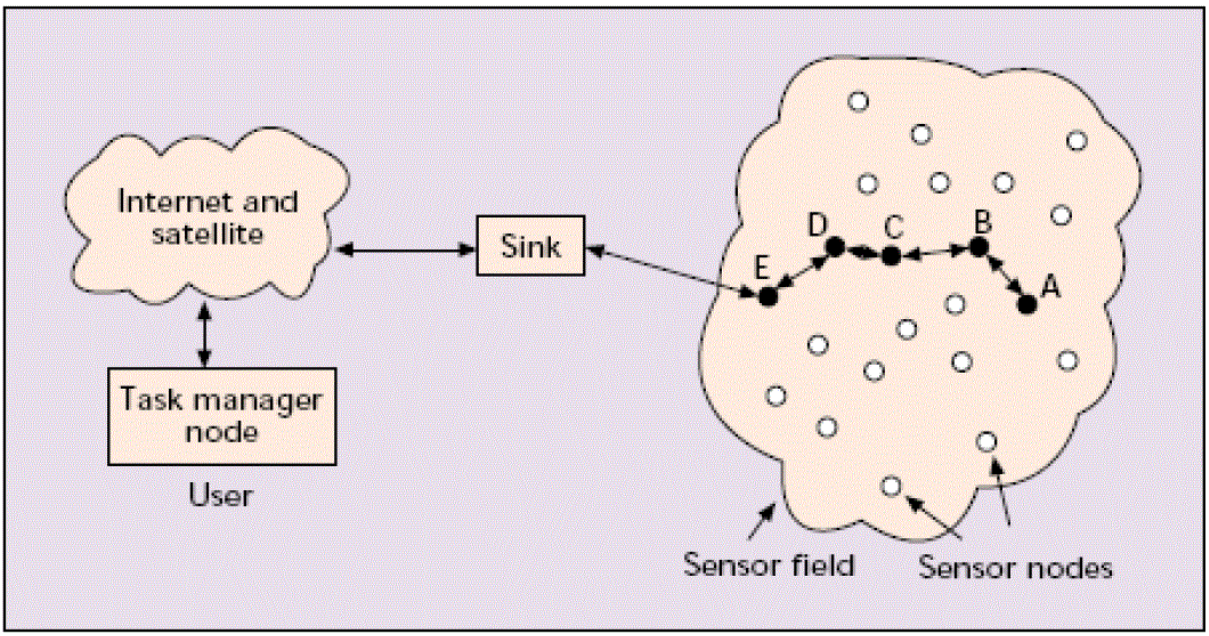

Figura 1.1: Struttura rete di Sensori

I nodi sono collocati all'interno dell'area in cui si verifica il fenomeno che si vuole analizzare, oppure, dove non è possibile, nelle sue immediate vicinanze. Questa attività può essere ripetuta quando c'è, per esempio, la necessità di rimpiazzare delle unità che hanno esaurito l'energia della loro batteria o sono state danneggiate. Generalmente ogni nodo viene associato ad un oggetto, a un essere vivente o a un luogo, vengono posti in pratica nei punti chiave del fenomeno. Il numero di sensori che compongono la rete varia da qualche decina a svariate migliaia. Le informazioni raccolte vengono poi inviate verso una stazione base, passando da un nodo ad un altro secondo un protocollo multi-hop, ed infine trasferite, via internet o via satellite, verso un centro di raccolta. Si possono prevedere configurazioni con più stazioni base, eventualmente mobili, e in questo caso, i nodi convogliano i dati verso un sottoinsieme delle stazioni stesse.
In altri casi le reti sono completamente isolate. Ciò può accadere quando i nodi sono equipaggiati con attuatori e controllori, e programmati per svolgere determinate attività.

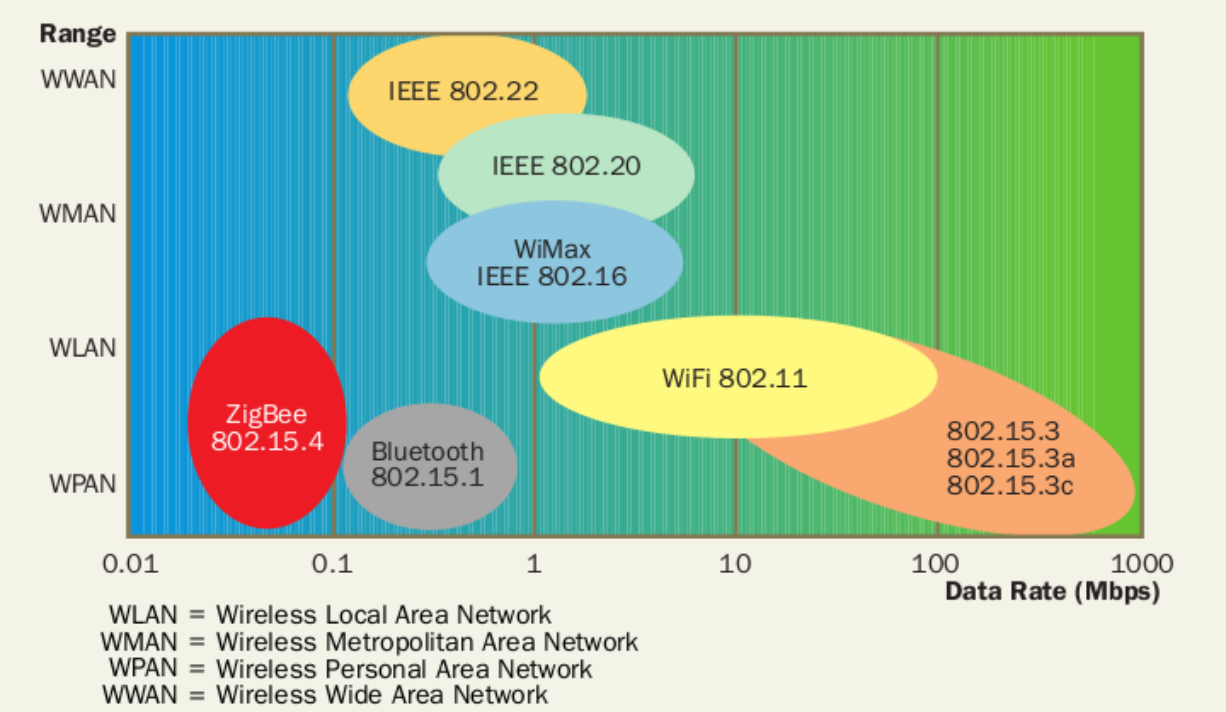

Figura 1.2: Protocolli TLC

1.2.1 Caratteristiche

Per una rete di sensori esistono una serie di fattori definiti critici, perché costituiscono le linee guida da seguire nella progettazione di algoritmi e protocolli e possono essere utilizzati come termine di paragone per differenti soluzioni. Tra i più importanti citiamo:

- Tolleranza ai guasti:

Alcuni nodi possono guastarsi o rimanere bloccati a causa di danni fisici, interferenze ambientali oppure mancanza di energia. Tuttavia il malfunzionamento di una unità non dovrebbe avere effetti collaterali sulle funzionalità dell'intera rete. Comunque, la probabilità che un nodo si guasti dipende in grossa parte dal tipo di applicazione per cui vengono impiegati.

- Scalabilità:

Il numero di sensori che possono comporre una rete variano da qualche decina alle centinaia di migliaia. I protocolli e gli algoritmi devono essere in grado di supportare un numero così elevato di nodi.

- Costi di produzione

Data l'enorme quantità di sensori che alcune applicazioni richiedono, il costo per unità non dovrebbe pesare sul costo complessivo di una rete di sensori. La speranza è quella di non superare il dollaro per un nodo, ma attualmente il prezzo del sistema radio più economico, il Bluetooth, si aggira sui dieci dollari. In più bisogna considerare che un nodo può essere equipaggiato con altri dispositivi, quali GPS, alimentatori, attuatori, che ne fanno lievitare il costo .

- Dimensioni

Un nodo sensore è generalmente costituito da un processore, un ricetrasmettitore, una batteria, un convertitore analogico/digitale, un modulo di memoria, e dei dispositivi per l'acquisizione di dati ambientali. Tutti questi moduli devono essere inseriti in una scatola che in alcuni casi non dovrebbe superare le dimensioni di un centimetro cubo se, per esempio, deve rimanere sospesa in aria.

- Consumo energetico

Un nodo sensore essendo un dispositivo microelettronico deve essere alimentato con una limitata sorgente di energia (< 0.5 A, 1.2 V). In alcuni casi i nodi possono essere equipaggiati con delle celle solari, ma quando ciò non è possibile bisogna adottare delle misure per ridurre al massimo i consumi.

Principalmente si tratta di tenere spenta la radio il più a lungo possibile, perchè tra tutti, è il dispositivo che consuma la maggior quantità di energia.

1.3 Nodi sensore

I componenti hardware che formano un singolo nodo sensore sono: il modulo di comunicazione, dei sensori, un modulo di memoria non volatile e una batteria. Spesso comunque, le applicazioni richiedono la possibilità di aggiungere delle espansioni, come attuatori, controllori, etc. Risulta quindi utile fornire i nodi di un connettore che possa interfacciarsi con il più alto numero di dispositivi.

Figura 1.3: Alcuni nodi sensore in commercio

In questo paragrafo descriveremo le caratteristiche dei componenti più interessanti, in relazione alle esigenze delle reti di sensori, come:

- Microcontrollore

I microcontrollori che possono essere montati su un nodo sensore devono essere molto semplici e progettati per consumare poca energia, ma la caratteristica fondamentale che devono possedere è la possibilità di funzionare in diverse modalità (dalla active alla sleep). Questo permette di regolare il consumo di energia in base delle funzionalità che la rete richiede in un dato momento. Un altro aspetto da tenere in considerazione sono le dimensioni del chip e la quantità di memoria disponibile all'interno di questo. I microcontrollori più utilizzati per le reti di sensori sono l'MPS 430 della Texas Instruments o quelli della famiglia Atmel.

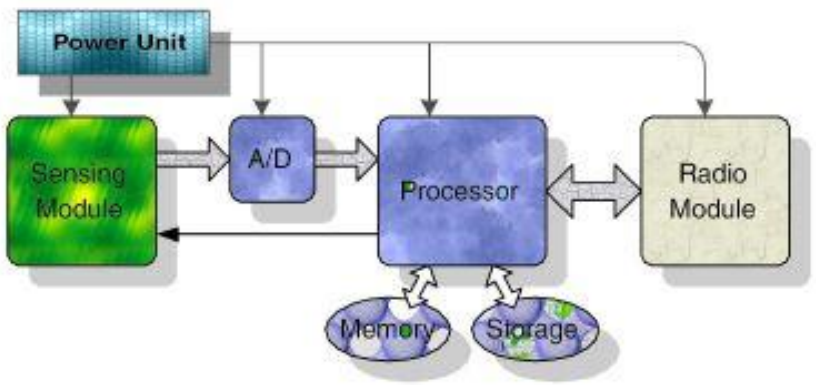

Figura 1.4 : Componenti di un sensore

- Sistema di comunicazione

Tipicamente viene utilizzato un sistema di comunicazione ad onde radio ma esistono particolari applicazioni per cui sono possibili altre soluzioni. Ad esempio, se la rete dovesse lavorare sott'acqua si potrebbe utilizzare un sistema di comunicazione ad ultrasuoni. Altri sistemi utilizzano dispositivi ottici per la trasmissione delle informazioni.

Nel caso di onde radio, i maggiori sforzi sono concentrati nello studio di sistemi in grado di risvegliare un nodo non appena avvertono l'inizio di una trasmissione. Questo favorisce il risparmio di energia, in quanto il maggior numero di nodi può attendere in modalità sleep, il verificarsi di un evento (che viene segnalato dai pochi nodi attivi con l'invio di un pacchetto). Il risveglio può riguardare tutti i nodi vicini in ascolto oppure solo quei nodi direttamente indirizzati. Attualmente i ricetrasmettitori radio più utilizzati sono i dispositivi RFM TR1001, Infineon o Chipcon. Generalmente viene utilizzata la modulazione ASK o FSK, solo il Berkeley PicoNodes usa la modulazione OOK.

-Alimentazione

Normalmente i nodi sono alimentati da batterie esauribili ed è necessario prevedere, dove possibile, sistemi per il ricarico di energia, come celle solari, etc.

1.4 Lo Standard IEEE 802.15.4

Lo standard IEEE 802.15.4 [3] definisce le specifiche dei livelli 1 e 2 per reti wireless a basso bit-rate e a basso consumo energetico. Già nel 1998, quando iniziarono i primi studi in quest'ambito, nacquero la ZigBee Alliance, un consorzio di aziende industriali e il comitato IEEE 802.15 che si occuparono da subito dello sviluppo di questa tecnologia. Attualmente la ZigBee Alliance, che conta più di 200 aziende (tra cui Freescale, Huawey, Mitsubishi, Motorola, Philips, Samsung, Siemens, ST Microelectronics, Texas Instruments, Telefonica,

KDDI, Telecom) si occupa dello sviluppo dei livelli alti, quali il livello rete e applicativo, mentre il comitato IEEE 802.15, e soprattutto il sottogruppo (.4), si occupa della standardizzazione del livello fisico e data-link.
Lo standard IEEE 802.15.4 prevede due tipologie di dispositivi, Full-Function (FFD) e Reduced-Function (RFD). Un dispositivo FFD può svolgere la funzione di Pan Coordinator e può comunicare sia con altri FFD che con RFD. Al contrario un RFD è un dispositivo con funzionalità più semplici, con capacità computazionali limitate e può comunicare esclusivamente con un FFD. Per questo motivo gli RFD sono principalmente utilizzati per compiti semplici. In base allo scenario applicativo, lo standard fornisce due diverse topologie di configurazioni: a stella o peer-to-peer (Figura 1.5). Nella topologia a stella la comunicazione è regolata da un dispositivo FFD che svolge la funzione di PAN Coordinator, che è responsabile dell'invio di tram tra due dispositivi RFD. La particolarità di questa configurazione è che la comunicazione avviene tra un singolo dispositivo e il PAN Coordinator. La topologia peer-to-peer prevede invece che i dispositivi FFD possano comunicare con tutti i dispositivi all'interno dell'area di copertura.

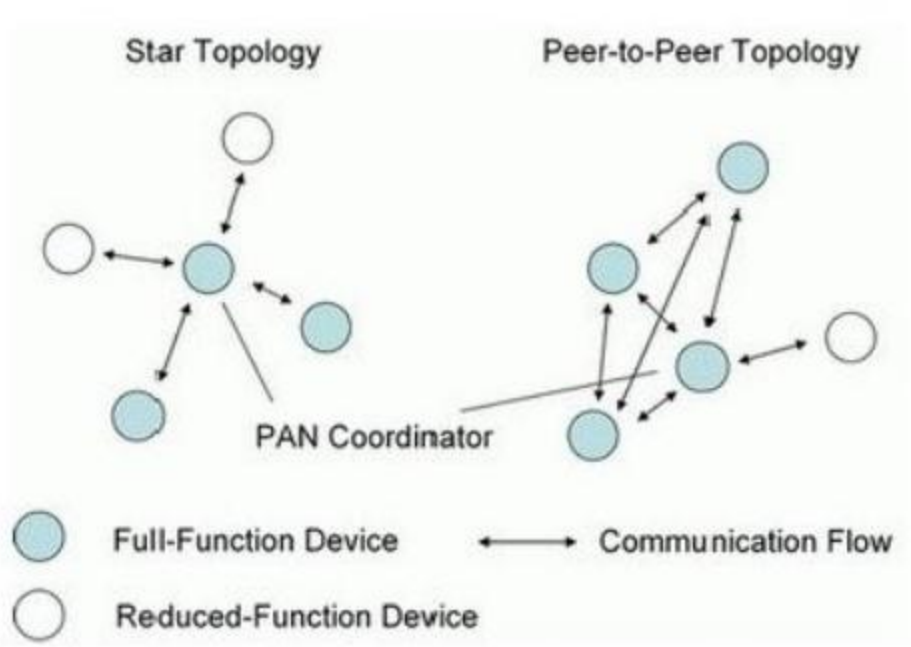

Figura 1.5 : Topologie di configurazione a Stella e Peer-to-Peer

Il livello fisico prevede quattro diversi schemi di modulazione in tre bande di funzionamento principali: Trasmissione a frequenza 868/915 MHz con modulazione BPSK (Binary Phase Shift Keying); Trasmissione a frequenza 2450 MHz (ISM) con modulazione O-QPSK (Offset-Quadrature Phase Shift Keying); trasmissione a frequenza 868/915 MHz con modulazione BPSK o ASK (Amplitude Shift Keying).

Il livello MAC prevede invece due modalità di funzionamento: una modalità beaconless e una modalità beacon-enabled. Il beacon ha funzioni di sincronizzazione dei periodi di attività dei dispositivi, ed è necessario per le procedure di network discovery. Nel caso in cui il PAN Coordinator non trasmetta la trama di beacon l'accesso al canale è senza alcuna sincronizzazione fra i dispositivi e si basa sull'algoritmo CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance). Nel momento in cui un dispositivo vuole tentare una trasmissione ascolta il canale e se è occupato il dispositivo attiva un timer di durata casuale (detto tempo di backoff) che viene decrementato solo durante i periodi di inattività del canale. Se il canale risulta libero lo prenota ed dopo un certo lasso di tempo trasmette. Opzionalmente la trama trasmessa può essere riscontrata con una trama di ACK. Nel caso in cui il PAN Coordinator trasmetta la trama di beacon (rete beacon-enabled) l'accesso al canale è di tipo CSMA-CA suddiviso a slot, cioè in blocchi o finestre. Il tempo sul canale è organizzato in supertrame, separate dalla trasmissione di due beacon frame, le quali possono essere divise in un Inactive Period in cui è possibile spegnere il dispositivo e un Active Period diviso in 16 slot. Perciò i dispositivi devono sincronizzarsi con l'inizio di uno slot e completare le eventuali trasmissioni entro il termine dello stesso slot. Opzionalmente il PAN Coordinator può riservare una parte di slot alla fine dell'Active-Period per applicazioni con particolari requisiti di banda.

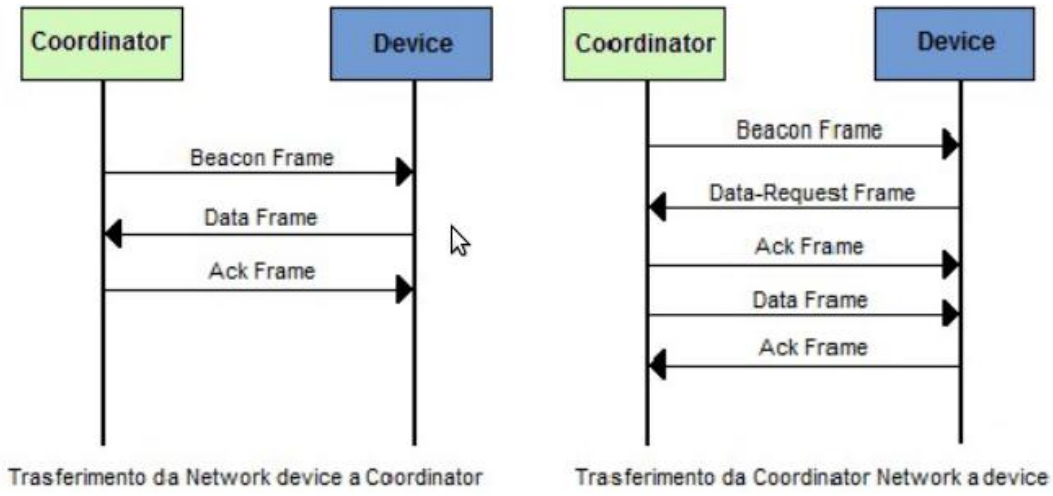

Figura 1.6 : Comunicazione ZigBee

Come visto, questa specifica imposta la sincronizzazione a livello di singolo hop. Nel caso di reti multihop con topologie ad albero o mesh non è definita alcuna metodologia standard per la trasmissione del beacon-frame. Per questo motivo la ricerca è molto attiva nell'ambito della sincronizzazione multihop. Come già detto la ZigBee Alliance è attiva nello studio della parte di alto livello relativa alle reti di sensori. Dal 2005 questo consorzio ha immesso sul mercato ZigBee [4] che si basa sullo standard 802.15.4 per la trasmissione radio e la gestione

dell'accesso al mezzo. Tuttavia lo stack protocollare è stato completato con l'aggiunta di software per la creazione di reti multi-tratta e l'instradamento.

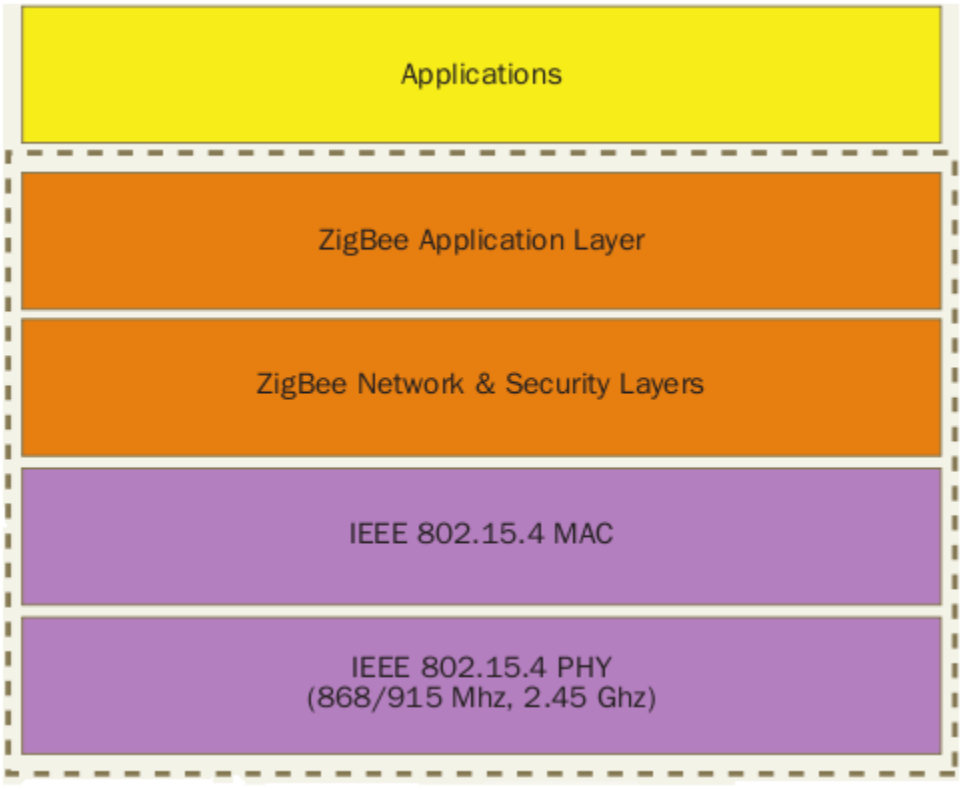

Figura 1.7 : Pila Protocollare ZigBee

Il tallone d'Achille consta nel fatto che si tratta di una soluzione a scatola chiusa che non consente la gestione a basso livello della piattaforma e l'ottimizzazione dei protocolli alle esigenze specifiche del sistema che si intende realizzare.

I campi di applicazione per la tecnologia ZigBee sono molto estesi sia in ambito indoor che outdoor ed interessano tematiche quali la domotica, la gestione dell'energia in ambito domestico ed industriale, l'elettronica di consumo e le reti di sensori per il monitoraggio dell'ambiente. Per garantire l'interoperabilità tra prodotti di diversi fornitori, la ZigBee Alliance cura la creazione di profili applicativi standard. Sono attualmente attivi i seguenti gruppi che curano la realizzazione dei profili applicativi Smart Energy, Home Automation,Telecom Application,Wireless Sensor Network, Personal Home and Health Care e Commercial Building Application. L'operatore, in questo scenario di rete pervasiva fisso/mobile estesa ad un ambiente sempre più intelligente, può giocare così un importante ruolo strategico, creando un ecosistema unico di cui assumere il controllo, non solo come trasportatore di dati, ma identificandosi come gestore della personalizzazione ed integrazione con le esigenze dell'utente finale.

1.5 Applicazioni

Grazie alla capacità di campionare grandezze come temperatura, umidità, pressione, luce, ma anche di rilevare il movimento di veicoli, la composizione del terreno, il livello di rumore e molti altri eventi, le Reti di Sensori, finora utilizzate soprattutto in ambito di ricerca accademica, stanno cominciando a diffondersi anche in ambito applicativo spinte soprattutto dal forte interesse dell'industria per l'uso di dispositivi wireless nei più svariati settori.
Possono essere impiegate con successo in tutti quelle applicazioni in cui la raccolta di dati continua è fondamentale, inoltre sono sicuramente utilizzabili in tutte le situazioni di monitoraggio di aree molto estese di difficile accesso.

Le applicazioni delle Reti di Sensori sono molteplici, è possibile classificarle in cinque grandi categorie: militari, ambientali, domestiche, commerciali e mediche.
A riguardo di quest'ultimo ambito si preannuncia il progetto LAURA che verrà trattato più ampiamente subito dopo.

Militari
La rapida distribuzione, l'auto-organizzazione e la tolleranza ai guasti fanno delle WSN un'ottima tecnologia sfruttabile in ambito bellico. Poiché le Reti di Sensori sono caratterizzate da un'alta densità spaziale dei nodi e dal basso costo dei componenti, la distruzione di alcuni motes da parte del nemico non danneggia l'operatività della rete (come potrebbe accadere invece con la distruzione dei dispositivi tradizionali).
Le applicazioni in ambito bellico sono molteplici e spaziano dal monitoraggio di forze alleate o nemiche, al controllo degli armamenti stipati nei magazzini passando per la sorveglianza del campo di battaglia.
È anche possibile utilizzare le WSN per effettuare il riconoscimento dei nemici, la stima dei danni provocati oppure identificare agenti chimici, biologici o radioattivi in un'area soggetta ad attacchi.
In ambito bellico è solito gettare da un aeroplano in volo i sensori [Figura 1.8].
Grazie all'auto-organizzazione essi riescono a creare correttamente una topologia di rete adatta agli scopi e sono in grado di monitorare il passaggio dei veicoli nell'area circostante. Questo progetto [5] è stato sviluppato dall'Università di Berkeley in collaborazione con United States Marine Corps Air/Ground Combat Center (MCA-GCC) ed ha avuto esiti positivi.

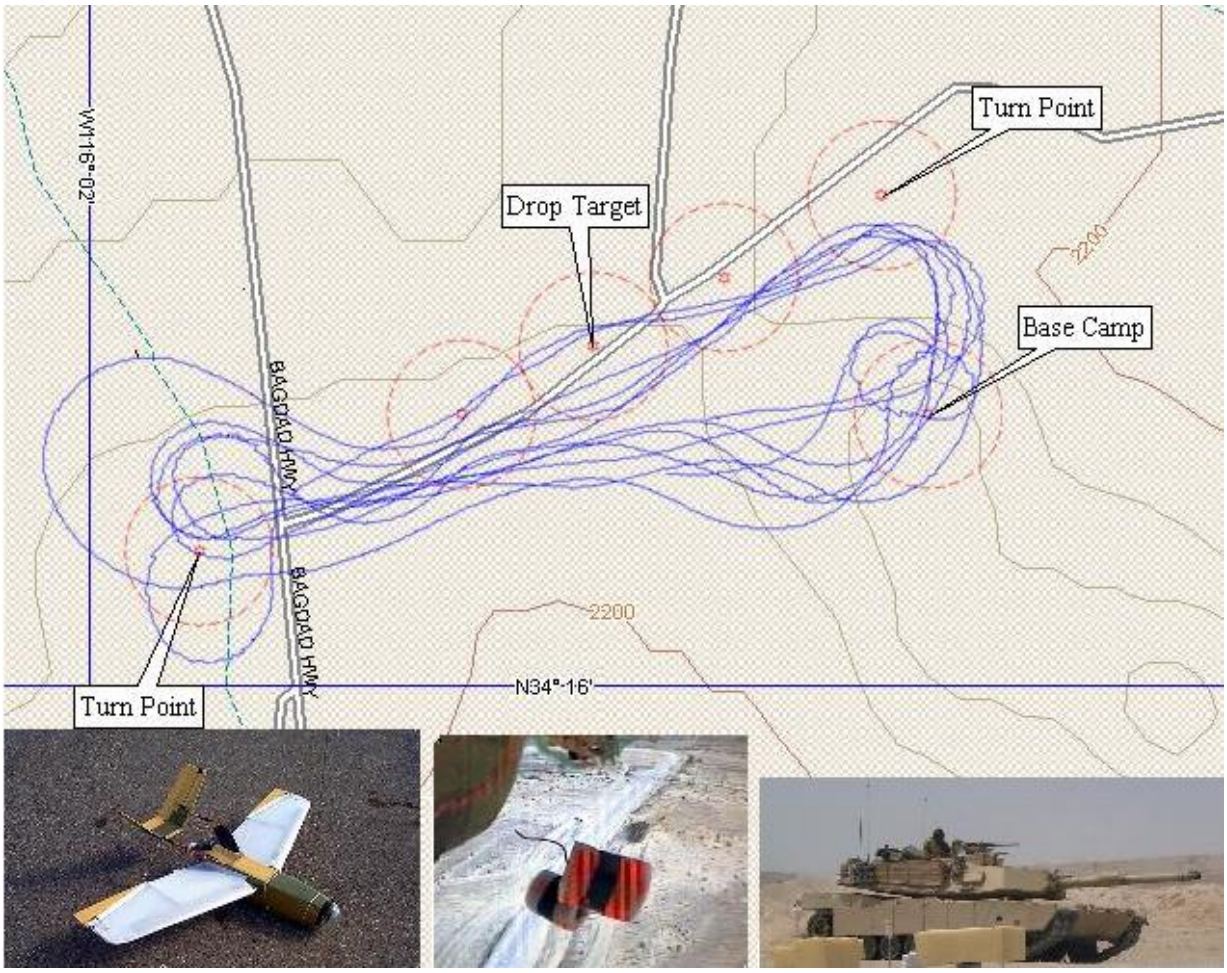

Figura 1.8 : Monitoraggio Mezzi Militari

Ambientali

La misura di parametri fisici di un ambiente è fondamentale per la conoscenza e lo studio di molti processi naturali. L'impiego delle WSN per il monitoraggio ambientale [13] rappresenta uno dei campi di applicazione con le più grandi potenzialità poiché in molte situazioni senza la disponibilità di una rete, con le citate caratteristiche, per la raccolta dei dati risulta estremamente difficile o addirittura impossibile il monitoraggio continuativo e su vasta scala. Alcune semplici applicazioni ambientali delle Reti di Sensori riguardano il monitoraggio di habitat naturali. Come ad esempio sull'isola Great Duck in Canada, dove alcuni ricercatori hanno posto un centinaio di nodi con lo scopo di monitorare l'habitat del Petrel, prima e dopo i suoi spostamenti. Il Petrel è un uccello che passa la maggior parte della sua vita in mare e che per piccoli periodi di tempo torna sulla terraferma per riprodursi e nidificare.

Le Reti di Sensori vengono anche utilizzate nell'agricoltura di precisione con lo scopo di monitorare in tempo reale il livello dei pesticidi nell'acqua, il grado di erosione del terreno o la quantità di sostanze inquinanti nell'aria.

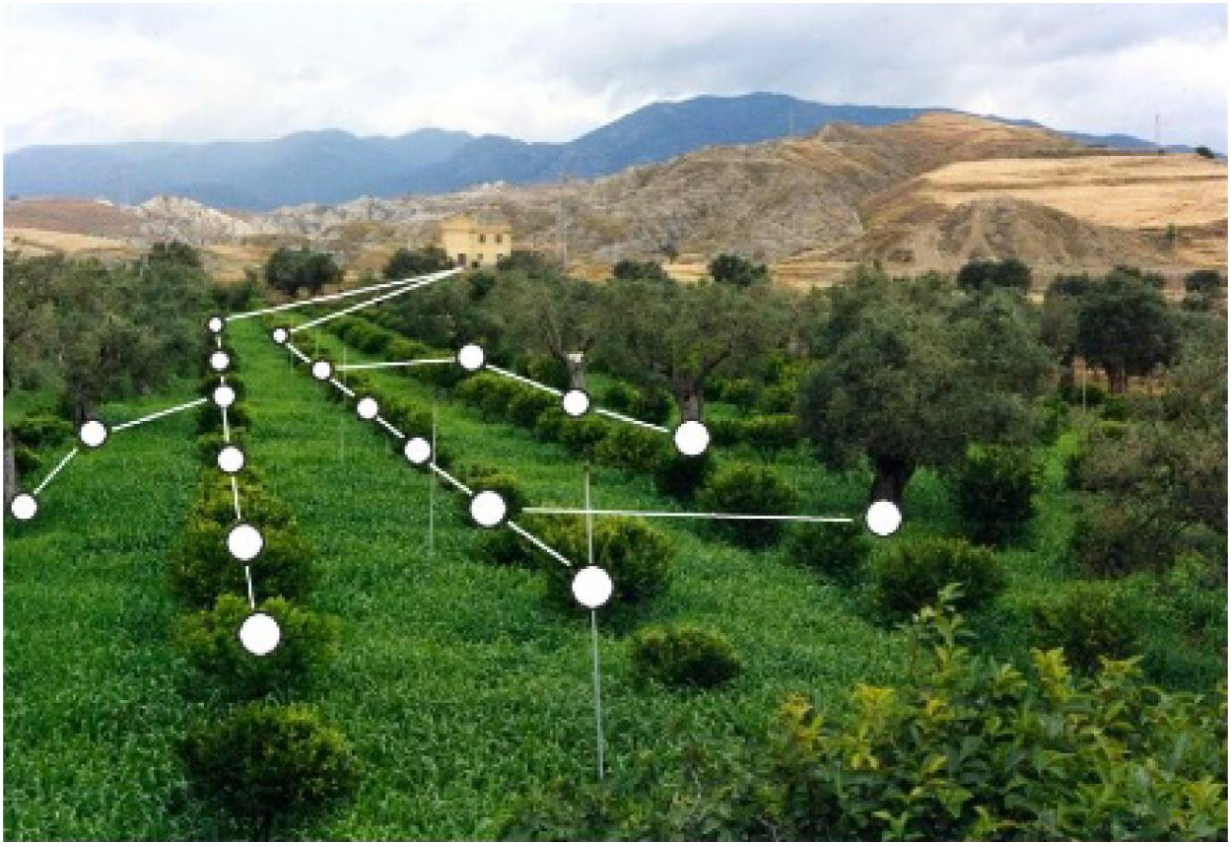

Figura 1.9 : Utilizzo di Sensori nell'architettura di precisione

È anche possibile controllare una zona boschiva per rilevare immediatamente eventuali incendi. Negli Stati Uniti è stato messo a punto un sistema capace di predire e rilevare le inondazioni, ALERT[6], il quale è basato su una rete formata da moltissimi nodi in grado di misurare le precipitazioni, la portata dei fiumi, il livello dell'acqua e i parametri relativi alle condizioni climatiche (pressione, umidità, temperatura, ecc.).

Domestiche

Un esempio di utilizzo di una Rete di Sensori in ambito domestico è l'automazione della casa che consiste nell'inserire motes nel forno, aspirapolvere, refrigeratore, videoregistratore, etc. (Figura 1.10). Questi nodi, inseriti negli elettrodomestici, possono interagire l'uno con l'altro ed anche con reti esterne tramite l'utilizzo di internet o del satellite, permettendone la gestione anche da distanze remote.

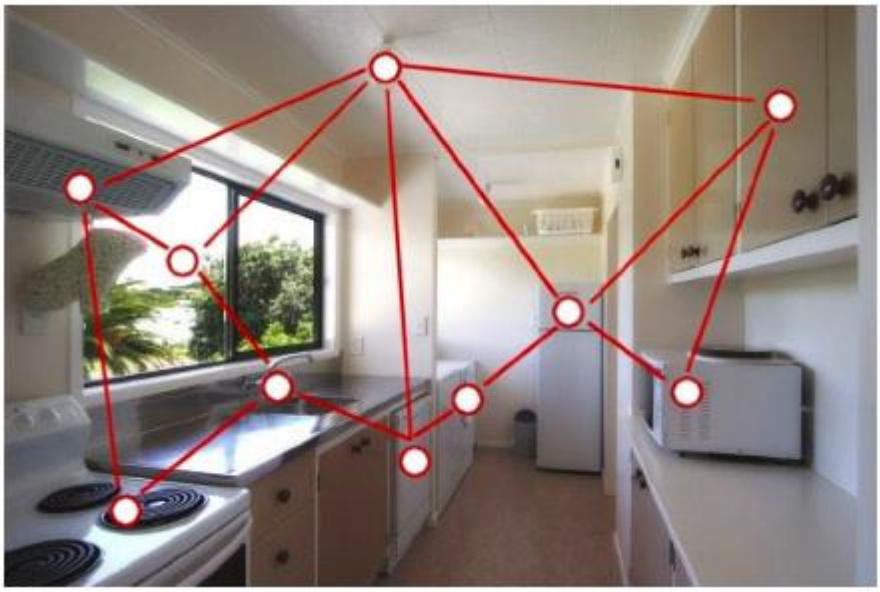

Figura 1.10 : Sensoristica in Ambiente Domestico

La domotica è la disciplina che si occupa di questi studi e impiega le WSN per permettere la coordinazione tra le componenti elettroniche della casa. L'ambiente domestico viene ad assumere così le stesse caratteristiche di un piccolo centro di elaborazione fornito di una rete in grado di mettere in comunicazione tra loro tutti i vari strumenti di cui l'ambiente è composto.

Mediche

Uno dei principali utilizzi in ambito medico-sanitario delle reti di sensori è il monitoraggio delle condizioni cliniche a distanza, applicazione utilizzata anche dalle agenzie spaziali per monitorare le condizioni fisiche degli astronauti. Tramite le WSN è possibile controllare costantemente i parametri vitali del paziente (pressione arteriosa, temperatura, frequenza cardiaca) in modo non invasivo, consentendo un rapido intervento del medico in caso di improvvise alterazioni.

Le WSN possono anche essere utilizzate per automatizzare la terapia farmacologica prevista per ogni paziente, rendendo più efficiente il lavoro degli infermieri e del personale medico di assistenza. Un'altra applicazione utile legate all'ambito medico è quello dell'utilizzo di nodi sensore per il controllo dell'attività fisica, sia per accelerare la ripresa dopo un intervento chirurgico o dopo un infortunio, sia per avere un supervisore che controlli il corretto svolgimento dell'attività fisica.

Il progetto CodeBlue [7] ha consentito invece l'utilizzo di reti di sensori per il monitoraggio dei parametri vitali dei pazienti all'interno del Boston Medical Center e del Spaulding Rehabilitation Hospital. I nodi wireless chiamati Pluto vengono applicati ad ogni paziente e consentono la misurazione real-time della frequenza del battito cardiaco, saturazione

dell'ossigeno ed elettrocardiogramma. I nodi wireless chiamati Pluto (Figura 1.11 a) vengono applicati ad ogni paziente e consentono la misurazione real-time della frequenza del battito cardiaco, saturazione dell'ossigeno ed elettrocardiogramma Per ottenere questi scopi, i mote sono equipaggiati con un sensore elettromiografo (EMG), un accelerometro, un giroscopio e un sensore per l'elettrocardiogramma (ECG) (Figura 1.12). I dati ricavati vengono costantemente controllati per essere sicuri che i parametri vitali della persona siano nella norma. Se viene riscontrata un'anomalia in un parametro misurato, un dottore dotato di palmare viene subito avvertito tramite la rete wireless (Figura 1.11 b)in modo da poter intervenire tempestivamente e prestare al paziente le cure necessarie.

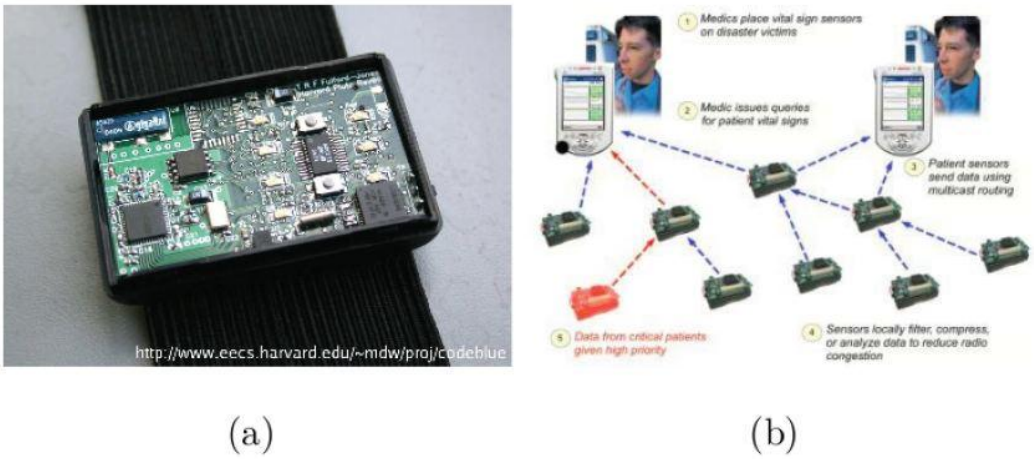

Figura 1.11 : (a) nodo Pluto , (b) architettura per le emergenze

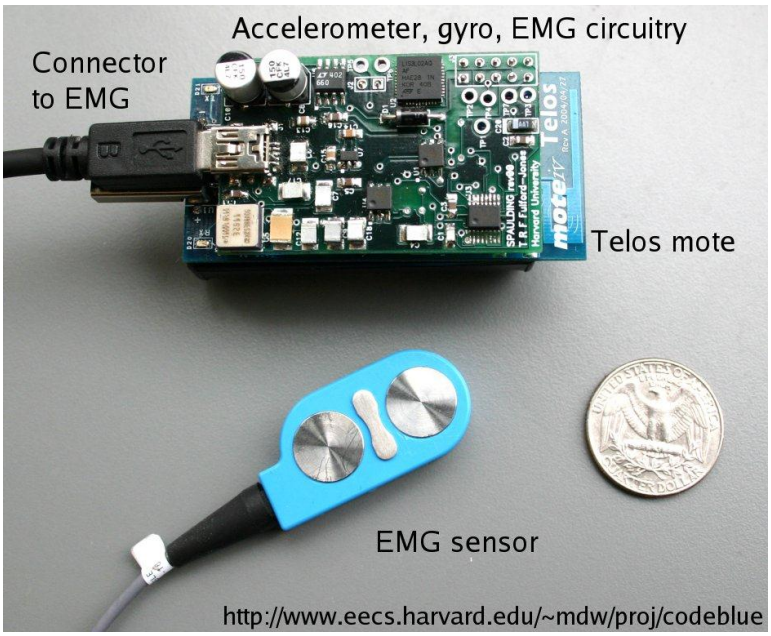

Figura 1.12 : accellerometro, giroscopio, EMG

Il monitoraggio continuativo del moto di un individuo, prima dell'introduzione delle WSN, veniva effettuato tramite un' architettura cablata di sensori. Un' architettura di questo tipo comporta però enormi svantaggi: la limitata mobilità che il paziente deve tollerare a causa dei cavi utilizzati per il cablaggio, la bassa scalabilità offerta, l'elaborazione dei dati che non può essere effettuata in tempo reale.

1.6 Sistema LAURA

Nell'ambito medico di applicazione di una Rete di Sensori, merita un paragrafo a parte la descrizione del Sistema LAURA[8], essendo l'ambito applicativo del lavoro di questa tesi.

1.6.1 Caratteristiche

Negli ospedali e nelle case di cura con pazienti che soffrono di disturbi cognitivi e/o fisici, uno dei problemi più comuni e significativi è l'efficienza dell'attività di controllo e supervisione dei pazienti. Questo problema può essere suddiviso in due aspetti principali:
- Localizzazione: il personale medico deve essere in grado di sapere in ogni momento l'esatta posizione di ciascun paziente; il target di precisione di tale dato può variare in base a diversi fattori : si può scegliere una precisione a livello di stanza oppure l'esatta localizzazione all'interno di una stanza.
- Monitoraggio dello stato: il personale medico deve essere sempre a conoscenza dello stato aggiornato dei principali livelli biometrici di ogni paziente. Si monitorano diversi parametri in modo da rilevare in essi una variazione che può rappresentare una situazione di pericolo.

Alcuni tentativi di sviluppare un sistema capace di adempiere a tali problematiche si sono concentrati sulle tecnologie GPS e video. Tali scelte si sono tuttavia rivelate inadeguate, in quanto soluzioni basate sul sistema GPS incontrano grosse difficoltà quando la localizzazione deve avvenire in luoghi chiusi, mentre soluzioni che prevedono l'utilizzo di videocamere non svincolano il personale medico dalla necessità di osservare continuamente i soggetti.
Considerando che esistono dispositivi mobili in grado di monitorare valori biometrici critici o di raccogliere dati dai quali si è in grado di dedurre alcune situazioni di pericolo, è nata l'idea per una nuova soluzione: l'utilizzo di apparecchiature "indossabili" dai soggetti monitorati in una rete di sensori, per permettere l'analisi centralizzata dei segnali da essi generati. Grazie a questa centralizzazione, l'attività di localizzazione e gran parte delle attività di monitoraggio possono essere completamente delegate ad un sistema automatizzato. Una soluzione di questo tipo deve essere in grado di rispondere ai seguenti

requisiti:
- I dispositivi applicati ai pazienti devo essere quanto meno invasivi possibile.
- I dispositivi devono periodicamente comunicare con un sistema centrale per la trasmissione dei dati raccolti.
- I sensori devono avere la capacità di comunicare in modo wireless.
- I sensori devono avere un'adeguata autonomia.
- Il sistema deve funzionare nell'intera area di interesse, sia essa un ambiente chiuso, che uno spazio aperto.
- Il sistema deve essere di facile installazione, fruizione ed il più compatibile possibile con le strutture preesistenti.

Un caso pratico di implementazione di questa idea è il progetto LAURA (LocAlization and Ubiquitous monitoRing of pAtients for healthcare support) [8]. La sperimentazione di tale sistema avviene in un contesto di cooperazione con la "Fondazione Eleonora e Lidia" di Figino Serenza (CO), un piccolo istituto specializzato nell'assistenza a persone con particolari necessità. La sperimentazione verterà principalmente sulla localizzazione e sul monitoraggio di pazienti che soffrono di attacchi epilettici, al fine di garantire un tempestivo intervento.

1.6.2 Architettura del Sistema

I tre blocchi principali che costituiscono l'architettura del sistema sono schematizzati nella figura sottostante. Personal Monitoring System (PMS), Personal Localization System (PLS) e Network Architecture (NA), quest'ultimo usato per trasmettere le informazioni ad un controllore centrale (Automatic Central Controller).

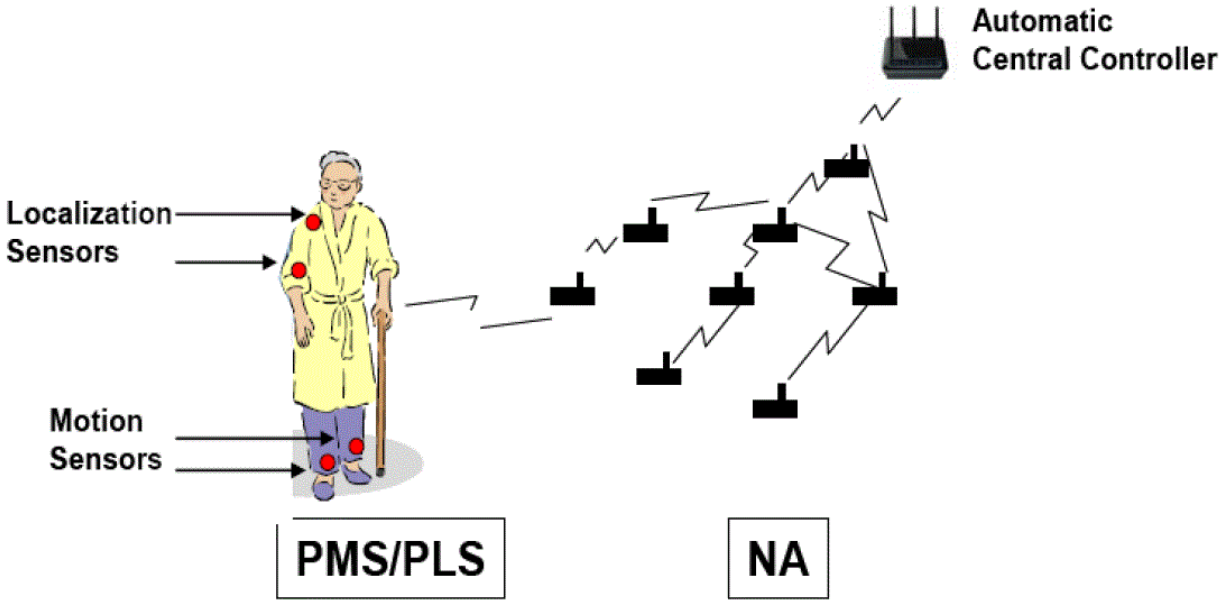

Figura 1.13 : Blocchi dell'architettura del progetto LAURA

- Personal Monitoring System (PMS): si occupa del monitoraggio centralizzato dello stato di ciascun paziente attraverso l'uso dei dispositivi wireless precedentemente citati. Tali dispositivi, indossati dai pazienti, non ne ostacoleranno le normali attività. Nel caso specifico della nostra cooperazione con la "Fondazione Eleonora e Lidia" ci limiteremo all'impiego di accelerometri al fine di rilevare eventuali cadute o crisi epilettiche dei degenti. Il PMS è composto da un'architettura di rete leggera e non intrusiva oltre che da una serie di algoritmi di elaborazione dei segnali in grado di ottimizzare l'informazione rilevata rispetto l'architettura scelta.

- Personal Localization System (PLS): si occupa, come suggerisce il nome, della localizzazione dei pazienti all'interno della rete di sensori, al fine di garantire un tempestivo intervento in caso di necessità. In base alla tecnologia utilizzata tale localizzazione può avvenire in modo più o meno accurato; avremo dati più veritieri se ci basiamo su ricevitori di ultrasuoni, mentre saremo meno precisi se ci si basa sulla misurazione dell'energia associata ai segnali emessi dai vari sensori. Così come il PMS anche il PLS è fondamentalmente composto da un'architettura di rete particolarmente "leggera" e da algoritmi di elaborazione di segnali, che vengono maggiormente centralizzati al fine di aumentare l'autonomia dei nodi, riducendone il carico di lavoro.

- Network Architecture (NA): implementa algoritmi di routing ad-hoc per una rete complessa come quella descritta, minimizzando il consumo energetico dei singoli nodi.

Il sistema LAURA è composto da:
- Anchor Nodes: fanno parte dell'infrastruttura stessa e sono staticamente disposti nell'area da monitorare;
- Client Nodes: sono posizionati sui pazienti al fine di permettere la loro localizzazione e il loro monitoraggio;

I dati raccolti sono poi inviati ad un punto di controllo attraverso una NA che si basa sul protocollo di routing gerarchico HAT, descritto successivamente, che organizza la rete in una topologia ad albero creata e gestita attraverso politiche di associazione dinamiche. Appena attivati, i nodi entrano in uno scanning state in cui raccolgono i beacon inviati dagli altri nodi. Successivamente viene individuato il nodo a cui associarsi attraverso una funzione di costo che dipende da tre fattori: Received Signal Strength Indicator (RSSI), attuale numero di nodi associati al candidato, distanza del candidato dal Pan Coordinator in numero di hop. Una volta

individuato il candidato parte la procedura di associazione che consiste in un rapido scambio di messaggi al fine di settare alcuni parametri. Successivamente e per tutta la durata dell'associazione parte l'invio periodico di beacon per il mantenimento di essa.

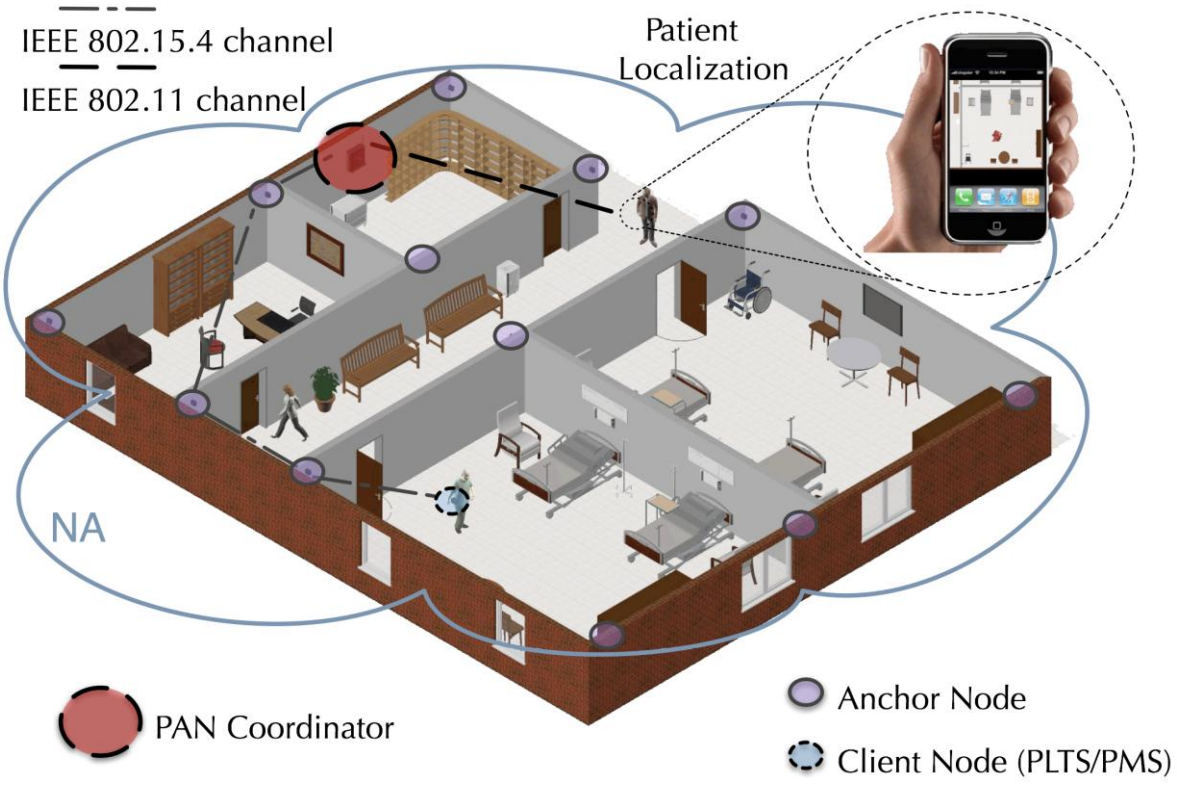

Figura 1.14 : Schema di LAURA

Capitolo 2

Strumenti Hardware e Software utilizzati

2.1 MICAz mote

La tipologia di nodi sensore che è stata utilizzata per questo lavoro è conosciuta con il nome di mote MICAZ (Figura 2.1).
I sensori MICAZ sono prodotti dall'azienda MEMSIC, che dal 2010 ha acquisito la l'azienda americana Crossbow Technology ,che dal 1995 fornisce sistemi di sensori inerziali per l'aviazione e per applicazioni in ambito militare e si occupa di commercializzare soluzioni complete per la realizzazione di WSN. I sensori MICAZ hanno dimensioni di 58 x 32 x 7 millimetri, sono alimentati da 2 batterie di tipo AA e possono essere equipaggiati con sensori di diverso tipo per rilevare fenomeni fisici quali ad esempio temperatura, pressione, accelerazione, luce, campi magnetici.

Figura 2.1 : Micaz Mote

Questi trasduttori sono costruiti sfruttando le proprietà di diversi materiali che variano le loro caratteristiche elettriche al variare delle condizioni ambientali e il valore di tensione che essi producono viene convertito in valore binario da un ADC. La ricetrasmittente è costituita da dal chip radio CC2420 [12], in grado di svolgere tutte le funzioni di livello fisico richieste dallo standard IEEE 802.15.4. Inoltre sono dotati di tre led (rosso, giallo e verde) che possono essere gestiti dal programmatore e costituiscono uno strumento fondamentale p\ er il debug (

soprattutto nel caso di software come quello da me realizzato, nel quale si programma a livello hardware, gestendo spesso eventi asincroni, che non sono riscontrabili tramite simulazione).
I sensori sono composti da un microprocessore programmabile, un Atmel ATMega128L, che per poter elaborare i dati e svolgere le funzioni richieste ha a disposizione 128 KByte di memoria programmabile riservata per l'installazione del software e 4 KByte di memoria riservata per le variabili temporanee. Per poter programmare i MICAZ è stata utilizzata la scheda di programmazione MIB520 mostrata in Figura 2.2, collegandola al PC attraverso la porta USB del computer.

Figura 2.2 : MIB520

2.1.1 Chip Radio CC2420

Il chip utilizzato per l'interfaccia radio è il CC2420, costruito appositamente per i dispositivi delle LR-PAN (Low-Rate Wireless Personal Area Networks) conforme allo standard IEEE 802.15.4 ed allo standard ZigBee.
Con questa interfaccia radio è possibile utilizzare tutti i canali nella banda a 2.4 GHz previsti dallo standard ed anche impostare la potenza di trasmissione su uno degli otto livelli elencati in Tabella 2.1.

Livello di Potenza	Potenza in Uscita [dBm]	Corrente Consumata [mA]
31	0	17.4
27	-1	16.5
23	-3	15.2
19	-5	13.9
15	-7	12.5
11	-10	11.2
7	-15	9.9
3	-25	8.9

Tabella 2.1: Livelli di potenza e corrente del Chip CC2420

Interrupt

Il chip utilizza due buffer, chiamati RXFIFO e TXFIFO , rispettivamente per memorizzare i pacchetti ricevuti e da trasmettere. La ricezione di un pacchetto viene segnalata dal CC2420 al processore attraverso un interrupt, il quale sarà poi gestito dal software che richiamerà la funzione per leggere il pacchetto in memoria. Un interrupt è un segnale asincrono generato dall'hardware (ad esempio il cambio del valore di un pin da 0 a 1) per richiedere al sistema operativo l'esecuzione di alcune operazioni.

Attivazione

La radio può lavorare in quattro modalità differenti: *trasmettitore*, *ricevitore*, *idle*, *sleep*. Le prime due costituiscono lo stato di attività (ricezione o trasmissione), nella terza modalità invece alcune funzionalità hardware sono disattivate ma si può comunque passare facilmente in modalità attiva mentre nella modalità sleep il chip è praticamente spento.

La procedura di attivazione del CC2420 si articola in tre fasi:

1. prima di tutto è necessario fornire un'alimentazione stabile di 1.8 V al CC2420 tramite l'avvio del Voltage Regulator, il quale ha bisogno di 0.6 ms per attivarsi;

2. la seconda fase consiste nell'attivare l'Oscillatore locale, la quale necessita in media 1 ms;

3. infine bisogna impostare alcuni parametri nei registri di configurazione del chip (per esempio: frequenza; potenza di trasmissione; ID del nodo; ecc.).

Quest'ultima fase richiede un tempo inferiore alle prime due e quindi può essere trascurato. Quindi il chip radio per passare dallo stato idle allo stato di attività richiede un tempo minimo

pari a 1.6 ms. Il tempo previsto deve essere leggermente sovradimensionato per evitare di non riuscire ad attivare il chip radio in tempo per una comunicazione.
Il CC2420, oltre a tutte le funzioni del livello fisico previste dallo standard IEEE 802.15.4, implementa in hardware anche altre funzioni per velocizzare le operazioni di elaborazione delle informazioni tra cui l'invio automatico degli ACK e l'Address Recognition (AR) che analizza l'header dei pacchetti ricevuti e li invia al livello MAC solo se sono destinati al dispositivo. Per utilizzare questo controllo, che permette di risparmiare energia e tempo, è però necessario che il CC2420 conosca l'indirizzo della propria interfaccia radio e l'identificativo della PAN.

2.2 TinyOS

TinyOS è un sistema operativo open-source, sviluppato dalla University of California at Berkeley. Data la possibilità di modificare il codice, questo sistema operativo è diventato la piattaforma di sviluppo per ogni soluzione proposta nel campo delle reti di sensori. In effetti grossi contributi sono stati forniti dalla comunità di sviluppatori, lo testimonia la lunga lista di progetti attivi relativi a tutti campi della ricerca, da protocolli per l'instradamento dei pacchetti alla localizzazione, dalla realizzazione di un interfaccia grafica per lo sviluppo delle applicazioni all'estensione del compilatore ncc per il supporto di nuove piattaforme hardware (come la piattaforma 8051).
La prima versione del sistema operativo è stata rilasciata nell'Ottobre del 2002 ed stata seguita da continui aggiornameti fino alla versione 1.15 rilasciata nel Giugno 2005. Questa prima versione ha dimostrato per alcuni limiti dovuti alla struttura non intuitiva e alla dipendenza troppo stretta di molti componenti che non permette il riutilizzo del codice.
Quindi, la necessità di avere a disposizione un sistema operativo più modulare ha portato allo sviluppo della versione 2.x, per la cui implementazione si è deciso di non utilizzare la versione 1.x come base ma di ridisegnarlo completamente per creare un sistema completamente nuovo, adattabile a diverse piattaforme hardware, di più semplice comprensione e rapidamente modificabile.
TinyOS è stato progettato principalmente per le reti di sensori. A differenza delle tradizionali architetture hardware, dove disponiamo di grandi quantità di memoria, complessi sottosistemi per la gestione dei dati di ingresso e per quelli d'uscita, forti capacità di elaborazione e sorgenti di energia praticamente illimitate, nelle reti di sensori ci troviamo a confronto con sistemi
di piccole dimensioni, fonti di energia limitate, scarsa quantità di memoria, modeste capacità di elaborazione, etc. Sono necessarie quindi soluzioni molto semplici ed efficienti, e che soprattutto riducano la massimo i consumi di energia. Anche il sistema operativo risente di queste limitazioni. Infatti TinyOS non possiede un nucleo ma permette l'accesso diretto all'hardware, inoltre sparisce i concetto di processore virtuale per evitare i cambi di contesto,

e quello di memoria virtuale: la memoria viene infatti considerata come un unico e lineare spazio fisico, che viene assegnato alle applicazioni a tempo di compilazione. In pratica viene eliminato qualsiasi tipo di overhead, poichè nelle reti di sensori questo causa un inutile dispersione di energia senza portare tangibili vantaggi.
Queste scelte impattano direttamente sull'architettura che risulta molto compatta e ben si sposa con le dimensioni ridotte della memoria che a volte raggiunge la quantità di pochi chilobyte.
TinyOS è implementato in NesC un linguaggio di programmazione basato su una logica a componenti creato per essere utilizzato per lo sviluppo di applicazioni in sistemi embedded. Un sistema embedded, avendo delle funzionalità note già durante lo sviluppo, potrà eseguirli grazie ad una combinazione hardware/software specificamente studiata per la tale applicazione riducendo lo spazio occupato ed il costo di produzione. Lo scopo dichiarato dei progettisti di TinyOS era infatti quello di
- ridurre i consumi di energia;
- ridurre il carico computazionale e le dimensioni del sistema operativo;
- supportare intensive richieste di operazioni che devono essere svolte concorremente e in maniera tale da raggiungere un alto livello robustezza ed un efficiente modularità.

Per soddisfare il requisito della modularità, TinyOS favorisce lo sviluppo di una serie di piccoli componenti, ognuno con un ben precisa funzione, che realizza un qualche aspetto dell'hardware del sistema o di un'applicazione.
Ogni componente poi, definisce un interfaccia che garantisce la riusabilità del componente ed eventualmente la sua sostituzione. Guardando in modello a strati del sistema operativo (figura x.y), possiamo intuire come sia facilitato

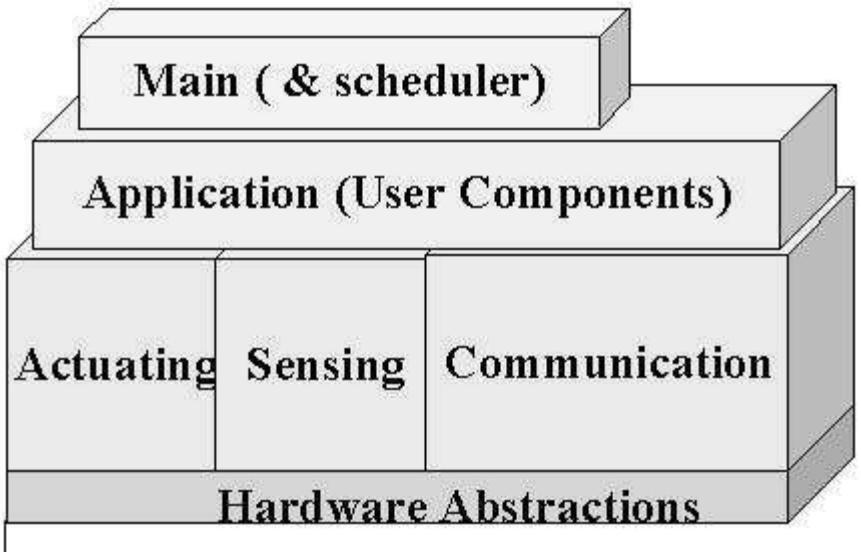

Figura 2.3 : Strati Sistema Operativo

lo sviluppo di applicazioni mediante il riutilizzo dei componenti esistenti. TinyOS è stato implementato seguendo il modello ad eventi. Nei tradizionali modelli infatti, è necessario allocare un certa quantità di memoria sullo stack per ogni attività in esecuzione e inoltre, il sistema è soggetto a frequenti commutazioni di contesto per servire ogni tipo di richiesta, come l'invio di pacchetti, la lettura di un dato su un sensore, etc. Naturalmente i nodi non dispongono di grosse quantità di memoria ed il modello ad eventi ben si addice ad un sistema continuamente sollecitato. Nel campo delle reti di sensori il consumo di energia è un fattore critico e l'approccio basato sugli eventi utilizza il microprocessore nella maniera più efficiente possibile. Quando il sistema viene sollecitato da un evento questo viene gestito immediatamente e rapidamente. In effetti non sono permesse condizioni di bloccaggio nè attese attive che sprecherebbero energia inutilmente. Quando invece non ci sono attività da eseguire il sistema mette a riposo il microprocessore che viene risvegliato all'arrivo di un evento.
TinyOS è un insieme di librerie e applicazioni che possono essere riutilizzate per costruire nuove applicazioni. Le librerie includono i protocolli di rete, i servizi distribuiti, i gestori dei sensori, e degli strumenti per l'acquisizione dei dati. A loro volta le librerie e le applicazioni sono costituite da un insieme di componenti i quali rappresentano i mattoni con cui si costruisce un applicazione.
Un altro elemento importante dell'architettura di TinyOS è lo schedulatore.
Come abbiamo già discusso, TinyOS utilizza un modello di programmazione basato su macchine a stati invece del tradizionale modello basato su thread. Ogni componente transita da uno stato ad un altro mediante una richiesta di operazioni oppure in seguito all'arrivo di un evento. Ogni transizione avviene praticamente in modo istantaneo, cioè non esiste nessun evento che possa interromperla, e richiede pochissime operazioni da parte del processore.
Comunque per operazioni che richiedono lunghe elaborazioni sono previsti speciali meccanismi.
Un altro vantaggio nella scelta di tale modello di programmazione consiste nel fatto che il livello di astrazione prodotto dall'hardware si propaga direttamente nel software, così segnali di interruzione provenienti dall'hardware si traducono immediatamente in eventi software.
Di seguito riportiamo un analisi più dettagliata sia dei componenti sia del
modello si esecuzione.

2.2.1 I componenti

Ogni componente è un unità indipendente e svolge un determinato compito.
L'insieme di componenti che forma un applicazione è infatti costituito da componenti preesistenti (che appartengono a librerie o applicazioni sviluppate in precedenza) e da nuovi componenti, cioè quelli necessari a completare l'applicazione e viene chiamato grafo dei componenti. Il grafo fissa una gerarchia nel senso che il processo di composizione

dell'applicazione produce una stratificazione all'interno della stessa. I componenti di livello inferiore segnalano eventi ai quelli di livello più alto, mentre i componenti di livello superiore richiedono dei servizi ai componenti di livello più basso. Lo strato inferiore infine, è formato da componenti che rappresentano direttamente i dispositivi hardware. Un componente è formato da quattro parti tra loro collegate:

- un insieme di comandi;
- un insieme di eventi;
- un frame;
- un certo numero di task.

Comandi, eventi e task operano all'interno del frame, modificandone lo stato.

Un componente definisce poi delle interfacce che permettono di realizzare applicazioni modulari.

2.2.2 Il Frame

Il frame viene allocato a tempo di compilazione e permette di conoscere non solo la quantità di memoria richiesta da un componente ma quella dell'intera applicazione. Questa soluzione permette di evitare gli sprechi dovuti all'overhead associato alle allocazioni dinamiche. Infatti il sistema operativo non deve compiere delle operazioni per risolvere gli indirizzi in memoria, dato che i riferimenti all'interno del programma corrispondono ad indirizzi fisici.

2.2.3 I Comandi

I comandi sono richieste di un servizio fornito da un componente di livello più basso e non sono bloccanti. Generalmente un comando deposita dei parametri nel frame e poi attiva un task, ma è possibile anche che questo chiami un altro comando. In quest'ultimo caso, il componente non può attendere per un tempo indeterminato la fine della chiamata. Il comando, ritorna poi un valore che indica se la chiamata ha avuto successo o meno.

2.2.4 Gli Eventi

Gli eventi sono, direttamente o indirettamente, dei gestori delle interruzioni hardware. Il componente di livello più basso trasforma un interruzione hardware in un evento che può essere provocato da un interruzione esterna, da timer, o dal contatore, e poi propaga tale richiesta ai livelli più alti. Similmente ai comandi, un evento può depositare dei parametri nel frame e poi attivare un task. Un evento può richiamare altri eventi e alla fine chiamare un comando, come a formare una catena che prima sale e poi scende. Per evitare che questa catena si possa chiudere viene impedito ai comandi di generare eventi.

2.2.5 Modello di concorrenza: i task

TinyOS esegue un solo programma alla volta. Questo è composto da un solo tipo di attività indipendente: il task. I task non hanno diritto di prelazione su nessuna delle attività in corso, in pratica sono atomici rispetto agli altri task. Inoltre possono richiamare comandi, generare eventi o attivare altri task all'interno dello stesso componente. La caratteristica di essere eseguito fino al suo completamento infine. permette di gestire un singolo stack che
viene assegnato, a turno, al task in esecuzione.
Generalmente la scelte del task da mandare in esecuzione viene effettuata secondo il criterio FIFO, ma ciò non toglie che si possa modificare lo schedulatore affinchè realizzi una più sofisticata scelta. Se la coda dei task è vuota lo schedulatore mette a riposo il microprocessore come riporta i seguente frammento di codice estratto dallo schedulatore.

```
main{
....
while(1){
while(more_tasks)
schedule_task;
sleep;
}
}
```

I gestori possono sempre interrompere un task o un gestore stesso nel caso questo abbia priorità maggiore dell'evento attualmente in esecuzione.
La figura sottostante riassume il modello di concorrenza del TinyOS.

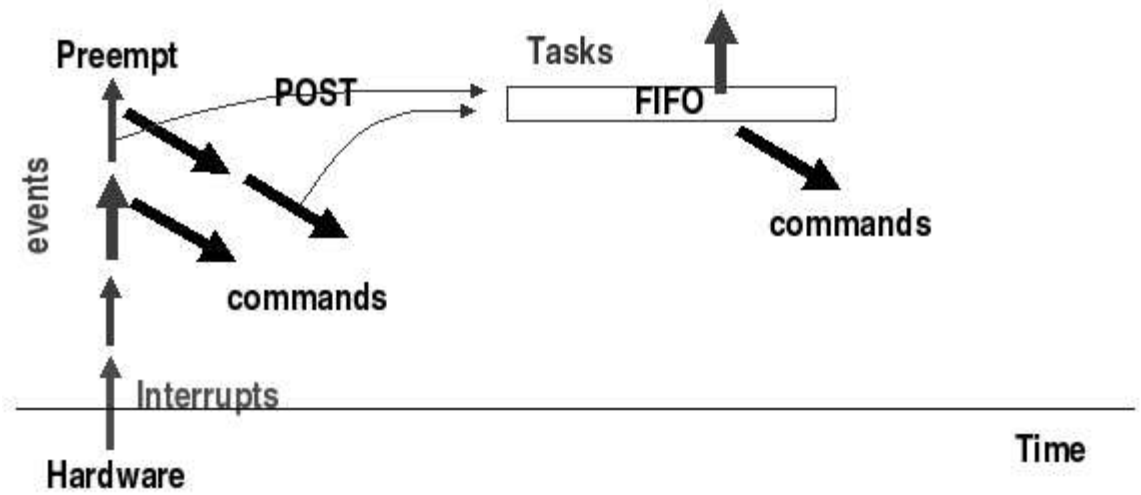

Figura 2.4 :Modello di Concorrenza

2.3 NesC

NesC è un linguaggio di programmazione ideato per lo sviluppo su sistemi embedded. NesC ha una sintassi simile al C e supporta il modello di concorrenza del TinyOS. Le caratteristiche principali del linguaggio nesC sono:

- Separazione della costruzione e della composizione: lo sviluppo di un applicazione in NesC è effettuata assemblando un serie di componenti (preesistenti o scritti appositamente), costituiti da porzioni di codice che implementano le funzioni che verranno utilizzate per costruire l'applicazione, in modo da creare le relazioni tra le interfacce. Ogni componente, assemblato attraverso i file di configurazione, deve definire le specifiche del suo comportamento e implementare tali direttive.

- Specifiche del componente mediante interfacce: un componente deve dichiarare sia le interfacce di altri blocchi che utilizza sia quelle fornite da lui stesso, queste ultime sono le operazioni implementate dal componente stesso e messe a disposizione agli altri. Dato che un'interfaccia descrive una serie di funzioni (con eventuali parametri richiesti e/o restituiti) risulta molto più semplice riutilizzare porzioni di codice per sviluppare nuove applicazioni.

- Interfacce bidirezionali: ogni interfaccia specifica una serie di funzioni che devono essere implementate dal modulo che le fornisce ed un insieme di eventi che devono essere gestiti dall'utilizzatore. Questo permette di realizzare operazioni complesse utilizzando una singola interfaccia.

- Staticità: il grafo che collega i componenti tra loro è statico e non può essere modificato durante l'esecuzione del programma. In questo modo è possibile velocizzare l'esecuzione, irrobustire il codice e analizzare staticamente l'applicazione migliorando l'efficienza del compilatore.

- Concorrenza: il modello concorrenziale di NesC è compatibile con quello caratteristico di TinyOS e consiste in processi che eseguono la loro attività contemporaneamente su dati condivisi.

2.3.1 Le interfacce

Ogni componente definisce le interfacce fornite e quelle usate che rappresentano gli unici punti di accesso per interagire con lo stesso. Un interfaccia dichiara una serie di funzioni che possono essere di due tipi: comandi ed eventi. I comandi devono essere implementati dal componente che li fornisce mentre gli eventi devono essere implementati dal componente che li usa. Un comando generalmente è una richiesta di servizio, mentre l'evento segnala il completamento di un servizio. Gi eventi possono essere generati anche in modo asincrono, per esempio in seguito ad un interruzione hardware o all'arrivo di un pacchetto. Si possono usare o fornire più di un interfaccia e istanze multiple di una stessa interfaccia (ovviamente rinominandole). Di seguito riportiamo il codice di una semplice interfaccia, implementata nel file /tinyos-1.x/tos/interfaces/SendMsg.nc :

```
interface SendMsg
{
command result_t send(uint16_t address, uint8_t length,
TOS_MsgPtr msg);
event result_t sendDone(TOS_MsgPtr msg,
result_t success);
}
```

2.3.2 I componenti

Un componente può esser di due tipi: modulo o configurazione. I moduli implementano le interfacce che dichiarano (sia i comandi \esportati" che gli eventi \importati"). Ecco qui un esempio:

```
module ComponentModule {
provides interface X
provides interface Y
uses interface Z
}
implementation {
// Codice dei comandi esportati dalle interfacce X e Y
...
// Codice degli eventi importati dall'interfaccia Z
...
}
```

Le configurazioni invece definiscono come più componenti devono essere assemblati e possono fornire delle interfacce anche se non vengono implementate esplicitamente (in effetti queste sono implementate in modo implicito dai componenti a cui sono collegate).

```
configuration ComponentConfiguration {
provides interface X
provides interface Y
uses interface Z
}
implementation {
// Codice che definisce come le interfacce X e Y si
// collegano con l'interfaccia Z
...
}
```

2.3.3 Assemblaggio dei Componenti

Per realizzare un applicazione è necessario collegare i vari componenti. Questa operazione viene definita “wiring" e produce il cosiddetto grafo dei componenti.

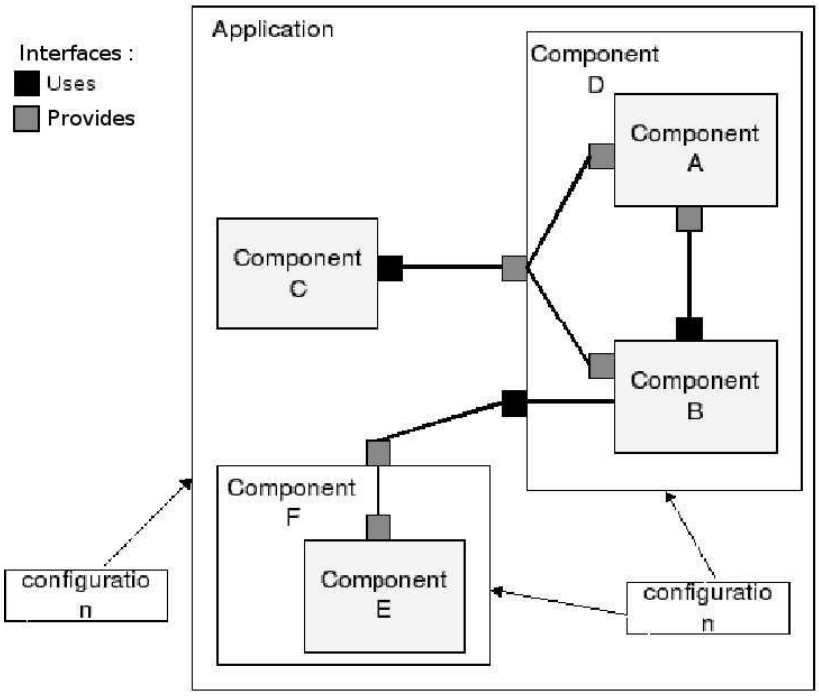

Figura 2.5 : grafo dei componenti

I componenti: ComponentD, - ComponentF, Application, sono delle configurazioni e, come abbiamo già spiegato, servono a collegare tra loro componenti preesistenti. Il resto dei componenti sono dei moduli.
Il componente ComponentD fornisce un interfaccia che verrà usata da ComponentC e utilizza un interfaccia sarà fornita da ComponentF.
Il componente ComponentF fornisce una sola interfaccia che corrisponde a quella fornita da ComponentE Il componente Application costituisce l'applicazione vera e propria. Non utilizza ne fornisce interfacce ma effettua il semplice collegamento tra i componenti ComponentC, ComponentD e ComponentF
Come possiamo notare la ragione per cui un componente si distingue in moduli e configurazioni è per favorire la modularità di un applicazione.
Ciò permette allo sviluppatore di assemblare velocemente le applicazioni. In effetti, si potrebbe realizzare un applicazione solo scrivendo un componente di configurazione che assembli componenti già esistenti. D'altra parte questo modello incoraggia l'aggiunta di librerie di componenti tali da implementare algoritmi e protocolli che possano essere utilizzati in una qualsiasi applicazione.

2.4 Tossim

Il principale scopo di un simulatore è quello di fornire un ambiente controllato e il più fedele possibile della realtà che si intende analizzare. In particolare, le simulazioni forniscono un metodo rapido per testare e validare una soluzione, per confrontare le le alternative proposte o per analizzare quelle interazioni che sono difficilmente osservabili nel sistema reale. Per raggiungere il massimo grado di accuratezza è necessario considerare il comportamento del sistema a tutti i livelli. Non è sufficiente cioè, poter simulare algoritmi e protocolli di rete ma, data la complessità delle interazioni a cui sono soggette le reti di sensori, applicazioni complete. In una rete di sensori infatti, i protocolli e le applicazioni possono interagire tra loro, facendo così scomparire il concetto di stratificazione. Ad esempio, nel processo di aggregazione dei dati, i protocolli di rete dipendono temporalmente dalle letture effettuate dai sensori. Infine, come discusso precedentemente, un fattore critico che caratterizza ogni soluzione studiata per una rete di sensori è la scalabilità: un simulatore dovrebbe esser in grado di supportare reti di migliaia di nodi senza che le prestazioni decadino in modo eccessivo.

Molte ricerche condotte in questo campo hanno concluso che realizzare uno strumento sulla base dei requisiti sopra citati (fedeltà e scalabilità) è praticamente impossibile, per questo motivo sono stati sviluppati simulatori che invece di riprodurre una qualsiasi implementazione di una rete di sensori descrivono un modello dei nodi. Tra tutti citiamo ns-2 e SENSE. Questo tipo di approccio è in gradi di riprodurre i ritardi della della rete, le capacità di trasferimento, le collisioni dei pacchetti, e gli effetti prodotti dall'utilizzo di

meccanismi per la gestione del consumo energetico. Tuttavia, un motivo per cui quest'approccio è risultato poco efficiente, (oltre ad non essere in grado di riprodurre il comportamento di un applicazione) è la differenza tra l'implementazione di un algoritmo fatta per essere testata dal simulatore e quella realizzata per essere installata sui nodi veri e propri. Questo causa due problemi: il primo è che, se le due implementazioni sono estremamente diverse i risultati ottenuti dalla simulazione possono non essere completamente esatti; il secondo si riferisce ad una questione più pratica: bisogna implementare due volte la stessa soluzione.

Tossim fornisce un ambiente di simulazione per reti di sensori aderente al sistema operativo TinyOS ed è stato progettato sulla base dei criteri sopra esposti, per questi motivi è ampiamente utilizzato dalla comunitàà di ricercatori e rappresenta un termine di paragone per lo sviluppo di nuovi simulatori.

2.4.1 Dettagli Tossim

Tossim è un simulatore ad eventi discreti per applicazioni basate sul sistema operativo TinyOS. Il principale obbiettivo di Tossim è quello di simulare nel modo più fedele possibile le

applicazioni realizzate per TinyOS. Infatti permette all'utente di testare e analizzare un applicazione, ed eliminarne gli errori, eseguendola in ambiente controllato e ripetibile. Tossim comunque non è la soluzione per tutti i problemi in quanto non tiene conto di alcuni aspetti del mondo reale. Per chiarire quali sono le possibilità offerte da questo strumento ne analizzeremo le caratteristiche.

2.4.2 Modello di esecuzione

Il cuore del modello di esecuzione di Tossim è la coda degli eventi. Gli eventi Tossim (che sono diversi dagli eventi TinyOS) sono generati dal sistema (per scandire il trascorrere del tempo, in fase di inizializzazione, etc.), dai comandi ed agli eventi TinyOS. Ad ogni evento è associato l'identificativo di un nodo, che permette di associare l'azione corrispondente ad un nodo, l'istante temporale in cui l'evento dovrà essere eseguito ed una funzione che rappresenta il gestore dell'evento. Gli eventi vengono accodati seguendo un ordine temporale: da quello più imminente a quello più lontano. Il gestore di un evento è tipicamente il gestore di un interruzione (nell'astrazione hardware dei componenti), il quale segnala eventi e richiama comandi TinyOS. Dopo ogni evento Tossim, il simulatore esegue i task della coda fino a che questa non è vuota. Questo tipo di soluzione non aderisce al comportamento del TinyOS. In effetti viene eliminata in questo modo il diritto di prelazione che hanno i gestori degli eventi TinyOS sui task.

2.4.3 Modello radio

Tossim fornisce un semplice ma efficace modello per simulare la connettività dei nodi. Questo consiste in un grafo, dove ciascun nodo rappresenta un nodo sensore e gli archi rappresentano le connessioni tra i nodi. Gli archi sono unidirezionali e a ciascuno di essi è associato un numero che rappresenta la Bit Error Rate (BER). In pratica un arco da un nodo x a un nodo y (x,y), rappresenta la probabilità di alterazione di un bit a cui è soggetta la trasmissione di un pacchetto dal nodo x verso y, e non il contrario. Tossim permette la costruzione di questo grafo o utilizzando strumenti esterni o specificandone le caratteristiche in un file secondo una ben definita sintassi. È previsto inoltre un modello dove ogni nodo può comunicare con tutti gli altri senza che si verifichino perdite di pacchetti a causa del mezzo trasmissivo.
Anche se questo modello non è in grado di simulare la propagazione delle onde radio, riproduce alcuni problemi che si verificano nella trasmissione dei pacchetti, come il mancato riconoscimento del simbolo di start, la corruzione dei dati, etc.

2.4.4 Livello Data Link

Tossim simula il comportamento del sistema di comunicazione a livello di bit.
In particolare è in grado di riprodurre il problema del nodo nascosto e può simulare errori in tutte le fasi della ricezione del pacchetto. Il problema del nodo nascosto si verifica quando due nodi , a e b, intendono comunicare con un terzo nodo c, ma entrambi non sono in grado di ascoltare le trasmissioni dell'altro. Se a e b inviano contemporaneamente un pacchetto a c, quello che ne risulta è un segnale disturbato. Tuttavia il modello simulato non prende in considerazione tutti i possibili problemi che possono sopraggiungere durante una trasmissione e che riguardano il riconoscimento del segnale, la sincronizzazione, la codifica dei dati, il mancato riscontro, etc.

2.4.5 Consumo Energetico

Tossim non fornisce nessuna informazione sul consumo energetico da parte dei nodi tuttavia esistono degli strumenti che ne permettono il calcolo come PowerTossim. Diversamente viene fornita una tecnica, applicabile utilizzano Tossim per il calcolo dei consumi. Questa tecnica consiglia di annotare lo stato dei componenti che consumano energia durante tutta la simulazione e poi applicare il modello energetico calcolando il consumo totali a fine simulazione.
Tuttavia qualsiasi tecnica adottata risulta leggermente imprecisa perchè Tossim non modella il tempo di esecuzione del processore.

2.4.6 Architettura

L'architettura del Tossim è composta da cinque parti:
- il supporto per la compilazione delle applicazioni TinyOS per il simulatore.

Il compilatore ncc è in grado di compilare applicazioni sia per l'hardware che per il simulatore, modificando alcune opzioni;
- la cosa degli eventi discreti;
- una serie di reimplementazioni di componenti relative all'hardware di basso livello. Questo permette la sostituzione dei componenti di basso livello in base alla piattaforma per cui vengono compilate (pc, mica, micadot);
- un meccanismo che permette di modellare la connettività tra i nodi e il fenomeno da analizzare;
- i servizi di comunicazione che permettono ad applicazioni esterne di interagire con la simulazione.

2.4.7 Servizi Di Comunicazione

Tossim fornisce dei meccanismi che consentono ad un programma esterno di visualizzare lo stato della simulazione, di controllarne alcuni parametri e di comunicare con i nodi utilizzando una connessione TCP/IP. La comunicazione tra il simulatore e l'applicazione esterna è definita da un protocollo che regola lo scambio di eventi e comandi. Tipici eventi sono i messaggi di debug che il programmatore inserisce nel codice, l'invio di pacchetti radio, le letture dei sensori, etc. I comandi invece possono spegnere o accendere un determinato nodo, iniettare pacchetti radio nella rete, e altri.

Capitolo 3

Mobilità nella WSN

3.1 Protocollo di Routing HAT : Hierarchical Addressing Tree

Il protocollo di instradamento, spunto di partenza sul quale basarsi per poi inserire i meccanismi di mobilità, è conosciuto in letteratura come HAT (Hierarchical Addressing Tree)[9].
Le rete si costituisce con uno schema ad albero assegnando ai nodi indirizzi gerarchici [Figura 3.1].

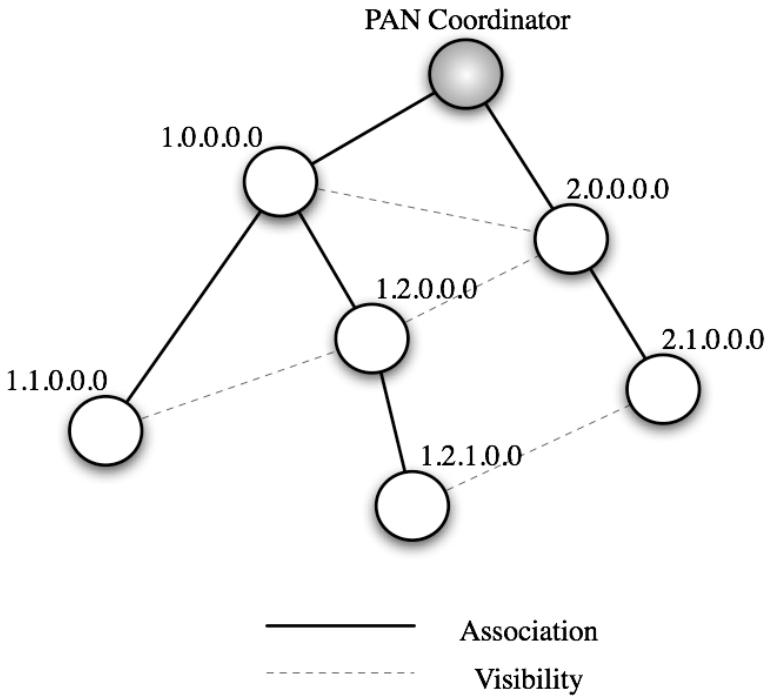

Figura 3.1 : Struttura ad Albero di HAT

Il nodo radice dell'albero di PAN (Personal Area Network) Coordinator è il primo ad attivarsi e inizialmente è l'unico a poter accettare richieste di associazione. Un nodo che vuole far parte della rete attende per un tempo prefissato di ricevere i beacon (A_BCN) dai vicini, dai quali ottiene informazioni che registra in una tabella [Figura 3.2].

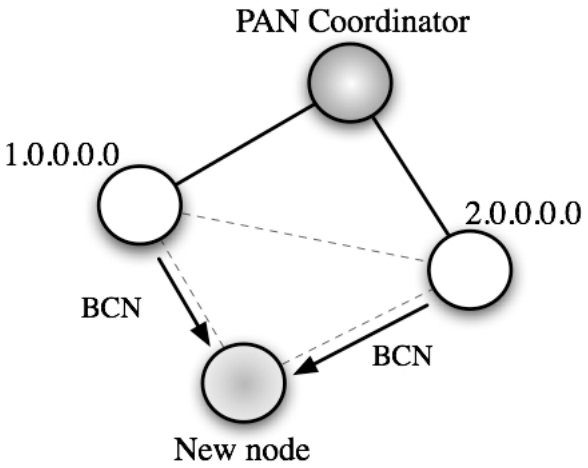

Figura 3.2 : Pacchetti di Beacon dai nodi vicini

Scaduto questo tempo di attesa, il nodo riordina la tabella in base ad una metrica prefissata che può tener conto del numero di hop del nodo vicino rispetto al PAN Coordinator, del numero di nodi già associati al nodo e soprattutto della potenza ricevuta. Attualmente viene utilizzata una funzione di costo che tiene presenti tutti e tre questi valori:

```
cost_function = median_rssi - 3 * num_child - 2 * route_cost
```

Una volta riordinata la tabella dei vicini, il nodo richiedente invia in unicast una richiesta di associazione (A_REQ) al nodo in testa alla tabella. Tale richiesta viene ricevuta dal nodo prescelto che risponde in unicast con un messaggio di risposta (A_RES) in cui è contenuto un indirizzo di rete valido assegnato al nuovo nodo. Il nodo richiedente una volta ricevuto il messaggio di risposta (A_RES) salva le informazioni riguardo al nodo prescelto e all'indirizzo assegnato e risponde con un messaggio di conferma (A_CFR) [Figura 3.3]. A questo punto il nodo prescelto diviene padre del nodo richiedente e salva le informazioni relative al nuovo figlio. Il nodo richiedente memorizza a sua volta le informazioni del nodo padre e invia un messaggio di livello middleware (MOTE_ANNOUNCEMENT) al PAN Coordinator comunicando il proprio indirizzo di rete, il proprio indirizzo MAC e quello del nodo.

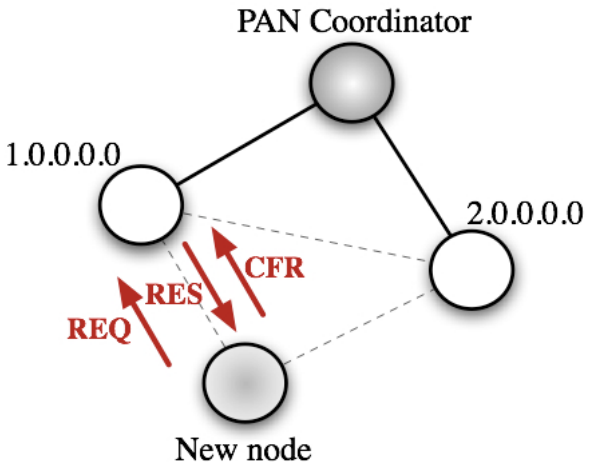

Figura 3.3 : Associazione Three Way Handshake

Tutti i messaggi del protocollo HAT hanno la seguente struttura:
_ `Packet Type`: identifica il tipo di messaggio inviato;
_ `Request node AM Address`: indirizzo MAC del nodo che invia il messaggio;
_ `Response node AM Address`: indirizzo MAC del nodo che invia il messaggio di risposta;
_ `Response node Network Address`: indirizzo di rete del nodo che invia un messaggio di risposta;
_ `Assigned Network Address`: indirizzo di rete assegnato;
_ `Sequence Number`: per associare un nodo di richiesta ad un nodo di risposta;
_ `Route Cost`: indica il numero di hop dal nodo alla radice;
_ `Assman Field`: indica il numero di figli del nodo e se esso è un nodo mobile o fisso.

L'instradamento avviene verificando prima di tutto il campo Packet Scope dell'header di rete. L'indirizzo di destinazione viene preso in considerazione solo se il pacchetto è in unicast. In tal caso il nodo controlla prima di tutto se il destinatario è in visibilità e quindi nella tabella dei vicini o se è il padre o uno dei figli. Se è così inoltra direttamente il pacchetto ad un hop, altrimenti il nodo opera un confronto fra l'indirizzo del nodo destinazione e il proprio, giungendo a due possibili conclusioni:
_ Il nodo destinazione si trova nel sottoalbero di cui il mittente è radice.
Viene determinato il figlio al quale instradare il pacchetto.
_ Il nodo destinazione si trova in un altro sottoalbero quindi il pacchetto viene instradato verso il nodo padre.
Il mantenimento delle associazioni avviene attraverso un periodico controllo di tabelle contenenti informazioni riguardo al nodo padre, ai nodi figlio e ai nodi vicini. Tali tabelle

vengono aggiornate attraverso lo scambio frequente di messaggi di beacon per percepire eventuali cambi topologici. (A-BCN).
Il protocollo HAT supporta, in linea di massima, la mobilità dei nodi quando la topologia della rete cambia, ma non efficacemente e velocemente.
Se il nodo ricevente è mobile, grandi quantità di traffico in down-link si perdono.

3.2 HAT Mobile

HAT Mobile [1] è un miglioramento di HAT volto a supportare la mobilità nella WSN.
HAT Mobile caratterizza i nodi in due categorie : fissi o mobili.
Ogni nodo ha quindi un parametro che li caratterizza, il MOTE_TYPE, che viene fissato in fase di programmazione dei motes in base alla funzione da essi svolta. Tale parametro può assumere il valore di FIXED o MOBILE a seconda che il nodo sia destinato ad essere un nodo con bassa o nulla mobilità oppure con alta mobilità.
Le nuove associazioni sono consentite solo con nodi fissi. Quindi i nodi mobili andranno a configurarsi come le foglie dell'albero nella topologia della rete.
Solo per i nodi mobili è stata introdotta una procedura periodica di controllo della tabella dei vicini, al fine di verificare se c' è qualche altro nodo che ha delle caratteristiche migliori rispetto a quelle dell'attuale nodo padre. Allo scattare del timer possono verificarsi due cose:
- se la potenza del nodo padre scende sotto una determinata soglia e la funzione di costo di uno dei nodi vicini è migliore rispetto a quella del nodo padre, scatta la procedura di Handover e quindi il nodo mobile si assocerà al nodo con caratteristiche migliori;
- il nodo padre è ancora il miglior nodo al quale essere connessi, non viene effettuata la procedura di handover.

3.2.1 Handover : Scelta del nodo padre e funzione di costo

Allo scattare del timer di Handover parte la scansione della tabella dei vicini al fine di trovare un nodo con caratteristiche migliori rispetto a quelle del nodo padre. Per prima cosa viene valutata la potenza del segnale ricevuto dal nodo padre attraverso il valore median_rssi che è la media su tre o cinque valori di potenza calcolati attraverso i beacon ricevuti. Se tale valore è migliore di un parametro predefinito di isteresi, la procedura di handover non ha inizio. La seconda condizione da valutare è quella legata alla funzione di costo dei nodi vicini e del nodo padre, calcolata tramite una funzione in cui compaiono l'rssi medio, il numero di figli e il numero di hop dal PAN Coordinator:

```
cost_function = median_rssi - 3 * num_child - 2 * route_cost
```

A questo punto viene riordinata la tabella dei vicini con tale formula e se la funzione di costo del nodo padre è minore di quella del nodo vicino primo della lista di almeno un valore soglia prede_nito, scatta la procedura di handover. La condizione totale che avvia o meno la procedura di handover risulta essere quindi:

```
(Parent_median_rssi < ISTERESI) AND (Parent_cost_function + THRESHOLD < Neighbour_cost_function)
```

3.2.2 Handover : Gestione dell'associazione e scambio di messaggi

Se la condizione precedentemente descritta viene rispettata parte la procedura di handover con il nodo prescelto. Tale procedura prevede l'invio di messaggi di livello rete di tipo TREE_ROUTING.

Il nodo mobile invia un messaggio di Handover Request (H_REQ) in unicast al nodo scelto (New_Parent) inserendovi l'indirizzo attuale di rete e l'indirizzo MAC.

Il New_Parent che riceve la richiesta, controlla prima di tutto se può accettare un nuovo nodo come figlio; infatti ogni nodo FIXED può accettare un numero di figli massimo pari a MAX_CHILD_PER_NODE.

Successivamente controlla che il suo livello sia minore del livello massimo, cioè che non si trovi ad un numero di hop dal PAN Coordinator superiore o uguale ad un valore prefissato che attualmente è pari a 5; questo perché l'indirizzamento è tale per cui la profondità dell'albero della rete è stabilito in fase di progettazione.

Infine controlla di avere almeno un indirizzo disponibile da assegnare; infatti come vedremo in seguito la procedura di handover blocca alcuni indirizzi di rete che i nodi FIXED possono assegnare.

Se una di queste condizioni non viene rispettata, la procedura di handover con tale nodo fallisce. A questo punto il nodo mobile controlla se nella sua tabella dei vicini c'è un altro nodo che rispetta la condizione di handover precedentemente descritta e avvia un'altra procedura di handover. Altrimenti non avviene alcuna nuova associazione.

Se invece queste condizioni vengono rispettate, il New_Parent salva i dati contenuti nell'H_REQ, stoppa l'invio di Beacon e risponde con un messaggio di Handover Response (H_RES) in cui inserisce il suo indirizzo di rete e MAC e soprattutto un indirizzo di rete valido da assegnare al nodo mobile.

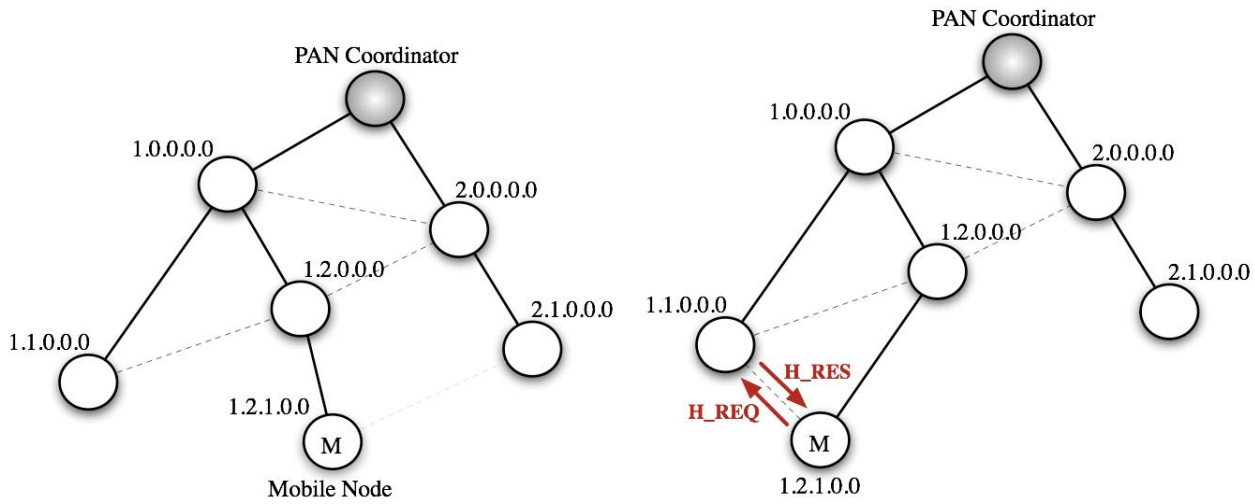

Figura 3.4 : Il Nodo Mobile si sposta e inizia la procedura di Handover con nuovo nodo padre

Una volta che il nodo mobile riceve il messaggio di H_RES, memorizza il nuovo indirizzo di rete assegnatogli e il New_Parent nella ParentEntry [Figura 3.4]. A questo punto rimane solo da comunicare la nuova associazione al vecchio nodo padre (Old_Parent) al quale prima era associato. Ciò è necessario affinché l'Old_Parent aggiorni la sua ChildEntryTable e per avviare la procedura di gestione locale di handover che vedremo successivamente. Il nodo mobile perciò invia un messaggio di Handover Update (H_UPD) all'Old_Parent in cui vengono inseriti il proprio indirizzo MAC, il nuovo indirizzo di rete e l'indirizzo MAC del New_Parent. Se l'H_UPD viene ricevuto correttamente, la procedura di Handover termina.

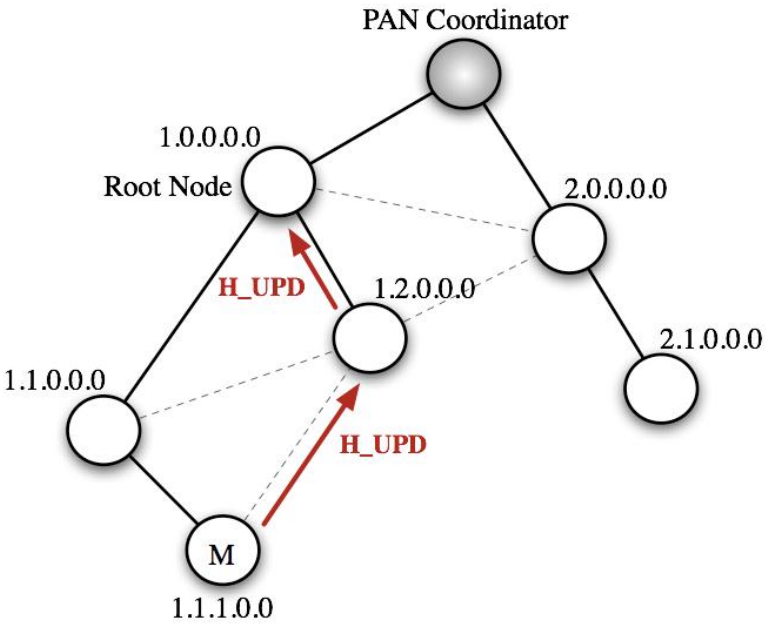

Figura 3.5 : Handover rimane locale fino al nodo radice di vecchio-nuovo padre

L'H_UPD viene utilizzato per gestire anche altre procedure, per questo motivo si è utilizzato il campo seq_num per discriminarle. Per gestire la fase di handover il seq_num del messaggio viene settato al valore 0. É importante sottolineare che i tre messaggi della procedura di handover (H_REQ, H_RES e H_UPD) sono seguiti dal un pacchetto di ACK e ciò rende robusta la procedura.

3.2.3 Gestione locale dell'handover

E' stata creata una procedura di gestione della mobilità a livello locale, grazie allo scambio di messaggi solo in sotto-rami dell'intero albero. La procedura realizzata propone l'utilizzo di tabelle di handover memorizzate in ogni nodo FIXED della rete, per la gestione delle informazioni relative all'handover. Come già detto il nodo mobile invia un H_UPD al vecchio padre per comunicare la nuova associazione con un nuovo padre. Il messaggio di H_UPD ricevuto dal vecchio padre viene inoltrato fino alla radice dei due nodi FIXED coinvolti nell'handover [Figura 3.5]. Nella peggiore delle ipotesi, il nodo radice è il PAN Coordinator e quindi equivale ad inviare un mote_announcement al gateway, nella maggior parte delle volte invece tale nodo è solo la radice di un sottoalbero. Quando il nodo radice riceve l'H_UPD, memorizza in una tabella le informazioni riguardanti l'handover in modo da poter poi instradare correttamente i messaggi destinati al nodo mobile in questione.
Il vecchio nodo padre ora inoltra al suo nodo padre che è la radice del sotto-albero formato da esso stesso e dal nuovo nodo padre. In questo modo non è necessario propagare l'informazione fino al PAN Coordinator. Il nodo radice a questo punto è capace di instradare correttamente i pacchetti destinati al nodo mobile in quanto ne conosce il vecchio e il nuovo indirizzo di rete.

3.2.4 Tabelle di handover e instradamento

Sono state introdotte in ogni nodo FIXED della rete delle tabelle di handover. Tali tabelle sono composte da un numero di Entry pari al valore MAX_CHILD_PER_NODE e ogni Entry ha i seguenti campi:
- MAC Address: indirizzo MAC del nodo mobile;
- Old Network Address: indirizzo di rete del nodo mobile prima della procedura di handover;
- New Network Address: indirizzo di rete assegnato al nodo mobile dopo la procedura di handover;
- Parent MAC Address: indirizzo MAC dell'attuale nodo padre.

All'arrivo di un H_UPD con seq_num pari a 0 ogni nodo della rete si comporta allo stesso modo. Viene analizzato il messaggio ricevuto e a seconda dei valori contenuti e della propria handover table possono verificarsi i seguenti casi:
- Nell'handover table non c'è nessuna entry relativa al MAC Address del nodo mobile contenuto nell'H_UPD. Ciò vuol dire che è la prima volta che tale nodo è coinvolto

nell'handover di quel particolare nodo mobile. In questo caso vengono inseriti nella prima entry libera dell'handover table le informazioni contenute nell'H_UPD.
- Il MAC Address in questione si trova già nella tabella di handover e il valore relativo al nuovo indirizzo di rete contenuto nell'handover table è uguale al valore del vecchio indirizzo di rete contenuto nell'H_UPD. Questo evento si verifica nel caso in cui il nodo mobile faccia due handover in due rami dello stesso nodo radice.
Nell'esempio vediamo il nodo mobile inizialmente connesso al nodo B (con un indirizzo x), successivamente si muove e provoca l'handover con il nodo C (a questo punto il nodo mobile ha un indirizzo pari ad y).

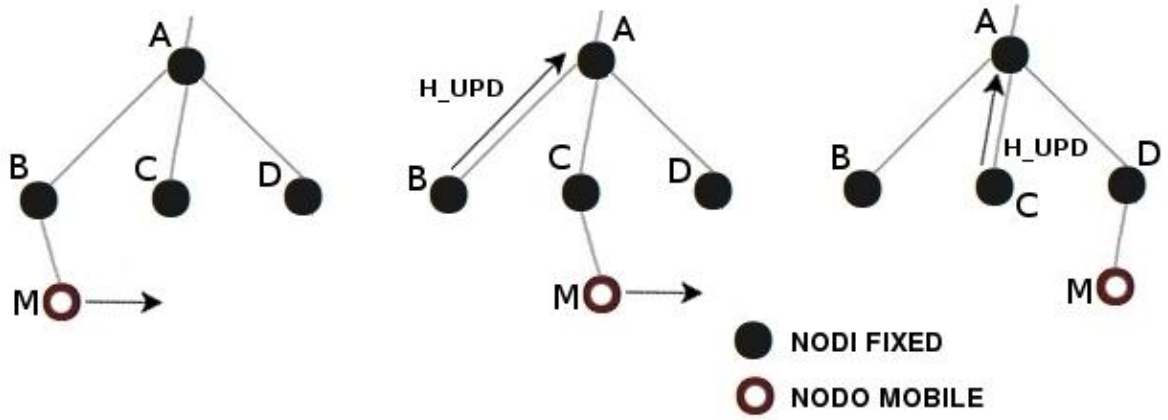

Figura 3.6 : Doppio Handover nello stesso albero

A questo punto il nodo B invia un H_UPD al nodo radice A, il quale aggiorna la sua handover table.

Infine il nodo mobile fa un ultimo spostamento e scatena l'handover con il nodo D (gli viene assegnato l'indirizzo z); il vecchio nodo padre C invia un H_UPD al nodo radice A che aggiorna la sua handover table di conseguenza [Figura 3.6].
- Il MAC Address del nodo mobile si trova già nella tabella di handover e il valore relativo al vecchio indirizzo di rete contenuto nell'handover table è uguale al valore del nuovo indirizzo di rete contenuto nell'H_UPD. Questo evento si verifica quando il nodo mobile si connette ad un nuovo nodo e in seguito si riconnette al nodo al quale era connesso precedentemente. In questo caso il nodo radice che riceve l'H_UPD non fa altro che cancellare la entry relativa al nodo mobile, perché sarebbe un'informazione inutile.
- Il MAC Address del nodo mobile si trova già nella tabella di handover, ma non si verifica nessuna delle condizioni precedenti. Semplicemente il nodo radice sovrascrive le informazioni ricevute.

Per quanto riguarda l'instradamento [Figura 3.7], ciascun nodo controlla prima di tutto la propria handover table. Se l'indirizzo del destinatario del pacchetto si trova nella tabella, vuol dire che il nodo destinatario ha effettuato un handover e quindi ha cambiato indirizzo. É perciò necessario modificare il campo Network Destination Address del pacchetto con il nuovo indirizzo contenuto nell'handover table in modo da poter essere inoltrato correttamente. Successivamente può essere avviata la normale procedura di inoltro dei pacchetti, in cui viene scelto il next hop in base a dei confronti tra l'indirizzo del destinatario e l'indirizzo del nodo che deve inoltrare il pacchetto.

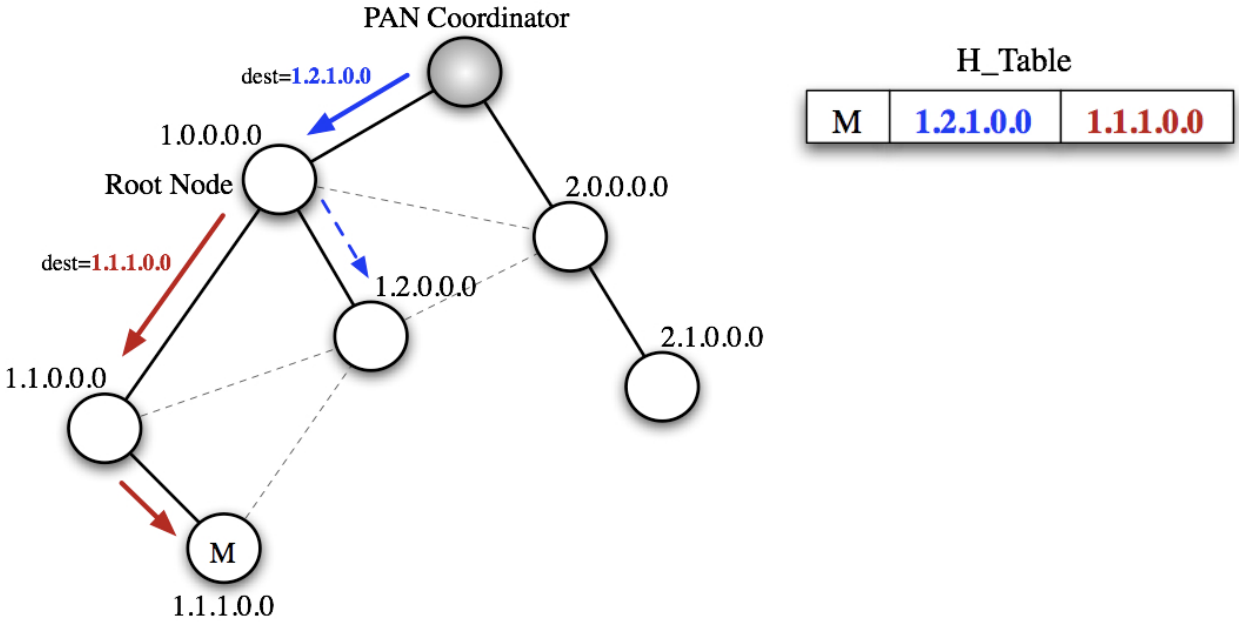

Figura 3.7 : Instadamento in HAT Mobile

3.2.5 Implementazione mobilità

L'implementazione dei meccanismi di handover e delle handover table è stata preceduta da uno studio accurato dei componenti preesistenti del livello rete e del protocollo di instradamento HAT, al fine di integrare al meglio il nuovo codice.

Si è cercato inoltre, quando possibile, di non modificare le procedure già esistenti e di riutilizzare interfacce e moduli in modo da non stravolgere la struttura complessiva già esistente.

I moduli principali dell'implementazione del livello network e del protocollo HAT e i collegamenti fra di essi sono i seguenti:

- NetworkP: implementa la logica del livello rete. Fornisce le primitive per l'invio di pacchetti di livello rete e genera gli eventi di ricezione di pacchetti di livelli inferiori;
- NetworkManagerP: implementa il protocollo HAT. Fornisce le primitive di inoltro dei pacchetti e di scelta del percorso di instradamento attraverso il comando findRoute;
- AddressP: fornisce le primitive per la gestione degli indirizzi;

- ForwardingManagerP: gestisce le code di invio dei pacchetti e il controllo della ricezione di pacchetti duplicati dal livello MAC;
- RoutingTableP: fornisce le primitive per la gestione delle ParentEntry, ChildEntryTable e NeighbourEntryTable. Inoltre in questo lavoro di tesi sono state inserite diverse primitive per la gestione delle HandoverTable;
- RssiControlP: fornisce le primitive per la gestione dei buffer per il calcolo delle potenze ricevute medie;
- AssociationManagerP: fornisce i timer e le primitive per l'invio e la ricezione dei messaggi necessari per consentire ad un nodo di associarsi alla rete;
- AssociationMantainerP: gestisce il mantenimento della connessione di un nodo alla rete. Inoltre sono state inserite diverse primitive per la gestione della mobilità dei nodi sensore.

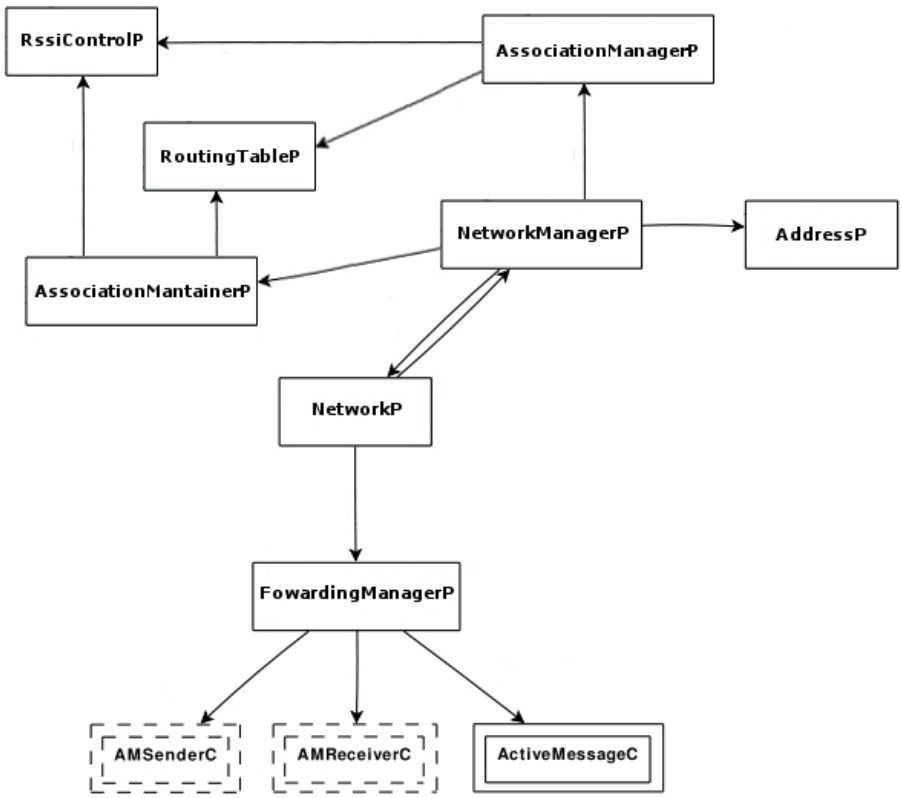

Figura 3.8 : Diagramma moduli principali del livello network

3.2.6 Timer

Come accennato in precedenza il modulo principale che gestisce la mobilità e quindi la procedura di handover è l'AssociationMantainerP. All'interno di tale modulo sono stati definiti quattro timer:

- HandoverTimer: è il timer principale per la gestione dell'handover. Temporizza il controllo della tabella dei vicini finalizzato alla ricerca di un nodo con caratteristiche migliori rispetto a quelle del nodo padre;
- ResendRequestHandoverTimer: regola l'invio delle richieste di handover;
- ResendUploadHandoverTimer: regola l'invio dei messaggi di handover update;
- ResendResponseHandoverTimer: regola l'invio delle risposte alle richieste di handover;

I primi tre timer sono implementati solo dai nodi mobili, mentre l'ultimo solo dai nodi fissi.

3.2.7 Gestione dell'associazione

Se allo scattare dell'handoverTimer il nodo mobile decide di effettuare un handover, parte la procedura di creazione di una nuova associazione attraverso l'invio di un messaggio di richiesta di handover. La gestione dell'associazione è regolata dalle seguenti funzioni:

-processHandoverRequest: viene lanciata alla ricezione di un H_REQ; blocca l'invio di beacon e il controllo delle tabelle, assegna un indirizzo al nodo richiedente e invia l'H_RES;

- processHandoverResponse: viene lanciata alla ricezione di un H_RES; viene memorizzato il nuovo indirizzo assegnato e le informazioni riguardanti il nuovo padre;

- processHandoverConfirm: viene lanciata alla ricezione dell'ack di avvenuta ricezione dell'H_RES da parte del nodo mobile; memorizza i dati del nuovo figlio e fa ripartire l'invio dei beacon e il controllo delle tabelle;

3.2.8 Gestione tabelle di handover

Come già detto le handover table sono state implementate al fine di mantenere locale l'informazione riguardante le associazioni dei nodi mobili. Ogni entry della tabella è formata da quattro campi: MAC nodo mobile, vecchio indirizzo, nuovo indirizzo e indirizzo MAC dell'attuale nodo padre.

Appena l'associazione con un nuovo nodo è avvenuta con successo, il nodo mobile invia un H_UPD con seq_num settato a 0 al vecchio nodo padre.

Quest'ultimo inoltra l'H_UPD al nodo radice del sottoalbero formato con i due nodi interessati dall'handover. Alla ricezione di un H_UPD viene lanciata la funzione processUploadRequest che cancella dalla ChildEntryTable il nodo figlio che ha attuato l'handover e aggiorna l'handoverTable.

Capitolo 4

Dettagli Implementativi

Le features di ottimizzazione nella gestione della mobilità di HAT MOBILE sono inserite in un codice per sensori facente parte di una versione protocollare "vecchia". Al contrario la nuova versione protocollare alla partenza di questo lavoro di Tesi, era sì l'ultima versione del branch di svilippo ma non presentava le caratteristiche di HAT Mobile, era solo incorporato la versione HAT (Hierarchical Addressing Tree). Si è dovuto intervenire quindi per portare nella versione più recente del codice sensori tutte quelle implementazioni proprie di HAT Mobile per la gestione della mobilità all'interno dell'albero e di conseguenza, di gestione degli Handover.

4.1 Criticità più importanti

In seguito analizziamo quali sono state le criticità nel lavoro di "merge", di fusione, andando ad analizzare alcuni dettagli implementativi.

1) codice_sensori\middlewareMessageType\MiddlewareMessageType.h.

Sono stati portati nella nuova versione protocollare i tipi MOTE_UPDATE, MOBILE_PING_DOWN, MOBILE_PING_UP, SENSOR_READ_RESPONSE e SENSOR_READ che sono specifici di HAT MOBILE.

2) codice_sensori\network\address\AddressP.nc

La nuova versione era sprovvista nella struttura dati nwk_addr_entry_t del campo am_addr_t node_assigned, ovvero dell' indirizzo MAC dei nodi.
Sono stati aggiunti alla nuova versione:
- command bool isInMyChildNetworkAddressSpace(network_addr_t nwk_addr);
- command network_addr_t Address.getAddress
- command am_addr_t Address.getAmAddress
- command bool Address.isThisAddressAvailable
- command bool isInMyChildNetworkAddressSpace(network_addr_t nwk_addr);

3) codice_sensori\network\association\AssociationManagerP.nc

Va integrato nella nuova versione protocollare quasi nella sua totalità, in quanto è il modulo principale per consentire le associazioni di nuovi nodi alla rete.

4) codice_sensori\network\association\AssociationMantainerP.nc

Va integrato nella nuova versione protocollare quasi nella sua totalità, in quanto è il modulo principale per mantenere le connessioni, una volta create e per la gestione della mobilità dei nodi sensore. E' quindi il modulo chiave nel codice di HAT MOBILE ed è il più critico da integrare nella nuova versione protocollare.

5) codice_sensori\network\engine\NetworkP.nc

Il valore corretto della variabile indirizzo di destinazione è il valore di ritorno della chiamata alla funzione RoutingTable.getHandoverNwkAddr2. In altra parole si interrogano le Handover Table per sapere a chi inoltrare i pacchetti.
Questa caratteristica è stata quindi inserita nella nuova versione del codice.

```
dst_addr = call RoutingTable.getHandoverNwkAddr2(iCont);
```

6) codice_sensori\network\routing\RountingTableP.nc

La vecchia versione protocollare utilizza tre tabelle distinte che ogni nodo deve possedere: la tabella dei vicini (neighbour table), la tabella dei padri (parent tabele) e la tabella dei figli (children table).

```
parent_node_t parent_entry;
child_node_t child_entries[MAX_CHILD_PER_NODE];
neighbour_node_t neighbour_entries[MAX_NEIGHBOUR_ENTRIES];
```

Nel dettaglio, la `ParentEntry` contiene le informazioni sul nodo con cui è stata portata a termine la procedura di associazione:

-`Network Address`: indirizzo di rete del nodo padre;
-`AM Address`: indirizzo di livello MAC del nodo padre;
-`Route Cost`: numero di hop dal nodo padre al PAN Coordinator;
-`Beacon Counter`: utilizzato per il mantenimento dell'associazione con il nodo padre;
-`Median rssi`: received signal strength indicator medio ricevuto dal nodo padre. Indicatore della potenza ricevuta in un canale radio.

Le informazioni sui nodi figlio sono mantenute nella `ChildEntryTable`:
-`Network Address`: indirizzo di rete del nodo figlio;
-`AM Address`: indirizzo di livello MAC del nodo figlio;
-`Beacon Counter`: utilizzato per il mantenimento dell'associazione con il nodo figlio;
-`Median rssi`: rssi medio ricevuto dal nodo figlio.

Le informazioni sui nodi vicini sono mantenute in un a `NeighbourEntryTable`:
-`Network Address`: indirizzo di rete del nodo vicino;
-`AM Address`: indirizzo di livello MAC del nodo vicino;
-`Median rssi`: rssi medio ricevuto dal nodo vicino.
-`Beacon Counter`: utilizzato per il mantenimento del nodo vicino nella tabella;
-`Num. Child`: numero di figli del nodo vicino;

Questa però è una ridondanza, un utilizzo eccessivo di risorse, di memoria.
La nuova versione protocollare ottimizza invece questo aspetto.
Utilizza una tabella dei padri (`parent table`) e una tabella in cui gestisce sia i figli che i vicini (`neighbour table`) . In quest'ultima tabella puntano due array distinti : l'indice dei vicini (`index_neighbour`) e l'indice dei figli (`index_child`).

```
parent_node_t parent_entry;
neighbour_node_t neighbour_entries[MAX_NEIGHBOUR_ENTRIES];
uint8_t index_child[MAX_CHILD_PER_NODE];
uint8_t index_neighbour[MAX_NEIGHBOUR_ENTRIES];
```

La vecchia versione utilizza anche un' ultima tabella `handover_entries`, specifica del protocollo HAT MOBILE, che quindi andrà integrata.

```
handover_table_t handover_entries[MAX_CHILD_PER_NODE];
```

I seguenti comandi, poi, sono anch'essi necessari per il funzionamento di HAT MOBILE e vanno quindi integrati:

```
command error_t clearParentTable()
command child_node_t* getChildTable()
command int8_t ParentCostFunction()
command int8_t NeighbourCostFunction(uint8_t ind)
```

```
command error_t setHandoverTable(...)
command uint8_t gethandoverFirstFreeCell()
command network_addr_t getHandoverAmAddr(uint8_t ind)
command network_addr_t getHandoverNwkAddr1()
command network_addr_t getHandoverNwkAddr2()
command error_t setHandoverParentAmAddr()
command am_addr_t getHandoverParentAmAddr()
```

Capitolo 5

Verifica Funzionamento e Risultati

5.1 Topologia Lineare. Funzionamento In Installation

Si è voluta creare una topologia lineare come da figura al fine di valutare i meccanismi di mobilità ovvero il buon esito dei meccanismi di handover.

Si è installato il codice su 6 sensori Micaz, uno dei quali lo si è spostato lungo un percorso lineare.

I nodi ancora da 0 a 4 si dispongono e si associano in cascata. Il nodo mobile 5 dapprima procede con una classica associazione col nodo PAN Coordinator (0), poi, man mano che si sposta, esegue l'handover coi nodi 1,2,3 e 4.

Osservando i led sul nodo mobile, programmati in modo tale da rilevare chi è il padre in ogni momento del percorso, si può notare come i led cambino da 000 (nodo 0, pan coordinator) a 100 (nodo 4).

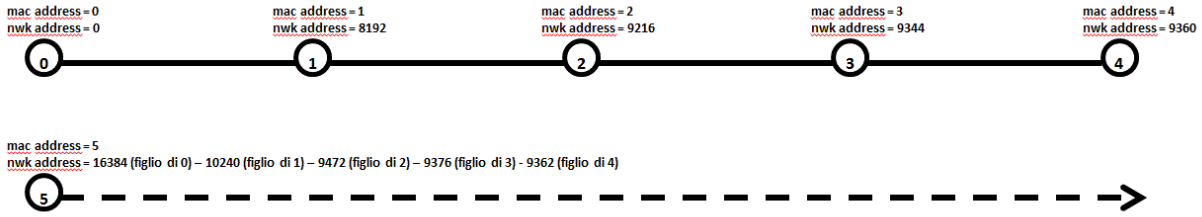

Figura 5.1 : Topologia Lineare.

Figura 5.2 : Test con Sensori reali : Panoramica Topologia e Associazione del mobile con 0


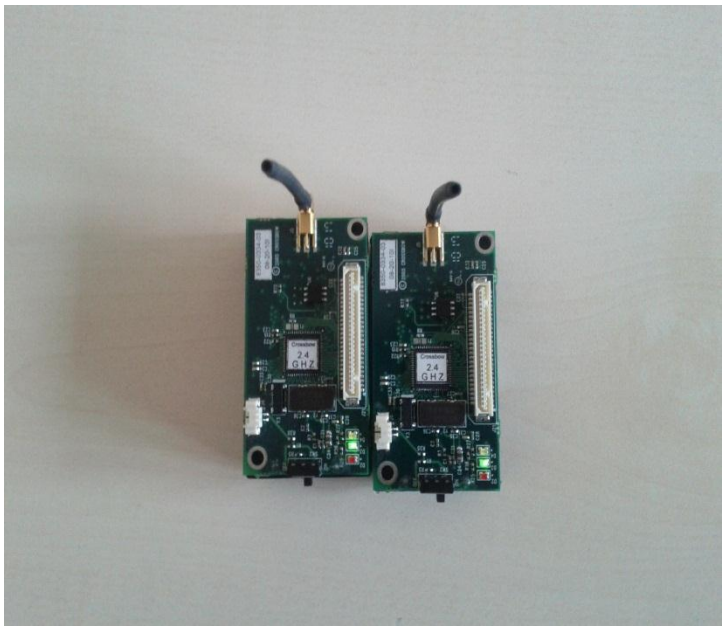

Figura 5.3 : Handover del nodo mobile con 1 e in seguito con 2


Figura 5.4 : Handover del nodo mobile con 3 e in seguito con 4

5.2 Topologia Lineare. Funzionamento in Simulazione

Si è compilato il codice in Simulazione, utilizzando quindi il simulatore di TinyOS TOSSIM, e in seguito si è lanciato uno script python che costruisse la topoologia e il modello di figura.

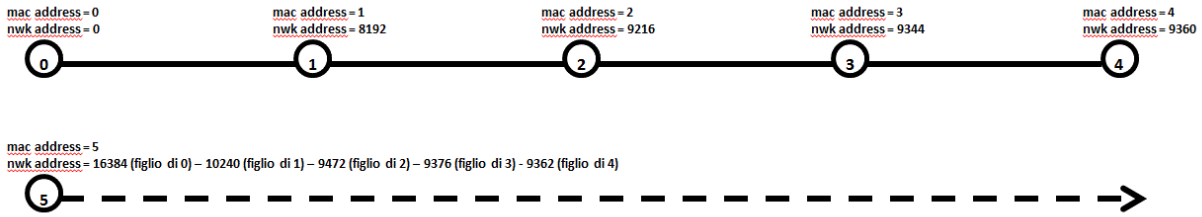

Figura 5.5 : Topologia Lineare.

```
Simulation time : 0:1:18.100585937
DEBUG (0): Received request from node 5
DEBUG (0): AssociationMantainerP__processAssociationRequest: INDIRIZZI DISPONIBILI: 6
MAX_CHILD_PER_NODE: 7
MAX_CHILD_PER_NODE/4)+1: 2
DEBUG (0): Assegno l'indirizzo 16384
DEBUG (0): sending a response to 5 assigning 16384 and child mask 2


Simulation time : 0:2:20.100585937
DEBUG (0): Checking Tables...
DEBUG (5): Checking Tables...
DEBUG (5): HANDOVER >>>>> Handover timer fired!
DEBUG (5): HANDOVER >>>>> rssi del vicino 8192 : -24 è migliore di quello del padre 0 : -35
DEBUG (5): HANDOVER >>>>> invio la richiesta di handover
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (5): HANDOVER >>>>> Sending a request to node 8192
DEBUG (1): HANDOVER >>>>> Received request of handover from node 5
DEBUG (1): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=1,
req_nod_id=5, assigned_addr=10240, res_nod_nwk_addr=16384
DEBUG (1): beacon timer stoppedd...
DEBUG (1): HANDOVER >>>>> sending a response to 5 con network addr: 16384 assigning 10240 and
child mask 2
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (5): rssi_count = 8; rssi del nodo= 3 con rssi=-53
DEBUG (4): beacon from parent
DEBUG (3): Sent beacon number  34..
DEBUG (5): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (5): HANDOVER >>>>> il mio vecchio indirizzo è 16384
DEBUG (5): HANDOVER >>>>> handover andato a buon fine con il nodo 8192
DEBUG (5): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (5): HANDOVER >>>>> Sending H_UPD to node 0 con upd_bit = 0
DEBUG (5): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (1): HANDOVER >>>>> invio dell H_RES avvenuto...
DEBUG (0): HANDOVER >>>>>  H_UPD ricevuto con upd_bit = 0...


Simulation time : 0:2:54.100585937
DEBUG (5): HANDOVER >>>>> Handover timer fired!
DEBUG (5): HANDOVER >>>>> rssi del vicino 9216 : -23 è migliore di quello del padre 8192 : -40
DEBUG (5): HANDOVER >>>>> invio la richiesta di handover
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (0): Checking Tables...
```

```
DEBUG (5): HANDOVER >>>>> Sending a request to node 9216
DEBUG (2): HANDOVER >>>>> Received request of handover from node 5
DEBUG (2): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=1,
req_nod_id=5, assigned_addr=9472, res_nod_nwk_addr=10240
DEBUG (2): beacon timer stoppedd...
DEBUG (2): HANDOVER >>>>> sending a response to 5 con network addr: 10240 assigning 9472 and
child mask 2
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (1): Checking Tables...
DEBUG (5): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (5): HANDOVER >>>>> il mio vecchio indirizzo è 10240
DEBUG (5): HANDOVER >>>>> handover andato a buon fine con il nodo 9216
DEBUG (5): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (5): HANDOVER >>>>> Sending H_UPD to node 8192 con upd_bit = 0
DEBUG (5): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (2): HANDOVER >>>>> invio dell H_RES avvenuto...
DEBUG (1): HANDOVER >>>>>  H_UPD ricevuto con upd_bit = 0...


Simulation time : 0:3:24.101562500
DEBUG (5): HANDOVER >>>>> Handover timer fired!
DEBUG (5): HANDOVER >>>>> rssi del vicino 9344 : -25 è migliore di quello del padre 9216 : -37
DEBUG (5): HANDOVER >>>>> invio la richiesta di handover
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (0): Checking Tables...
DEBUG (5): HANDOVER >>>>> Sending a request to node 9344
DEBUG (5): rssi_count = 16; rssi del nodo= 1 con rssi=-55
DEBUG (2): beacon from parent
DEBUG (1): Sent beacon number  31..
DEBUG (3): HANDOVER >>>>> Received request of handover from node 5
DEBUG (3): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=1,
req_nod_id=5, assigned_addr=9376, res_nod_nwk_addr=9472
DEBUG (3): beacon timer stoppedd...
DEBUG (3): HANDOVER >>>>> sending a response to 5 con network addr: 9472 assigning 9376 and
child mask 2
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (1): Checking Tables...
DEBUG (5): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (5): HANDOVER >>>>> il mio vecchio indirizzo è 9472
DEBUG (5): HANDOVER >>>>> handover andato a buon fine con il nodo 9344
DEBUG (5): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (5): HANDOVER >>>>> Sending H_UPD to node 9216 con upd_bit = 0
DEBUG (5): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (3): HANDOVER >>>>> invio dell H_RES avvenuto...
DEBUG (2): Checking Tables...
DEBUG (2): HANDOVER >>>>>  H_UPD ricevuto con upd_bit = 0..


Simulation time : 0:3:52.104492187
DEBUG (5): HANDOVER >>>>> Handover timer fired!
DEBUG (5): HANDOVER >>>>> rssi del vicino 9360 : -27 è migliore di quello del padre 9344 : -38
DEBUG (5): HANDOVER >>>>> invio la richiesta di handover
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (5): HANDOVER >>>>> Sending a request to node 9360
DEBUG (0): Checking Tables...
DEBUG (4): HANDOVER >>>>> Received request of handover from node 5
DEBUG (4): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=0,
req_nod_id=5, assigned_addr=9362, res_nod_nwk_addr=9376
DEBUG (4): beacon timer stoppedd...
DEBUG (4): HANDOVER >>>>> sending a response to 5 con network addr: 9376 assigning 9362 and
child mask 1
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (1): Checking Tables...
DEBUG (0): child seen: 1
DEBUG (3): Checking Tables...
DEBUG (2): Checking Tables...
DEBUG (5): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (5): HANDOVER >>>>> il mio vecchio indirizzo è 9376
DEBUG (5): HANDOVER >>>>> handover andato a buon fine con il nodo 9360
```

```
DEBUG (5): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (5): HANDOVER >>>>> Sending H_UPD to node 9344 con upd_bit = 0
DEBUG (5): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (4): HANDOVER >>>>> invio dell H_RES avvenuto...
DEBUG (5): rssi_count = 19; rssi del nodo= 0 con rssi=-69
DEBUG (1): beacon from parent
DEBUG (0): Sent beacon number  140..
DEBUG (2): child seen: 1
DEBUG (3): HANDOVER >>>>>  H_UPD ricevuto con upd_bit = 0...
```

5.3 Topologia ad Albero. Funzionamento in Simulazione

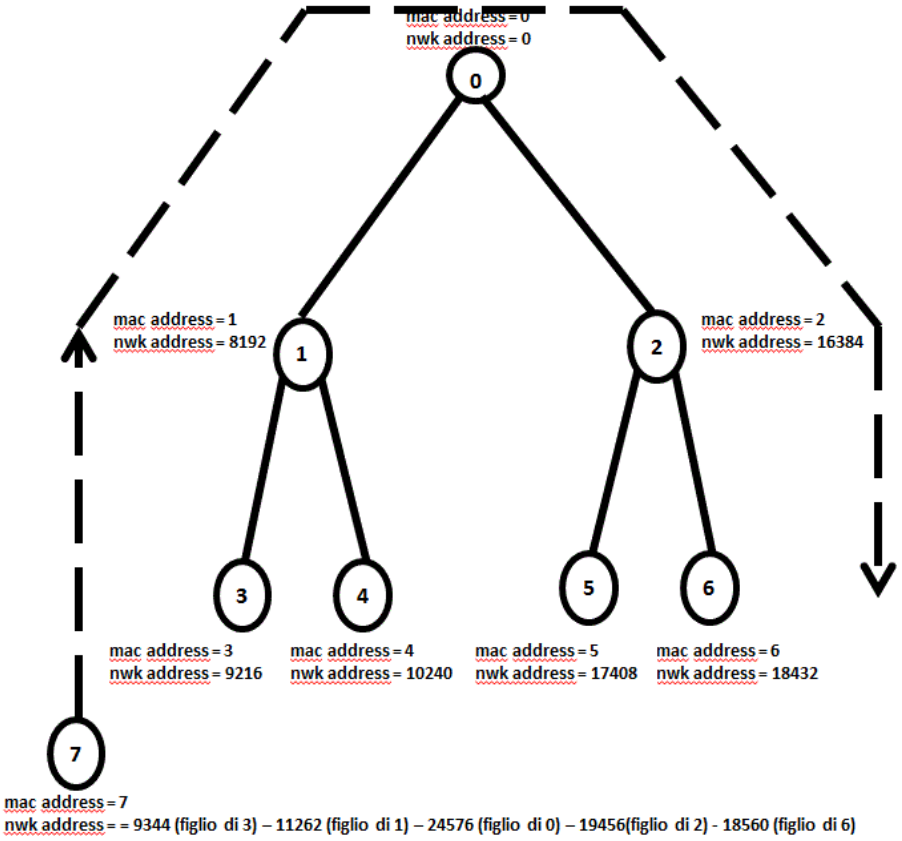

Figura 5.6 : Topologia ad Albero.

```
Simulation time : 0:1:48.108398437
DEBUG (3): Received request from node 7
DEBUG (3): AssociationMantainerP__processAssociationRequest: INDIRIZZI DISPONIBILI: 7
MAX_CHILD_PER_NODE: 7
MAX_CHILD_PER_NODE/4)+1: 2
DEBUG (3): Assegno l'indirizzo 9344
DEBUG (3): sending a response to 7 assigning 9344 and child mask 1
DEBUG (3): Starting confirm timeout timer...
DEBUG (6): child seen: 0
DEBUG (3): associazione avvenuta con 7

Simulation time : 0:2:58.101562500
DEBUG (7): HANDOVER >>>>> Handover timer fired!
DEBUG (7): HANDOVER >>>>> rssi del vicino 8192 : -25 è migliore di quello del padre 9216 : -41
DEBUG (7): HANDOVER >>>>> invio la richiesta di handover
DEBUG (7): beacon timer stoppedd...
```

```
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (7): HANDOVER >>>>> Sending a request to node 8192
DEBUG (0): Checking Tables...
DEBUG (1): HANDOVER >>>>> Received request of handover from node 7
DEBUG (1): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=2,
req_nod_id=7, assigned_addr=11264, res_nod_nwk_addr=9344
DEBUG (1): beacon timer stoppedd...
DEBUG (1): HANDOVER >>>>> sending a response to 7 con network addr: 9344 assigning 11264 and
child mask 4
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (5): Checking Tables...
DEBUG (7): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (7): HANDOVER >>>>> il mio vecchio indirizzo è 9344
DEBUG (7): HANDOVER >>>>> handover andato a buon fine con il nodo 8192
DEBUG (7): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (7): HANDOVER >>>>> Sending H_UPD to node 9216 con upd_bit = 0
DEBUG (7): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (1): HANDOVER >>>>> invio dell H_RES avvenuto...


Simulation time : 0:3:30.100585937
DEBUG (7): HANDOVER >>>>> Handover timer fired!
DEBUG (7): HANDOVER >>>>> rssi del vicino 0 : -23 è migliore di quello del padre 8192 : -40
DEBUG (7): HANDOVER >>>>> invio la richiesta di handover
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (7): HANDOVER >>>>> Sending a request to node 0
DEBUG (3): Checking Tables...
DEBUG (0): HANDOVER >>>>> Received request of handover from node 7
DEBUG (0): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=2,
req_nod_id=7, assigned_addr=24576, res_nod_nwk_addr=11264
DEBUG (0): beacon timer stoppedd...
DEBUG (0): HANDOVER >>>>> sending a response to 7 con network addr: 11264 assigning 24576 and
child mask 4
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (5): Checking Tables...
DEBUG (7): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (7): HANDOVER >>>>> il mio vecchio indirizzo è 11264
DEBUG (7): HANDOVER >>>>> handover andato a buon fine con il nodo 0
DEBUG (7): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (7): HANDOVER >>>>> Sending H_UPD to node 8192 con upd_bit = 0
DEBUG (7): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (0): HANDOVER >>>>> invio dell H_RES avvenuto...


Simulation time : 0:4:18.101562500
DEBUG (7): HANDOVER >>>>> Handover timer fired!
DEBUG (7): HANDOVER >>>>> rssi del vicino 16384 : -26 è migliore di quello del padre 0 : -39
DEBUG (7): HANDOVER >>>>> invio la richiesta di handover
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (3): Checking Tables...
DEBUG (7): HANDOVER >>>>> Sending a request to node 16384
DEBUG (2): HANDOVER >>>>> Received request of handover from node 7
DEBUG (2): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=2,
req_nod_id=7, assigned_addr=19456, res_nod_nwk_addr=24576
DEBUG (2): beacon timer stoppedd...
DEBUG (2): HANDOVER >>>>> sending a response to 7 con network addr: 24576 assigning 19456 and
child mask 4
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (5): Checking Tables...
DEBUG (7): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (7): HANDOVER >>>>> il mio vecchio indirizzo è 24576
DEBUG (7): HANDOVER >>>>> handover andato a buon fine con il nodo 16384
DEBUG (7): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (7): HANDOVER >>>>> Sending H_UPD to node 0 con upd_bit = 0
DEBUG (7): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (2): HANDOVER >>>>> invio dell H_RES avvenuto...
```

```
Simulation time : 0:5:0.100128165
DEBUG (7): HANDOVER >>>>> Handover timer fired!
DEBUG (7): HANDOVER >>>>> rssi del vicino 18432 : -11 è migliore di quello del padre 16384 : -
35
DEBUG (7): HANDOVER >>>>> invio la richiesta di handover
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (7): HANDOVER >>>>> Sending a request to node 18432
DEBUG (3): Checking Tables...
DEBUG (5): Checking Tables...
DEBUG (2): Checking Tables...
DEBUG (0): child seen: 3
DEBUG (0): Checking Tables...
DEBUG (1): Checking Tables...
DEBUG (6): Checking Tables...
DEBUG (7): rssi_count = 24; rssi del nodo= 0 con rssi=-57
DEBUG (2): beacon from parent
DEBUG (1): beacon from parent
DEBUG (0): Sent beacon number  146..
DEBUG (4): Checking Tables...
DEBUG (6): HANDOVER >>>>> Received request of handover from node 7
DEBUG (6): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=0,
req_nod_id=7, assigned_addr=18560, res_nod_nwk_addr=19456
DEBUG (6): beacon timer stoppedd...
DEBUG (6): HANDOVER >>>>> sending a response to 7 con network addr: 19456 assigning 18560 and
child mask 1
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (7): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (7): HANDOVER >>>>> il mio vecchio indirizzo è 19456
DEBUG (7): HANDOVER >>>>> handover andato a buon fine con il nodo 18432
DEBUG (7): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (7): HANDOVER >>>>> Sending H_UPD to node 16384 con upd_bit = 0
DEBUG (7): HANDOVER >>>>> H_UPD inviato!!!
```

5.4 Topoligia ad Albero Alternativa. Funzionamento in Simulazione

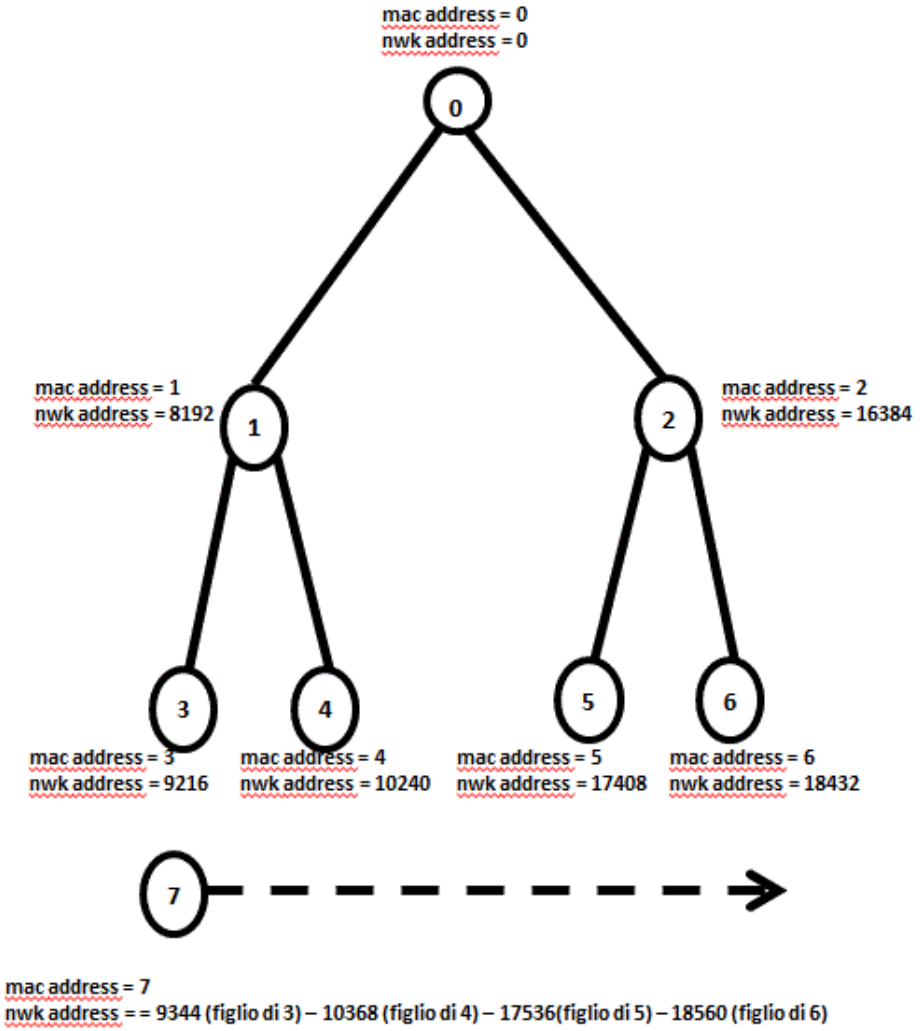

Figura 5.7 : Topologia ad Albero Alternativa.

```
Simulation time : 0:1:48.106445312
DEBUG (3): Received request from node 7
DEBUG (3): AssociationMantainerP__processAssociationRequest: INDIRIZZI DISPONIBILI: 7
MAX_CHILD_PER_NODE: 7
MAX_CHILD_PER_NODE/4)+1: 2
DEBUG (3): Assegno l'indirizzo 9344
DEBUG (3): sending a response to 7 assigning 9344 and child mask 1


Simulation time : 0:3:38.104492187
DEBUG (7): HANDOVER >>>>> Handover timer fired!
DEBUG (7): HANDOVER >>>>> rssi del vicino 10240 : -24 è migliore di quello del padre 9216 : -37
DEBUG (7): HANDOVER >>>>> invio la richiesta di handover
DEBUG (7): beacon timer stoppedd...
```

```
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (7): HANDOVER >>>>> Sending a request to node 10240
DEBUG (7): beacon from parent
DEBUG (7): rssi_count = 14; rssi del nodo= 3 con rssi=-32
DEBUG (3): Sent beacon number  78..
DEBUG (4): HANDOVER >>>>> Received request of handover from node 7
DEBUG (4): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=0,
req_nod_id=7, assigned_addr=10368, res_nod_nwk_addr=9344
DEBUG (4): beacon timer stoppedd...
DEBUG (4): HANDOVER >>>>> sending a response to 7 con network addr: 9344 assigning 10368 and
child mask 1
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (7): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (7): HANDOVER >>>>> il mio vecchio indirizzo è 9344
DEBUG (7): HANDOVER >>>>> handover andato a buon fine con il nodo 10240
DEBUG (7): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (7): HANDOVER >>>>> Sending H_UPD to node 9216 con upd_bit = 0
DEBUG (7): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (4): HANDOVER >>>>> invio dell H_RES avvenuto...


Simulation time : 0:4:20.113281250
DEBUG (7): HANDOVER >>>>> Handover timer fired!
DEBUG (7): HANDOVER >>>>> rssi del vicino 17408 : -30 è migliore di quello del padre 10240 : -
43
DEBUG (7): HANDOVER >>>>> invio la richiesta di handover
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (3): Checking Tables...
DEBUG (7): HANDOVER >>>>> Sending a request to node 17408
DEBUG (4): Checking Tables...
DEBUG (5): HANDOVER >>>>> Received request of handover from node 7
DEBUG (5): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=0,
req_nod_id=7, assigned_addr=17536, res_nod_nwk_addr=10368
DEBUG (5): beacon timer stoppedd...
DEBUG (5): HANDOVER >>>>> sending a response to 7 con network addr: 10368 assigning 17536 and
child mask 1
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (7): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (7): HANDOVER >>>>> il mio vecchio indirizzo è 10368
DEBUG (7): HANDOVER >>>>> handover andato a buon fine con il nodo 17408
DEBUG (7): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (7): HANDOVER >>>>> Sending H_UPD to node 10240 con upd_bit = 0
DEBUG (7): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (5): HANDOVER >>>>> invio dell H_RES avvenuto...


Simulation time : 0:4:42.101516752
DEBUG (7): HANDOVER >>>>> Handover timer fired!
DEBUG (7): HANDOVER >>>>> rssi del vicino 18432 : -13 è migliore di quello del padre 17408 : -
38
DEBUG (7): HANDOVER >>>>> invio la richiesta di handover
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (4): Checking Tables...
DEBUG (7): HANDOVER >>>>> Sending a request to node 18432
DEBUG (0): child seen: 3
DEBUG (2): child seen: 3
DEBUG (6): HANDOVER >>>>> Received request of handover from node 7
DEBUG (6): HANDOVER >>>>> Questi sono i dati registrati nel nodo richiesto: iCont=0,
req_nod_id=7, assigned_addr=18560, res_nod_nwk_addr=17536
DEBUG (6): beacon timer stoppedd...
DEBUG (6): HANDOVER >>>>> sending a response to 7 con network addr: 17536 assigning 18560 and
child mask 1
DEBUG (7): beacon timer stoppedd...
DEBUG (7): HANDOVER >>>>> H_REQ send done, waiting for new parent response...
DEBUG (2): beacon from parent
DEBUG (1): beacon from parent
DEBUG (0): Sent beacon number  80..
DEBUG (5): Checking Tables...
DEBUG (6): beacon from parent
DEBUG (5): beacon from parent
```

```
DEBUG (2): Sent beacon number  77..
DEBUG (7): HANDOVER >>>>>  H_RES ricevuto...associazione in corso
DEBUG (7): HANDOVER >>>>> il mio vecchio indirizzo è 17536
DEBUG (7): HANDOVER >>>>> handover andato a buon fine con il nodo 18432
DEBUG (7): HANDOVER >>>>> Segnalo al vecchio padre che non sono più suo figlio!
DEBUG (7): HANDOVER >>>>> Sending H_UPD to node 17408 con upd_bit = 0
DEBUG (7): HANDOVER >>>>> H_UPD inviato!!!
DEBUG (6): HANDOVER >>>>> invio dell H_RES avvenuto...
```

5.5 Topoligia Lineare vs Topologia ad Albero. Risultati Raccolti

Per testare l'efficienza del protocollo di mobilità HAT Mobile sulla nuova versione protocollare sono stati fatti una serie di test col simulatore Tossim.
E' stato creato un traffico di pacchetti, di messaggi UNICAST dal PAN Coordinator al nodo mobile al fine di contare quanti pacchetti venivano effettivamente ricevuti dal nodo mobile a fronte di quelli inviati.

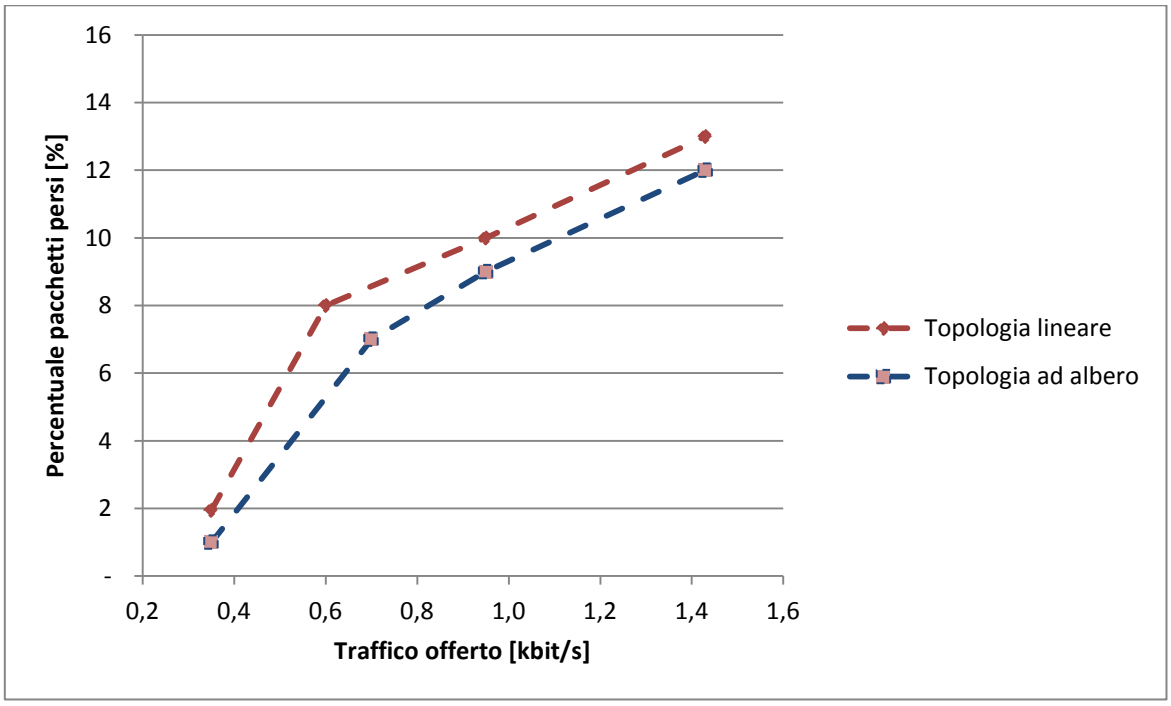

Figura 5.8 : Curva Percentuale pacchetti persi in funzione del traffico offerto.

Si osserva come la perdita di pacchetti in una topologia ad albero è leggermente minore rispetto a quella in una topologia lineare. Questo perchè nella topologia ad albero il nodo mobile è mediamente ad un numero minore di hop dal Pan Coordinator, mentre in quella lineare può anche trovarsi a cinque hop di distanza. In altre parole l'instradamento su più hop comporta una maggiore perdita di pacchetti.

In ogni caso l'andamento della funzione è monotono crescente; ciò è dovuto al fatto che all'aumentare della frequenza trasmissiva si ha un aumento proporzionale della percentuale di pacchetti persi. Ciò è ipotizzabile in quanto è prevedibile che le procedure di handover provochino una perdita di pacchetti che cresce con il numero di pacchetti in circolazione.

Confrontando l'efficienza del protocollo HAT Mobile, capace di ottimizzare maggiormente le procedure di handover in caso di mobilità di nodi, rispetto alla versione HAT classica , si nota come la curva dei pacchetti persi di HAT Mobile stia sotto quella di HAT.

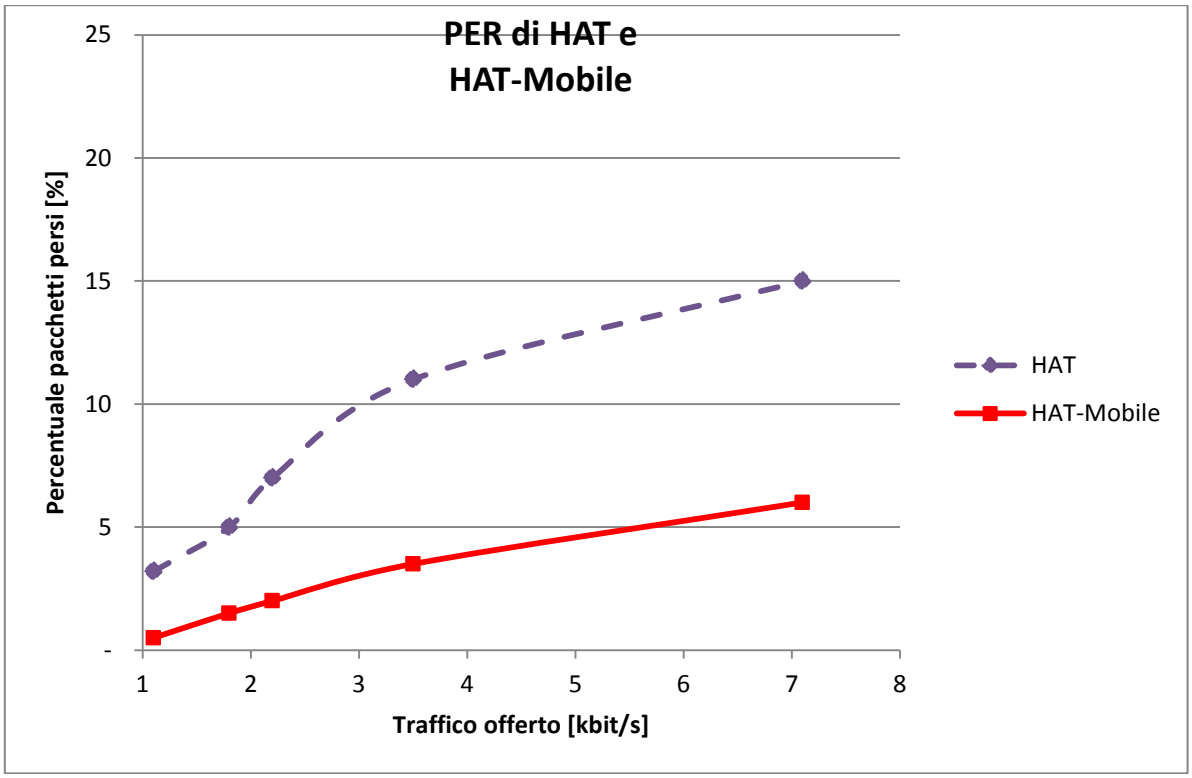

Figura 5.9 Topologia Lineare vs Topologia ad Albero. Risultati Raccolti

Capitolo 6

Conclusioni e Sviluppi Futuri

A conclusione di questa Tesi, si possono trarre varie considerazioni su quali possibili sviluppi futuri questo lavoro potrà avere.
Principalmente si può pensare di adattare il codice per ampliare il protocollo di mobilità al caso di uno scenario multi-albero.
Vediamo nel dettaglio in che modo.

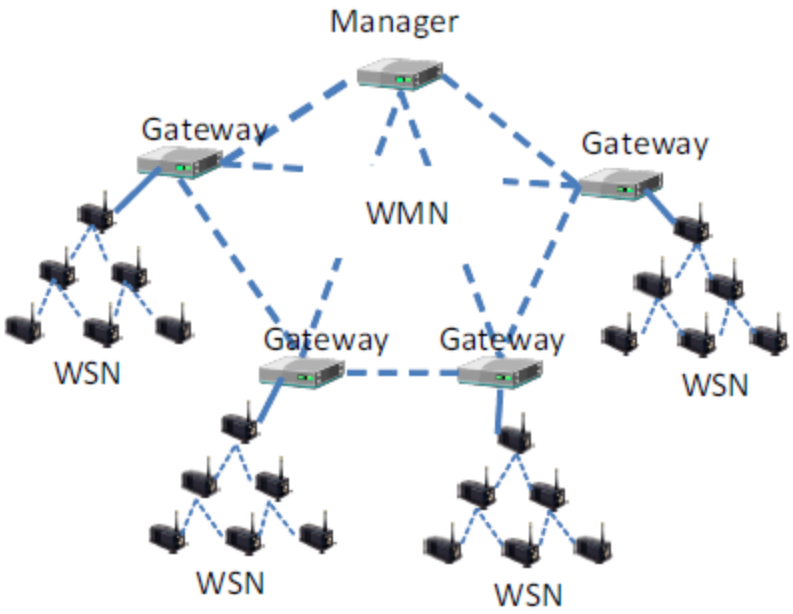

Figura 6.1 : Scenario MultiAlbero : Più WSN interconnesse mediante Wireless Mesh Network

Si possono proporre due strade distinte, una più esplicita ma con più difetti, una implicita e che presenta un vantaggio chiave.
Nella prima strada bisogna considerare di intervenire su due livelli.
Primo Livello. E' un livello più "basso", si intende a Livello di Codice sensori, operando in TinyOS\NesC.
Secondo Livello. E' un livello più "alto", si intende a livello Middleware, Codice per i gateway, operando in Java.
A livello sensori bisogna implentare:

-Sfruttatamento nella struttura dati nwk_hdr_pointer presente nel modulo NetworkP della variabile nwk_hdr_pointer->nwk_id=NET_ID , usando appunto un valore NET_ID per ogni diverso albero.
-Uso del campo net_id nei beacon che i nodi si scambiano.
-Aggiunta di un messaggio di tipo TO_PAN_COORDINATOR che il nodo mobile deve mandare al suo pan coordinator per avvisare dell'imminente cambio d'albero.
Poi questo messaggio di basso livello deve essere mappato in un messaggio di livello Java dal PAN al suo GW (interconnessi tramite seriale).
A livello gateway bisogna implementare:
-a fronte del messaggio che arriva dal nodo mobile il gateway deve cancellare dal suo database le informazioni inerenti il nodo mobile.
Svantaggi : presuppone l'intevento sul codice a livello sensori, TinyOS, e presuppone l'abilità del nodi mobile di capire quando si sta allontanando dal suo vecchio albero in modo da avvisarlo per tempo.

Vediamo ora la seconda strada per gestire la mobilità multi-albero.
Si intende l'associazione del nodo mobile nel nuovo albero come una qualsiasi associazione standard. Quando il dispositivo mobile si riassocia con un nodo ancora del nuovo albero, quest'ultimo deve accorgersi che proveniva da un albero precedente (e non è invece un associazione classica a quel albero) .
Il nodo ancora può fare ciò mandando un messaggio di interrogazione al suo gateway che a sua volta deve chiedere agli altri GW del sistema se quel nodo mobile apparteneva precedentemente ad un altro albero.
In questo caso è il GW nuovo che si accorge di un nuovo dispositivo sotto la sua gestione ed è lui che avvisa il vecchio GW esplicitamente.
In seguito a questo messaggio "tree-handoff" da gateway nuovo a gateway vecchio, quest'ultimo può completamente cancellare dal suo database ogni traccia del nodo mobile che è quindi completamente disassociato.
Vantaggi: Non c'è bisogno di scrivere alcun codice per i sensori, lato TinyOS.
Si Riesce a tenere la Mobilità Multi-Albero, Multi Gateway intervendo solo a livello Middleware , lavorando solo sul codice Java.

In ogni caso lo strumento che ci permette di interconnettere, gestire e integrare diverse reti di sensori è quello del Middeware MobiWSN [2]. MobiWSN è un'architettura multilivello per l'interconnessione di numerose reti di sensori mediante una rete di backbone di tipo wireless mesh. MobiWSN presenta ancora alcune limitazioni dovute alla sua struttura centralizzata che ne riduce notevolmente la robustezza e la scalabilità. Inoltre, la mancanza di un servizio di Publish/Subscribe costringe le applicazioni client, che interagiscono con MobiWSN, a

periodiche operazioni di polling per ottenere informazioni aggiornate sullo stato delle reti di sensori.

Eventualmente una versione migliorata da utilizzare per gestire lo scenario Multi Albero è DTAC (Distributed Topology-Aware Chord) [10], un'architettura distribuita basata su Chord. DTAC è completamente distribuito, scalabile e gestito in maniera autonoma; esso si adatta automaticamente all'arrivo, alla partenza ed al fallimento dei nodi. DTAC offre tutte le funzionalità di una qualunque altra DHT, abilita la creazione di un middleware completamente distribuito e facilita l'integrazione di numerose reti di sensori, garantendo elevata robustezza. Contemporaneamente, lo sviluppo di un sistema di Publish/Subscribe basato su DAC (Distributed Asynchronous Collection) fornisce uno strumento di massima espressività per le applicazioni client, che possono sottoscriversi agli eventi di interesse in maniera semplice ed essere notificati “non appena" essi accadano.

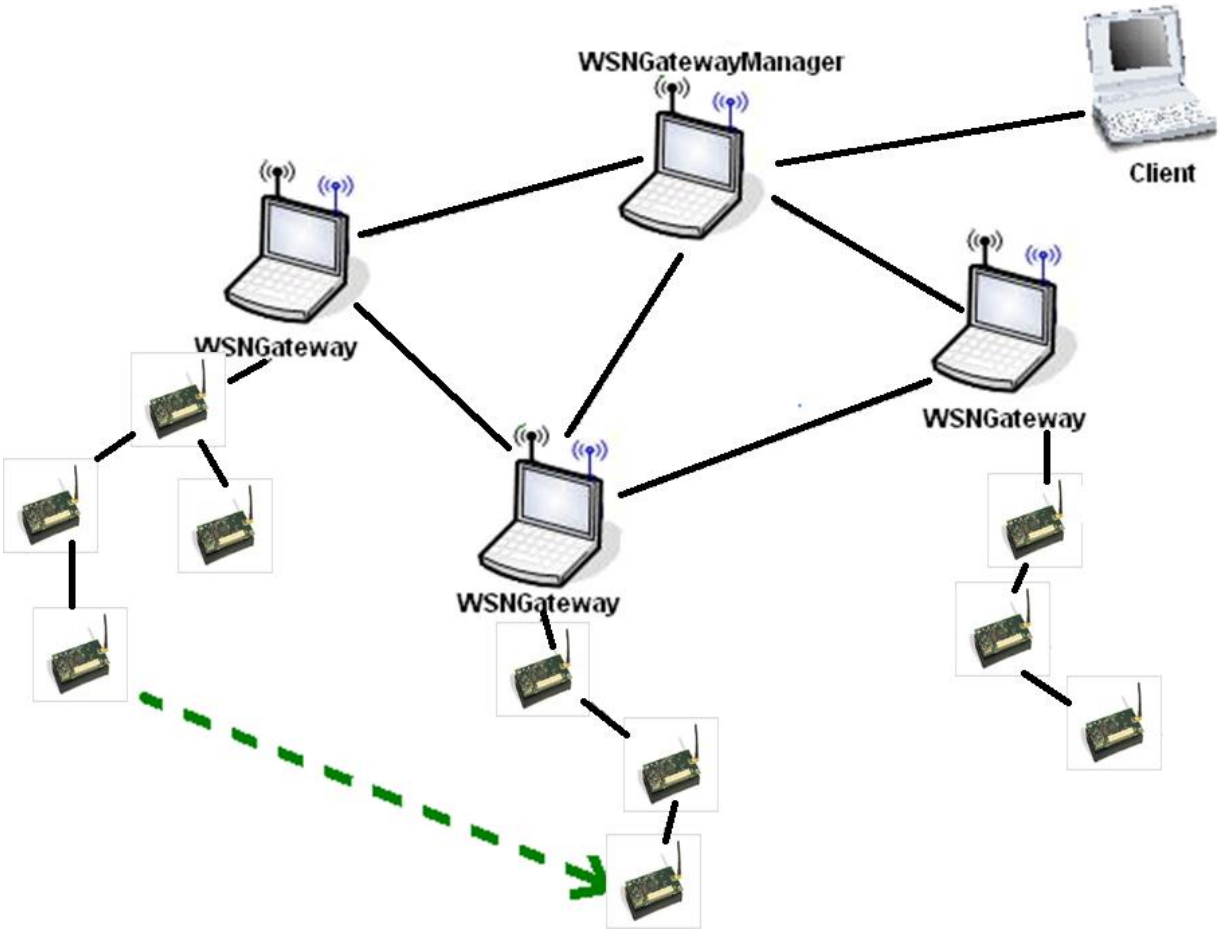

Figura 6.2 :Mobilità Inter-Albero, mobilità tra due diverse reti di sensore.

Bibliografia

[1] Sergio Guglielmi, *Implementazione e Valutazione di un Protocollo di Gestione della mobilità per reti di sensori*, Politecnico di Milano, 2009.

[2] Alessandro Laurucci, *MobiWSN: un middleware dinamico e flessibile per la gestione di reti ibride di sensori mobile*, Politecnico di Milano, 2008

[3] Standard IEEE 802.15.4 : http://www.ieee802.org/15/pub/TG4.html

[4] Z.B. Alliance. *Zigbee specification version 1.0*, 2004.

[5] Università di Berkeley in collaborazione con United States Marine Corps Air/Ground Combat Center (MCA-GCC) – applicazione militare.

[6] *Automated Local Evaluation in Real Time (ALERT)*. The Pima County Regional Flood Control District." http://www.rfcd.pima.gov/wrd/alertsys/.

[7] David Malan, Thaddeus FulfordJones, Matt Welsh, and Steve Moulton, *CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency MedicalCare*. International Workshop on Wearable and Implantable Body Sensor Networks, 2004.

[8] LAURA : http://laura.como.polimi.it/index.php

[9] Davide Riveran, *Progetto e implementazione di un'architettura ibrida per reti di sensori*, Politecnico di Milano, 2007.

[10] Francesco Borrello, *Progetto ed implementazione di un sistema distribuito per la gestione di middleware per reti di sensori*, Politecnico di Milano, 2010.

[11] University of California at Berkeley. *Tiny OS*, Website, http://www.tinyos.net/.

[12] Texas Instrument. *CC2420 Datasheet*, Chipcon AS, 2003.

[13] Joseph Polastre. *Design and implementation of wireless sensor networks for habitat monitoring*. University of California at Berkeley, 2003.

[14] Weilian Su, Özgür B. Akan e Erdal Cayirci. *Wireless sensor networks*, Kluwer Academic Publishers, 2*004*.